Simulación de terrenos usando redes neuronales multicapa

Sistemas de Inteligencia Artificial

26/03/2018

Grupo 2
Carla Barruffaldi - 55421
Luciano Bianchi - 56398
Tomás Cerdá - 56281
Marcelo Lynch - 56287

Introducción

Se implementó una red neuronal feedforward multicapa en Octave que permite simular terrenos de diferentes formas a partir de mediciones de altura, longitud y latitud. El objetivo de la red es a partir de dos coordenadas obtener la altura del terreno. La red es entrenada con métodos de *backpropagation* de manera supervisada a partir de cierto conjunto de datos.

Se analizaron redes con distintas morfologías, así como también con distintos parámetros de funcionamiento y de entrenamiento para llegar a la solución óptima dados los datos del terreno propuesto. Al no tratarse de un problema de clasificación, se tuvieron en cuenta distintos parámetros de tolerancia para determinar la correctitud de la predicción de la red neuronal.

Implementación

El sistema desarrollado permite crear perceptrones multicapa, con distintas inicializaciones de pesos iniciales, y posteriormente permite entrenar la red en base a un set de datos de entrenamiento.

Todos los parámetros necesarios para configurar la red se pueden encontrar y personalizar en el archivo de configuración `config.m`.

Función de activación

Las funciones de activación soportadas son la función escalón (`step`), la función lineal (`linear`), la tangente hiperbólica (`tanh`) y la función logística (`logistic`). La misma función, y su correspondiente derivada, se respeta en todas las unidades de la red.

Capas ocultas

Se puede seleccionar una cantidad variable de capas ocultas (o ninguna para un perceptrón simple), con distintas cantidades de unidades en cada una de ellas. Se utiliza la notación `[x1 x2 ... xn]` donde `x1` es la cantidad de neuronas en la primera capa oculta, `x2` en la segunda capa y `xn` en la n-ésima capa oculta.

Función de costo

La función de costo soportada es la de costo cuadrático medio, y la actualización de pesos durante el *backpropagation* se realiza minimizándola.

Inicialización de pesos

Los tipos de inicialización soportados están basados y recomendados en los papers *Efficient Backprop*[1] y *Understanding the Difficulty of Training Deep Feedforward Neural Networks*[2], y son los siguientes:

`xavier:` $W \sim N(\mu = 0, \sigma = 2/(ni + no))$

`xavier_uniform:` $W \sim U(-4\sqrt{6}/\sqrt{ni+no}, 4\sqrt{6}/\sqrt{ni+no})$

`normal:` $W \sim N(\mu = 0, \sigma = \sqrt{ni})$

`uniform:` $W \sim U(-1/\sqrt{ni}, 1/\sqrt{ni})$

`uniform_one:` $W \sim U(-1, 1)$

Donde ni *es la cantidad de entradas de la capa y* no *la cantidad de salidas.*

Con los pesos iniciales se desea optimizar el entrenamiento haciendo que la función sigmoidea en cada unidad sea activada en su región "lineal". Si los pesos son muy pequeños o muy grandes, el resultado será una sigmoidea saturada, con un gradiente pequeño que hará que el aprendizaje sea más lento.

La motivación detrás de las inicializaciones `xavier` y `xavier_uniform` está en evitar la saturación temprana de las neuronas, ya que disminuyen la varianza de los pesos de cada capa durante el proceso de *feedforward* y de *backpropagation*.

Cabe destacar que la inicialización de pesos se realiza de manera pseudoaleatoria, usando una semilla para generarlos.

Entrenamiento

Se implementaron tres algoritmos de backpropagation:

- Tradicional (`vanilla`).
- Adaptativo (`adaptive`), con `cost_interval`, `inc_steps`, `lr_increase`, `lr_decrease_factor` configurables.
    - `cost_interval`: indica cada cuantas iteraciones se comparará el costo de la red. Por ejemplo, si `cost_interval = 5`, cada 5 iteraciones se verificará si el costo aumentó o disminuyó.
    - `inc_steps`: indica cuántas veces debe el costo disminuir antes de aumentar el learning rate. Por ejemplo, si `int_steps = 3`, cuando el costo disminuya en 3 comparaciones consecutivas se aumentará el learning rate.

[1] http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf
[2] http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

    - `lr_increase`: indica cuánto aumentará el learning rate cuando corresponda;
    - `lr_decrease_factor`: indica en cuanto se disminuirá el learning rate al dar con una comparación de costa mayor a la anterior. Por ejemplo, si `lr_decrease_factor = 0.1`, se disminuirá el learning rate un 10%.
- Momentum (`momentum`), con `alfa` configurable, donde `alfa` es el factor por el cual se multiplica el gradiente de pesos de la iteración de entrenamiento previa para sumar en la iteración actual.

En cualquiera de los métodos mencionados es posible configurar un `batch_size` para realizar entrenamiento tipo *batch*. Indicando un `batch_size` de 1, se obtiene entrenamiento *incremental* mientras que indicando un `batch_size` igual o mayor al conjunto de patrones de entrenamiento (si es mayor se trunca al tamaño exacto del conjunto) se obtiene un entrenamiento *batch*. Cualquier otro valor entre 1 y el tamaño del conjunto corresponde a *mini-batch*.

Normalización

Existe la posibilidad de normalizar tanto los datos de la entrada como los de la salida de la red. Esto es, *mapear* el conjunto de números de un rango hacia otro, un método conocido como *Feature Scaling*.

Se tomó la decisión de normalizar ambos sets de datos, por motivos diferentes. En el caso de la entrada, como los datos estaban en un rango de (-4, 4), es deseable llevarlos a un rango de (-1, 1) para disminuir la probabilidad de que se saturen las unidades en la primer capa de la red, ya que multiplicando los pesos por número grandes es probable que la función de activación no esté en su región lineal.

Por otro lado, para la salida, se escalan los datos a un rango menor ya que las imágenes de la tangente hiperbólica y de la función logística no pueden abarcar a todo el rango de los patrones de salida, que en este caso es de (-9, 9). Por esta razón, se mapea el rango de patrones de salida a un intervalo igual al de la imagen de la función de activación elegida para la red.

Para normalizar se utilizó la siguiente expresión. Dado un rango de valores (x, y) que se desean mapear a un rango (a, b), a cada valor z le corresponde c tal que:

$$c = a + \frac{z - x}{y - x} * (b - a)$$

Conjunto de prueba y de entrenamiento

Para la evaluación de la red neuronal, se toma el conjunto de datos y se lo divide en conjuntos de *prueba* y de *entrenamiento*. La cantidad de patrones que van a cada uno de ellos depende del `test_ratio` indicado en la configuración, dónde `test_ratio` representa la tasa de patrones que se destinarán para el conjunto de *prueba*.

Números aleatorios

Debido al uso de números pseudoaleatorios para la inicialización de los pesos de la red y para la elección de los conjuntos de prueba y de entrenamiento, es necesaria una semilla para la ejecución del programa. Se decidió dar la posibilidad al usuario de configurar dicha semilla (`seed` en el archivo de configuración) en lugar de usar algún número autogenerado, ya que puede ser deseable usar varias veces la misma `seed` para poder replicar exactamente una misma red más de una vez. Si se desea utilizar una `seed` diferente cada vez que se ejecuta basta asignarle el valor de la función `time`.

Consideraciones

Dada la enorme cantidad de variaciones posibles que puede tener una red que solucione de manera aceptable este problema, se propone realizar un análisis preliminar que tiene el objetivo de reducir a un número accesible las combinaciones de parámetros que se van a comparar. Con este objetivo, se divide el análisis en dos etapas: la de *selección* y la de *entrenamiento*.

En la etapa de *selección,* se busca determinar la configuración de la red (incluyendo cantidad y forma de las capas ocultas, y función de activación) para resolver de forma óptima el problema en cuestión.

Se asume la forma óptima de resolución del problema aquella que cumpla con la característica de poseer el mínimo número de neuronas posible, pues esto nos asegura un tiempo de ejecución post-entrenamiento que pueda ser acorde al contexto de un videojuego, y asimismo que pueda generalizar el terreno dado en el menor tiempo posible siempre y cuando lo pueda aprender con la tasa de error propuesta.

La tasa de error es el porcentaje de patrones que no son predichos correctamente por la red. Se define un parámetro de tolerancia que se explicará a continuación para determinar si una predicción es acertada o no.

Se toma en cuenta esta tasa de error entendiendo que se quiere determinar la topología de un terreno de un videojuego, por lo que no se busca una alta precisión en la altura de todos los puntos, sino que basta con que la red represente de forma fiable la forma general del mismo. En cambio, se valora más la capacidad de generalización de la red, ya que los puntos conocidos dan una simple idea de la topología del terreno, por lo que se busca que la red represente una gran cantidad de puntos por fuera del conjunto de los puntos conocidos. Como última prioridad se considera al tiempo de entrenamiento, ya que se asume que para un problema como el propuesto no debería ser un limitante el tiempo para desarrollar la red. Además, para la cantidad de datos a analizar, las redes propuestas en general toman, en el peor de los casos, un tiempo en el orden de los minutos para su entrenamiento.

En segundo lugar, en el *entrenamiento*, se analizan los distintos métodos de optimización al algoritmo de *backpropagation* para entrenar la red. Entre ellos incluimos los métodos `vanilla`, `momentum` y `adaptive`, mencionados previamente.

Si bien el resultado final depende tanto de la estructura de la red como del método de backpropagation, el grado de relación es bajo, por lo que se decidió llevar a cabo las etapas del estudio de forma separada. En teoría, una red con una configuración adecuada debe ser capaz de llegar a niveles de aprendizaje y generalización adecuados, independientemente de la forma en la que se la vaya a entrenar (siempre y cuando durante el entrenamiento no se caiga en un mínimo local).

Dicho de otro modo, si existiese un *oráculo* que seleccione los pesos óptimos de cualquier red, se quiere encontrar la configuración más eficiente (en cantidad de unidades y de capas) que solucione el problema con la precisión requerida si se utilizaran los pesos indicados por este oráculo ideal.

Selección

En una primera instancia, se opta por buscar una configuración (arquitectura de la red y función de activación) para la cual se cumplan los requisitos propuestos, es decir, que la red aprenda al menos un porcentaje propuesto del terreno usando la menor cantidad de neuronas y en el menor tiempo posible. Además se decide tomar en cuenta el valor del costo, buscando entre las configuraciones propuestas aquellas que alcanzan el objetivo con mayor costo, pues esto normalmente significa un mayor grado de generalización.

Cantidad de capas y neuronas

Se comenzó la prueba a partir de observar si el terreno puede ser resuelto con una única capa. Para esto se fijaron ciertos parámetros:

- `learning_rate = 0.01`
- `batch_size = 5`
- `test_ratio = 0.35`
- `algorithm = ‘vanilla’`
- `initialization = 'uniform_one'`
- `tolerance = 0.1`
- `max_tries = 1000`
- `min_cost_change = 1e-6`
- `seed = 42`

Notar el alto grado de patrones que se utilizarán como testeo (35%) dejando así un 65% para el entrenamiento. Esto se así porque se busca generalizar lo mayor posible.

Se decidió dar por terminado el test cuando la red logra aprender al menos un 97% de los puntos, en otras palabras, se acepta que la red no pueda resolver hasta un 3% de los puntos. Ésto se expresa en el parámetro `test_error` de la configuración. Esta decisión fue

derivada de que ese porcentaje es correspondiente a los extremos de los puntos dados y no resultan prioritarios a la hora de representar la topología del terreno.

Dado que éste no es un problema de clasificación, se define un parámetro de tolerancia (`tolerance`) con el cual se considera si la salida de la red es correcta o no. Si la diferencia entre la salida de la red y la salida esperada es menor a la tolerancia, entonces se considera que la resolución es acertada para ese patrón.

Finalmente, fue necesario definir un límite capaz de definir si la red se encuentra saturada, lo cual hace que no valga la pena continuar el entrenamiento con dicha configuración. Si bien el tiempo de entrenamiento no es una prioridad, se debe determinar un punto del entrenamiento en el cual la red está tan saturada que no aprenderá más nada en un tiempo razonable. Con el objetivo de finalizar el entrenamiento al llegar a este punto de saturación se determinaron los parámetros de `max_tries` y `min_cost_change`, que representan lo siguiente:

- Si entre dos épocas *consecutivas* de entrenamiento, el *costo de entrenamiento* no disminuyó *al menos* `min_cost_change`, se lo considera como "un intento".
- Si durante `max_tries` intentos *consecutivos* no se logró disminuir el costo al menos `min_cost_change`, se da por terminado el entrenamiento considerándolo un fracaso.

Los valores de estos parámetros se buscaron elegir de forma tal que si la red se encuentra en un *plateau* respecto a la función de costo, se le dé el tiempo suficiente para que pueda salir del mismo.

Una capa oculta

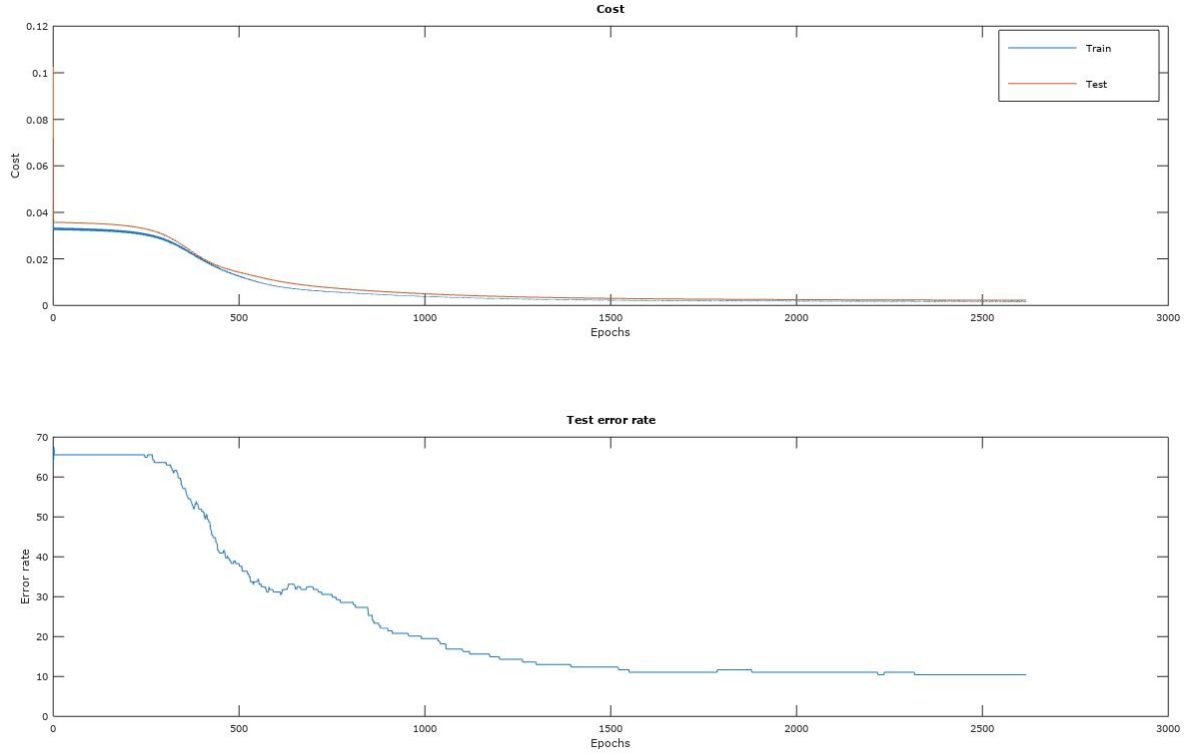
Se evaluaron las funciones `tanh` y `logistic` con una única capa oculta de 30, 60 o 90 neuronas. A continuación se muestran los resultados obtenidos.

# Caso	Función	Neuronas	Error	Costo	Épocas	Tiempo (seg)
1	`tanh`	30	10.4%	17.519×10^{-4}	2618	338
2	`logistic`	30	10.4%	13.804×10^{-4}	2606	363
3	`tanh`	60	9.1%	17.558×10^{-4}	1341	216
4	`logistic`	60	9.1%	15.036×10^{-4}	1940	316
5	`tanh`	90	9.1%	19.607×10^{-4}	2942	363
6	`logistic`	90	98%	1389.72×10^{-4}	1000	18

Tabla 1 - Resultados única capa oculta

En primera instancia se evalúa la red con 30 neuronas. La cantidad de épocas resultó ser elevada, tanto para el caso de la función `tanh` como `logistic`, sin embargo no lograron

cumplir el objetivo propuesto de disminuir el error de testeo a un 3%. En la figura 1 se puede ver que, si bien la red se encontró con un *plateau*, logró salir del mismo. A su vez, se puede ver la estrecha relación entre el costo y la tasa de error en las primeras épocas.



.Figura 1 - Costo y error del caso 1.

Se evalúa entonces para los casos 3 y 4 una capa oculta de 60 neuronas. Los resultados obtenidos son levemente más favorables: se alcanza una menor tasa de error en un menor tiempo y un costo de entrenamiento similar. Sin embargo, sigue estando lejos del objetivo propuesto.

Como en la figura 1, se puede valorar en la figura 2 que si bien la red se encontró con un *plateau* pudo salir del mismo, lo que nos da a entender que los valores de `max_tries` y `min_cost_change` son adecuados. En la figura 3 se puede ver una vez más que hay una estrecha relación entre el costo y la tasa de error en las primeras épocas.

Finalmente, se evalúan las dos funciones con una capa de 90 neuronas pero los resultados no son alentadores: si bien el caso de `tanh` se logró un mayor grado de generalización pues alcanza un error similar con un mayor costo de entrenamiento, continúa lejos del objetivo propuesto. Con esta semilla en particular, para el caso de `logistic`, en la primer época se cae a un mínimo local del cual la red no logra salir, a pesar de que el tamaño del batch no sea elevado. Se comprobó que lo mismo sucede incluso con una metodología incremental.

En general, `tanh` alcanza errores similares a `logistic` con costos más elevados, dando a entender que presenta una mayor capacidad de generalización.

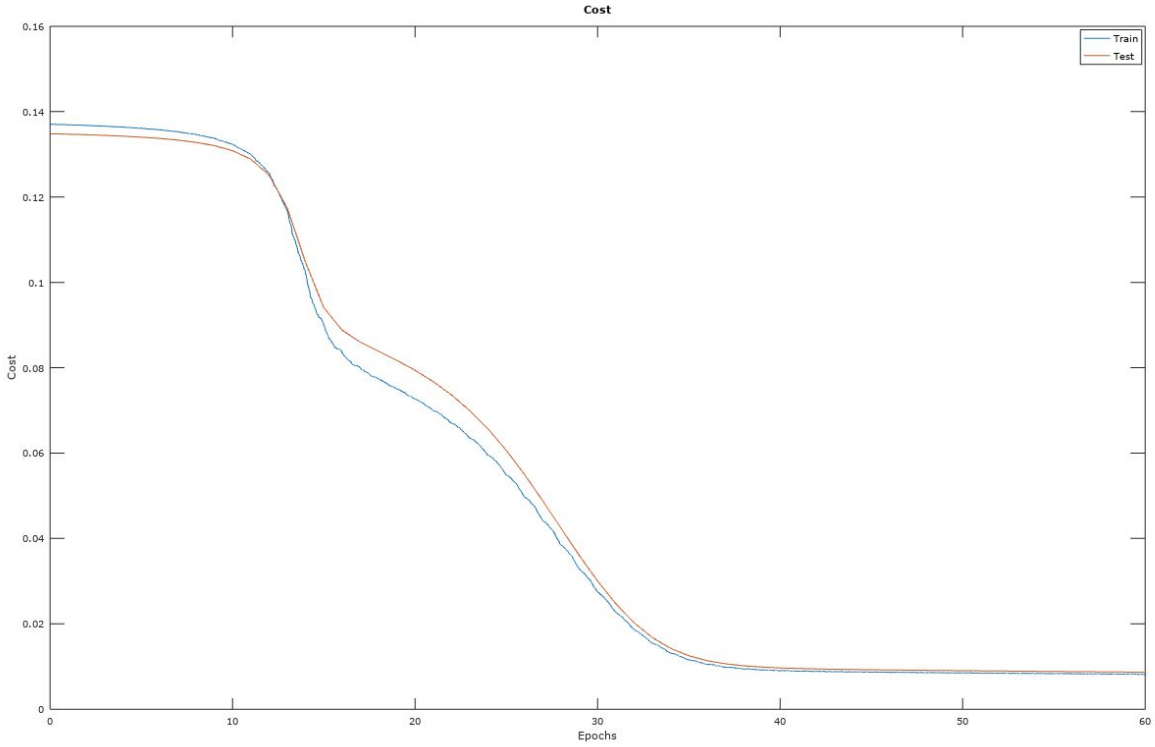


Figura 2 - Costo del caso 4 en las primeras 60 épocas.
Superan dos *plateaus* en las primeras 30 épocas.

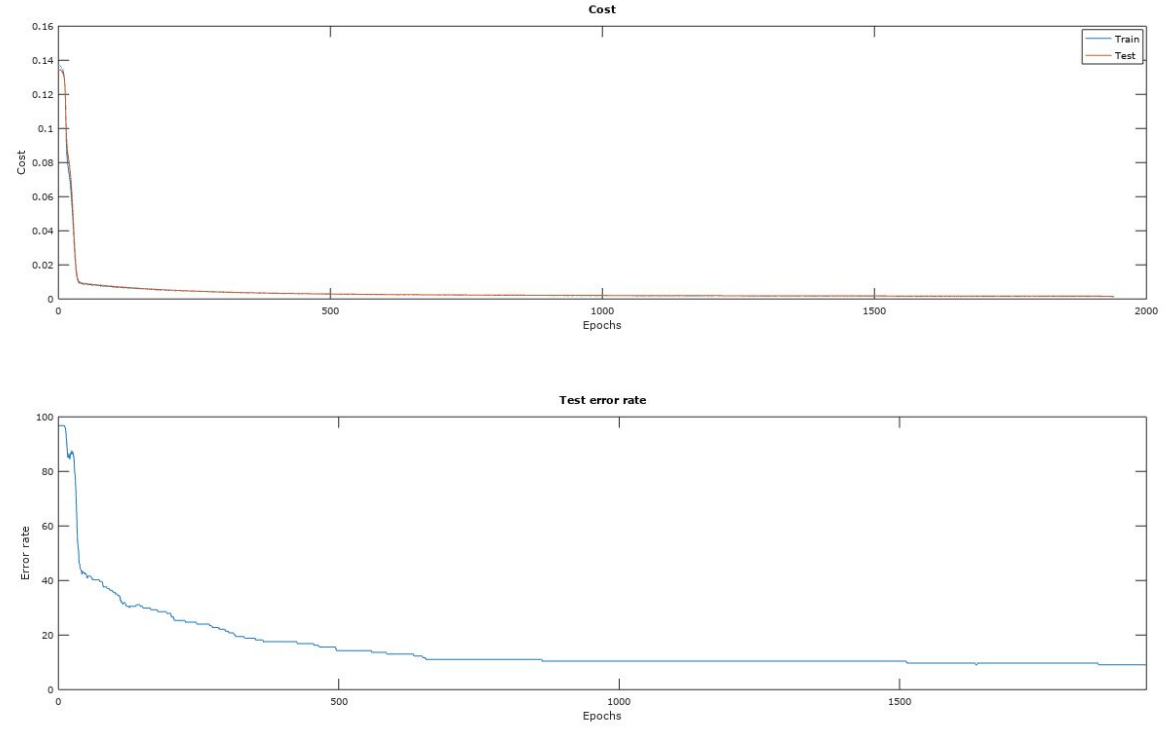


Figura 3 - Costo y error del caso 4.
Hay una estrecha relación entre el costo y tasa de error.

Dado que ninguna configuración representa notables mejores en comparación a las demás, se concluye que una capa (incluso una de 90 neuronas) no es suficiente para almacenar toda la información necesaria del terreno. Si bien se logra llegar a un error del 9%, todavía hay patrones que la red no puede predecir con precisión.

Se decide entonces agregar una segunda capa a la red.

Dos capas ocultas

# Caso	Función	Neuronas	Error	Costo	Épocas	Tiempo (seg)
1	`tanh`	[90 60]	<3%	12.198×10^{-4}	321	59
2	`logistic`	[90 60]	3.2%	5.2397×10^{-4}	1498	415
3	`tanh`	[60 30]	<3%	4.3811×10^{-4}	2492	544
4	`logistic`	[60 30]	7.1%	8.4052×10^{-4}	1753	407
5	**`tanh`**	**[40 20]**	**<3%**	**6.5412×10^{-4}**	**1093**	**135**
6	`logistic`	[40 20]	7.1%	9.7694×10^{-4}	2289	457
7	`tanh`	[40 10]	3.9%	7.5378×10^{-4}	1793	213
8	`logistic`	[40 10]	7.1%	11.971×10^{-4}	2905	552

Tabla 2 - Resultados de pruebas con dos capas ocultas

Se comenzó evaluando 90 neuronas en la primer capa y 60 neuronas en la segunda, con la función de activación `tanh` esperando que estos valores se ajusten a lo necesitado. Exitosamente la red logró llegar a lo propuesto en 321 épocas, con un costo de 12.198×10^{-4} (un grado de generalización relativamente alto), en 59 segundos.

Es notable que `logistic` no logra alcanzar el objetivo propuesto con la misma morfología de red, alcanzando un costo considerablemente menor, dando a entender que la red tiende a memorizar más y generalizar menos con `logistic` que con `tanh`.

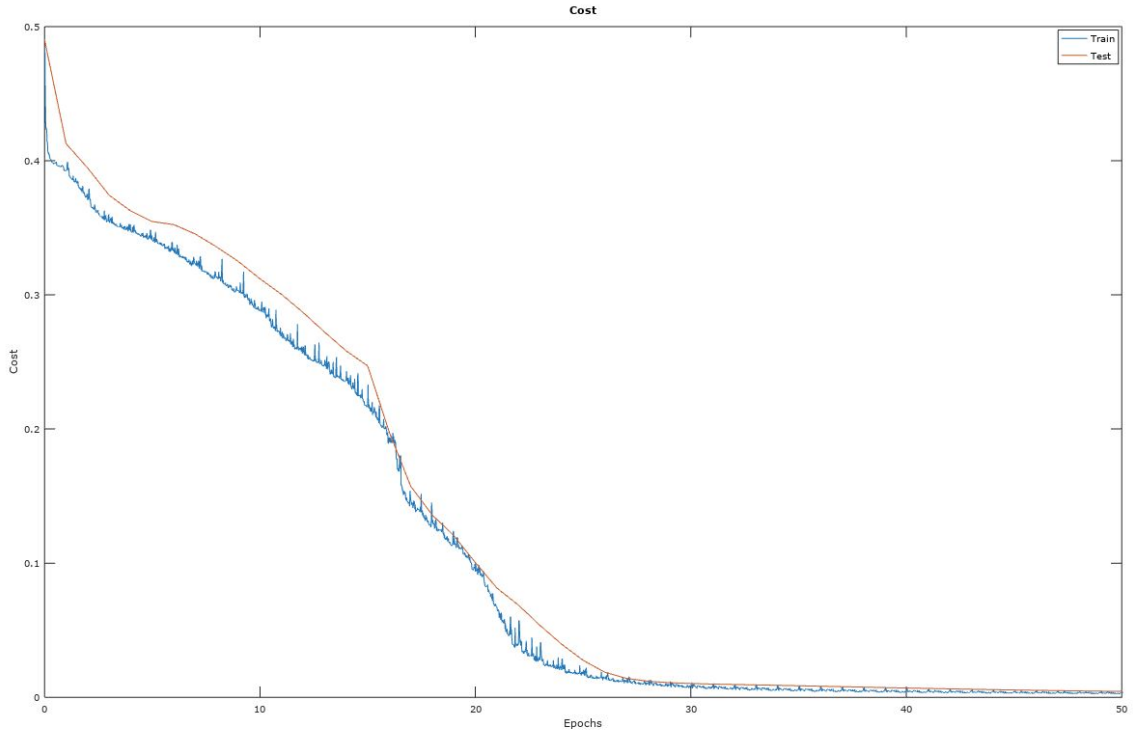


Figura 4 - Costo de las primeras 50 épocas del caso 1.
Se pueden apreciar una mayor amplitud de oscilaciones que cuando había 1 sola capa.

Sabiendo que fue posible llegar a lo propuesto, se propuso descender la cantidad de neuronas en cada capa buscando la de menor cantidad posible.

Nuevamente se llega a lo propuesto con una arquitectura de [60 30] `tanh`, aunque con un grado peor de generalización y un tiempo de entrenamiento mucho más superior.

Luego de evaluar variadas combinaciones, se llegó a la conclusión de que el número conveniente de neuronas era 40 neuronas en la primer capa y 20 neuronas en la segunda. Este resultado fue elegido, ya que la combinación de 40 y 10 no lograba alcanzar el objetivo propuesto, catalogándolo como un caso no exitoso, mientras que las combinaciones [90 60] y [60 30] presentan un mayor número de neuronas, siendo peores según nuestros criterios que priorizan la menor cantidad de neuronas.

Al haber determinado que [40 20] era la mejor morfología, se realizan varias pruebas con distintas `seeds` para corroborar su funcionamiento. En las figuras 5 y 6 se muestran los resultados con diferentes `seeds` para la misma distribución de neuronas. Los resultados parecen ser consistentes.

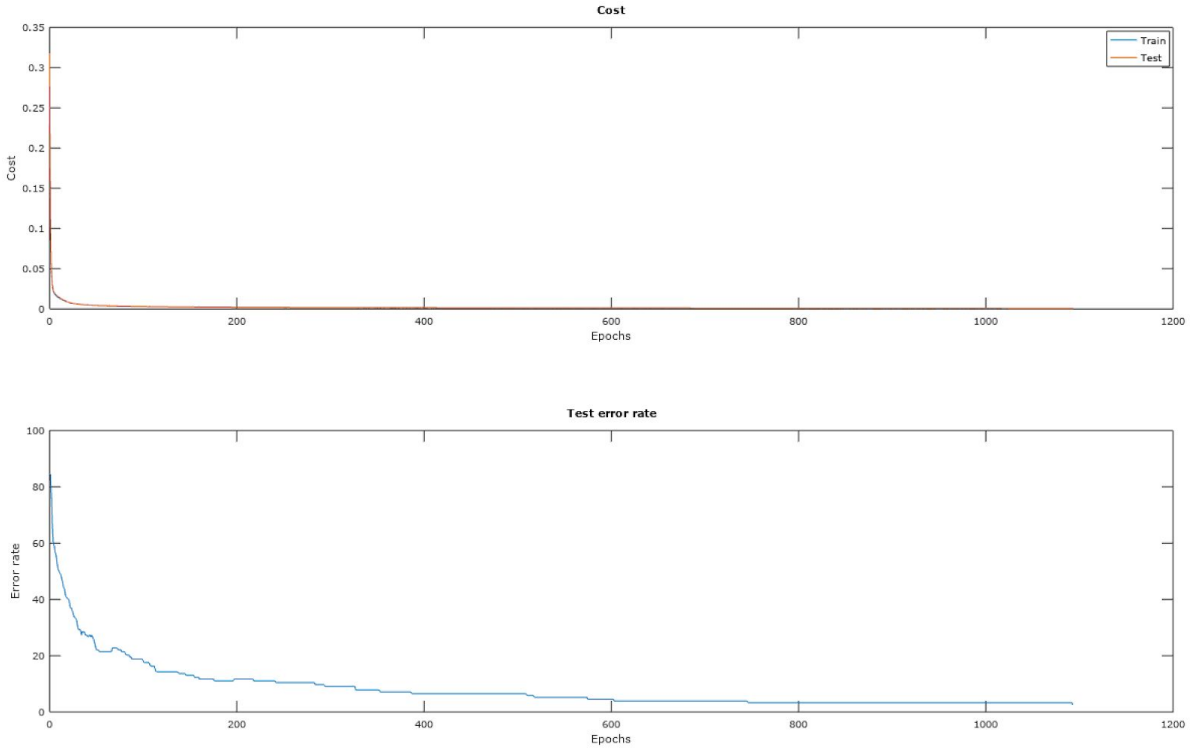


Figura 5 - Costo y error de la configuración ganadora con `seed 42`.

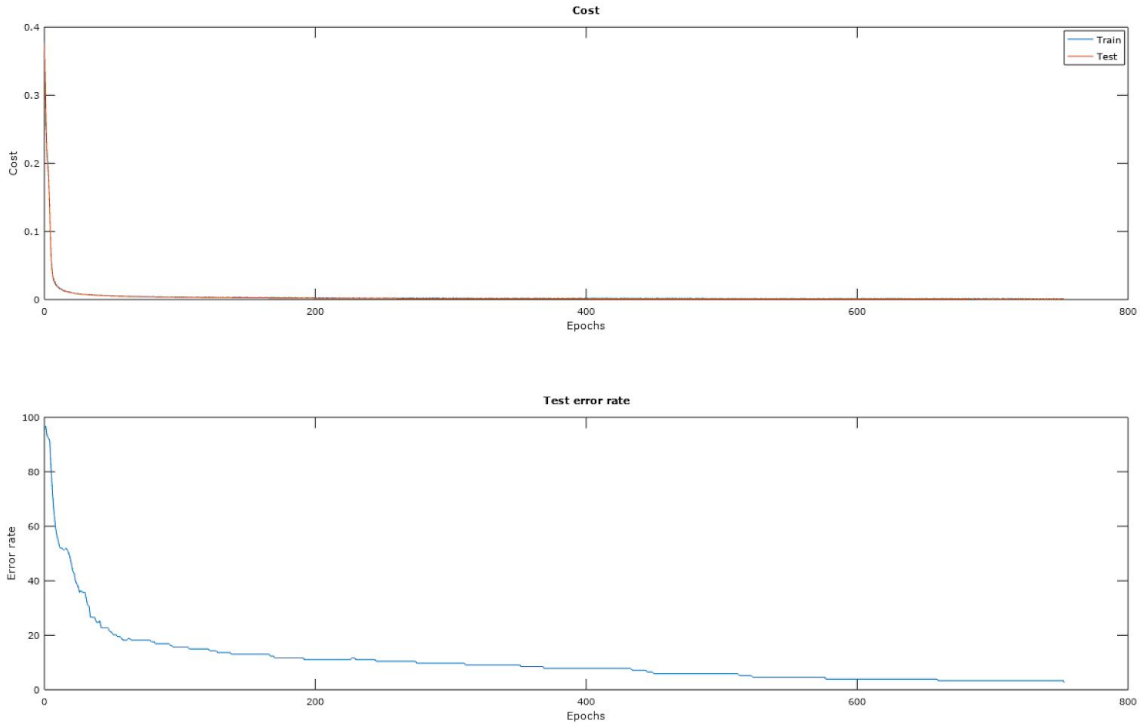


Figura 6 - Costo y error de la configuración ganadora con seed 30834

A partir de estos resultados, se toma la decisión de considerar 40 neuronas en la primer capa y 20 en la segunda como una morfología óptima para la resolución del problema.

Cabe destacar que en ningún caso la función `logistic` alcanza el objetivo propuesto por grandes márgenes excepto en el caso 2. Aún así, en dicho caso el costo es bastante menor al de la función `tanh`. Se elige entonces optar por la tangente hiperbólica debido a que la logística no alcanzó en ningún caso el objetivo propuesto, esbozando una performance según los criterios ya descritos inferiores al de `tanh` tanto para 1 como 2 capas. Otro motivo para la elección de `tanh` fue la velocidad de convergencia. Usando `tanh`, la red llega a un error menor más rápidamente, lo cual también coincide con el comportamiento descrito en *Efficient Backprop[4.4].*

En conclusión, finalizada la etapa de selección, se determinó que la configuración de la red más adecuada para el problema era la de 2 capas ocultas con 40 y 20 neuronas respectivamente, 2 unidades de entrada y una de salida. Todas la red activada con la tangente hiperbólica.

Entrenamiento

A continuación se evaluará la mejor configuración según los criterios ya descritos para los 3 algoritmos de entrenamiento implementados: `vanilla, momentum y adaptive.` En cada uno de ellos se utilizará la red elegida en el proceso de *Selección* de [40 20] neuronas y se variarán los parámetros de `learning rate` y `batch_size` como también los parámetros que sean específicos a un algoritmo en particular.

Dado que la cantidad de iteraciones en cada batch varía según el tamaño del mismo, se cambió el valor de `max_tries` dependiendo del tamaño del batch:

batch_size	max_tries
1	100
5	1000
batch	20000

Tabla 3 - Cambio del parámetro max_tries respecto de batch_size

Vanilla

Siendo este el algoritmo de entrenamiento tradicional, sin optimizaciones, los únicos parámetros de entrenamiento que se pueden variar son `learning_rate` y `batch_size.` Se evalúa entonces la red variando el `learning_rate` entre 0.1 y 0.01 y el `batch_size` entre 1 (incremental), 5 (mini-batch) y batch.

Los casos se enumeran del 1 al 6 y realizaron con un `seed` de 8099 y con la inicialización de pesos `uniform_one.`

Resultados

# Caso	learning rate	batch_size	Error	Costo	Épocas	Tiempo (seg)
1	0.1	1	27%	38.175×10^{-4}	210	212
2	**0.1**	**5**	**<3%**	**8.1321×10^{-4}**	**386**	**71**
3	0.1	*batch*	<3%	7.6641×10^{-4}	17935	135
4	0.01	1	<3%	8.3187×10^{-4}	459	450
5	0.01	5	4.5%	7.4719×10^{-4}	1895	396
6	0.01	*batch*	7.8%	21.318×10^{-4}	22147	166

Tabla 4 - Resultados Vanilla

El caso 2 de un mini-batch de tamaño 5 y un learning rate elevado de 0.1 es considerada la mejor configuración dentro de los 6 casos evaluados.

Si bien el caso 3 tuvo también un buen desempeño, su costo de entrenamiento alcanzado es menor al del caso 2, dando a entender que el caso 2 posee mayor capacidad de generalización, atributo mayormente valorado según los criterios descritos. A su vez, el caso 2 alcanzó el objetivo en casi el 50% del tiempo.

En cuanto al caso 4, si bien fue exitoso y con un costo mayor al del caso ganador, esta diferencia es despreciable en contraste a la diferencia importante en cuanto al tiempo de procesamiento. Notable es también que el caso 2 haya alcanzado el objetivo en una menor cantidad de épocas considerando que al tener un tamaño del batch de 5, hay alrededor un ⅕ de iteraciones en cada época que en el caso incremental.

Esto nos da a entender que el ascenso de estabilidad adquirido debido al aumento del tamaño del batch a 5, permite alcanzar el objetivo propuesto de forma más efectiva con un `learning_rate` de 0.1, en contraposición al `learning_rate` más conservador de 0.01 que requiere el caso incremental.

Como se puede ver en el caso 1, la inestabilidad causada iteración a iteración en el caso incremental con un `learning_rate` elevado de 0.1 provocó que el error de testeo diverja y el costo oscile ampliamente, como se puede ver en las figuras 7 y 8.

Por último, se puede ver en los casos 5 y 6 que un `learning_rate` de 0.01 es demasiado conservador para los casos mini-batch y batch, no pudiendo alcanzar el objetivo propuesto.

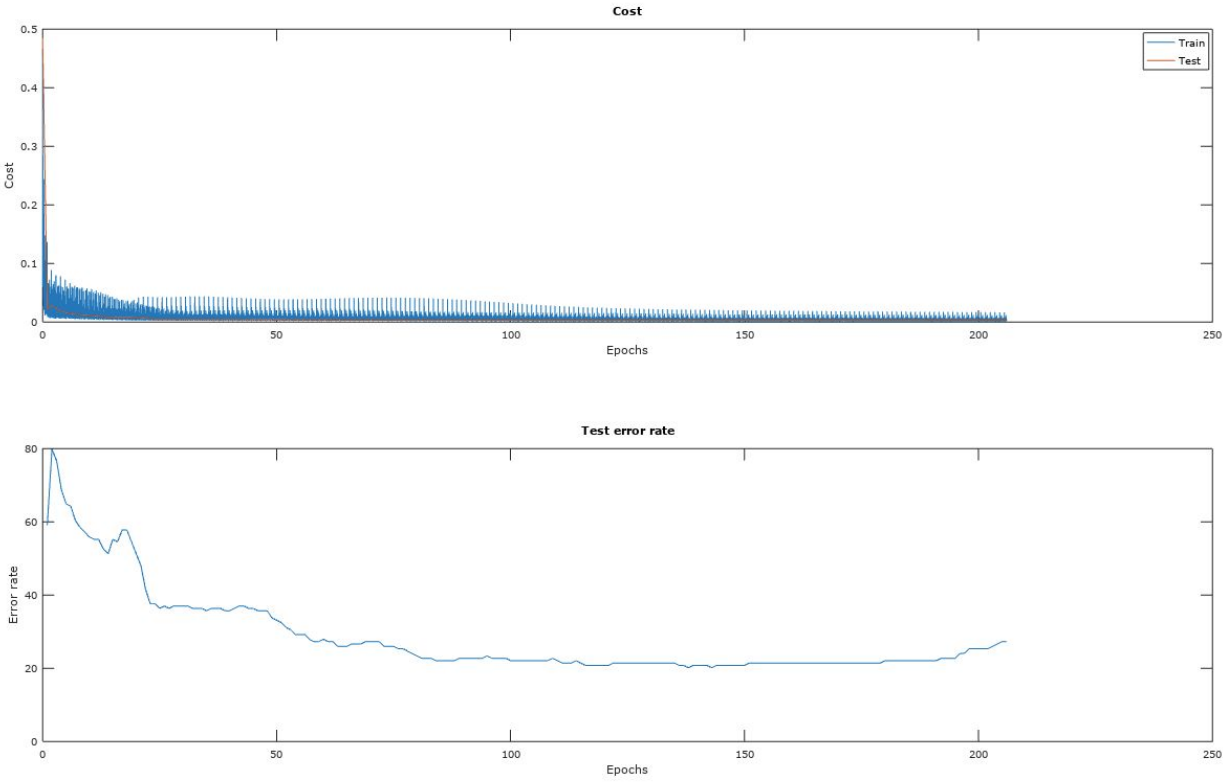


Figura 7 - Costo y error de testeo del caso 1 Vanilla.
Las oscilaciones se mantienen si bien disminuyen su amplitud.
El error de testeo no disminuye y hasta aumenta.

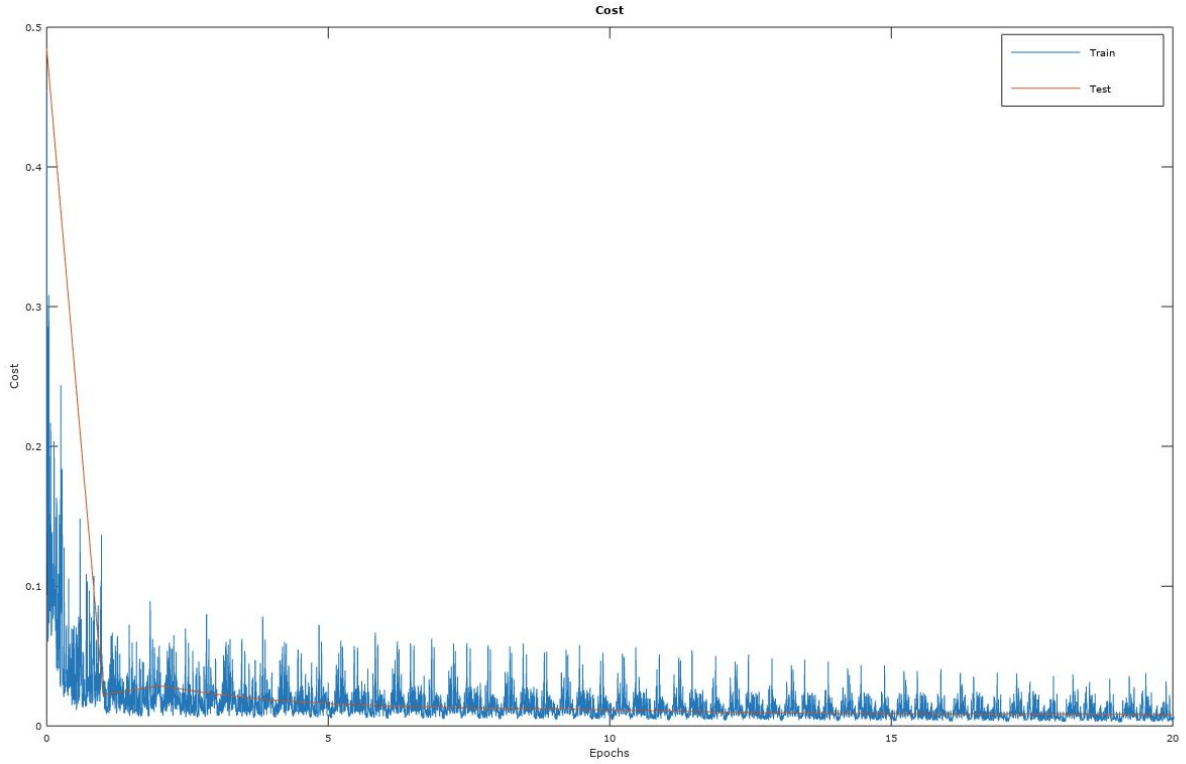


Figura 8 - Costo de las primeras 20 épocas del caso 1 Vanilla, incremental.
Se presentan oscilaciones notables.

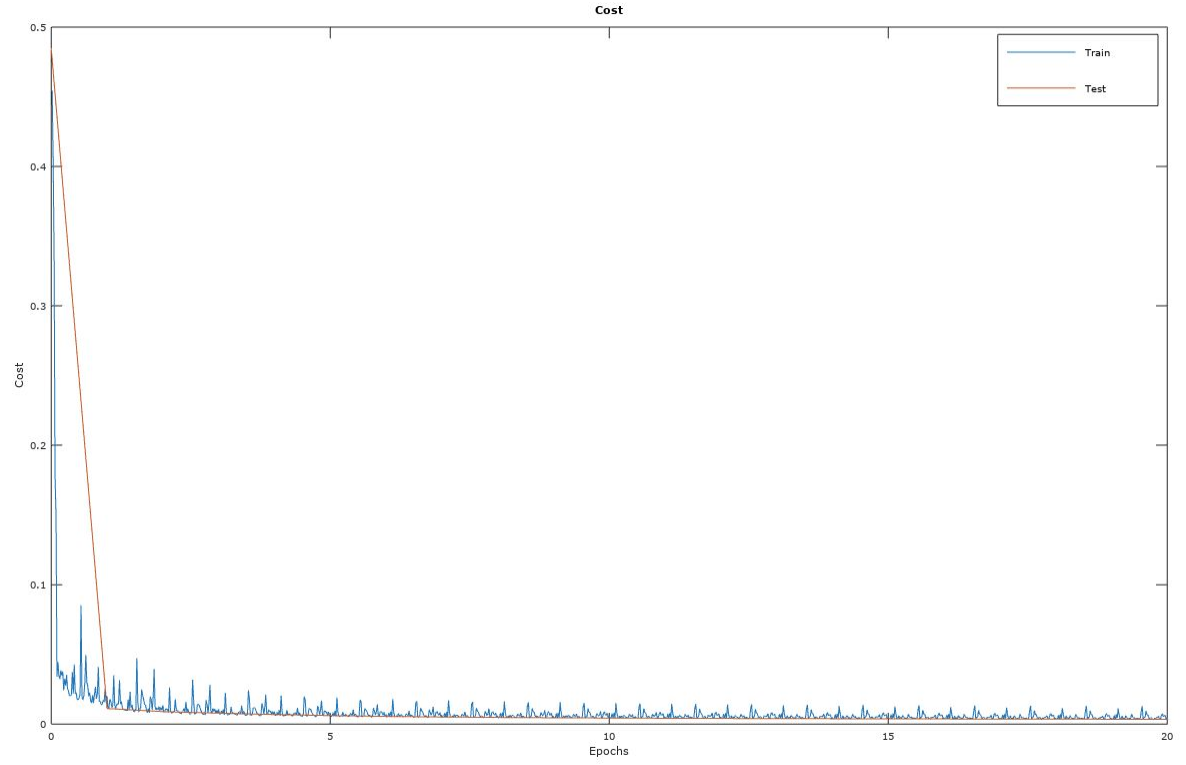


Figura 9 - Costo de las primeras 20 épocas del caso 2 Vanilla, mini-batch
Se presentan oscilaciones leves.

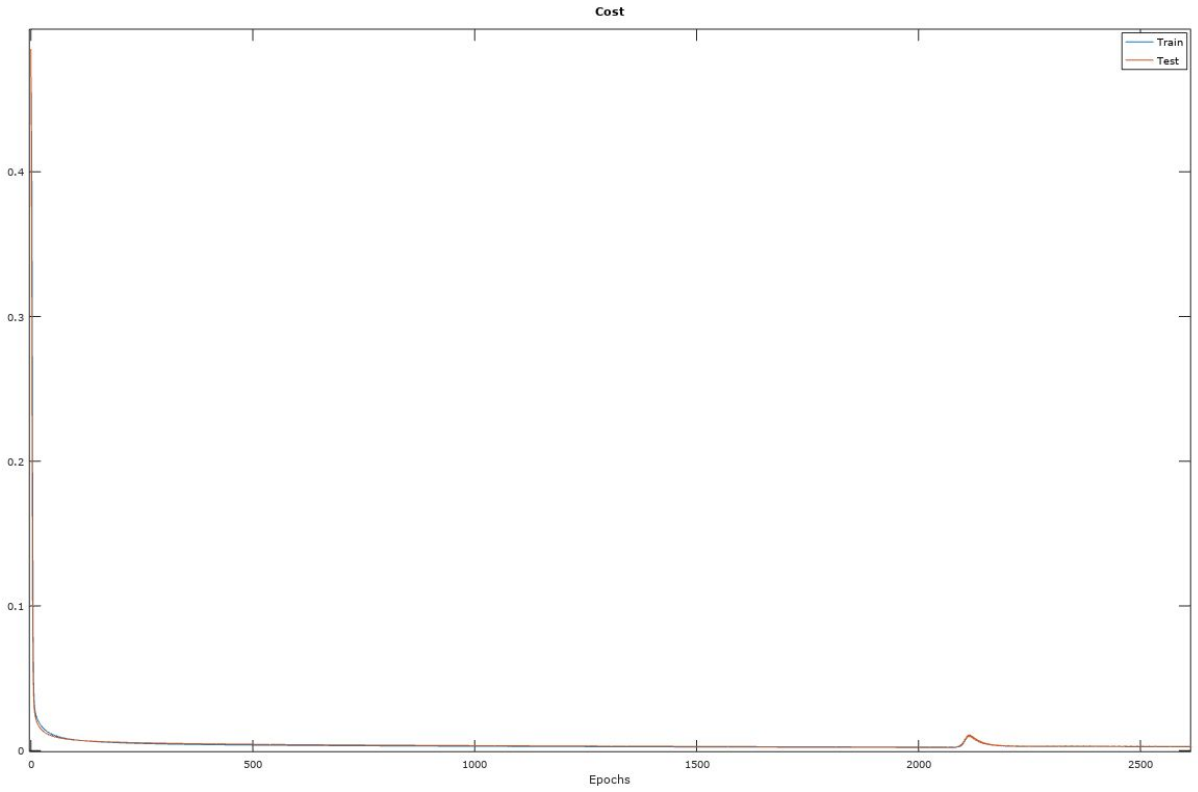


Figura 10 - Costo de las primeras 2600 épocas del caso 3 Vanilla, batch
No se presentan oscilaciones.

Otras inicializaciones

Otra variable analizada para este tipo de optimización fue la de cambiar la inicialización de los pesos de `uniform_one` a las otras descritas en la sección de *Implementación*. En términos generales, las otras inicializaciones comúnmente recomendadas no resultaron muy exitosas. Se realizó un análisis de las capas ocultas para tratar de determinar a que se atribuye su fracaso.

Tomando a la inicialización `normal` ($N(\mu = 0, \sigma = \sqrt{\overline{ni}})$), con `learning_rate` de 0.1, `batch_size` de 5, `seed` 8099 y backpropagation tradicional se obtuvieron los siguientes resultados:

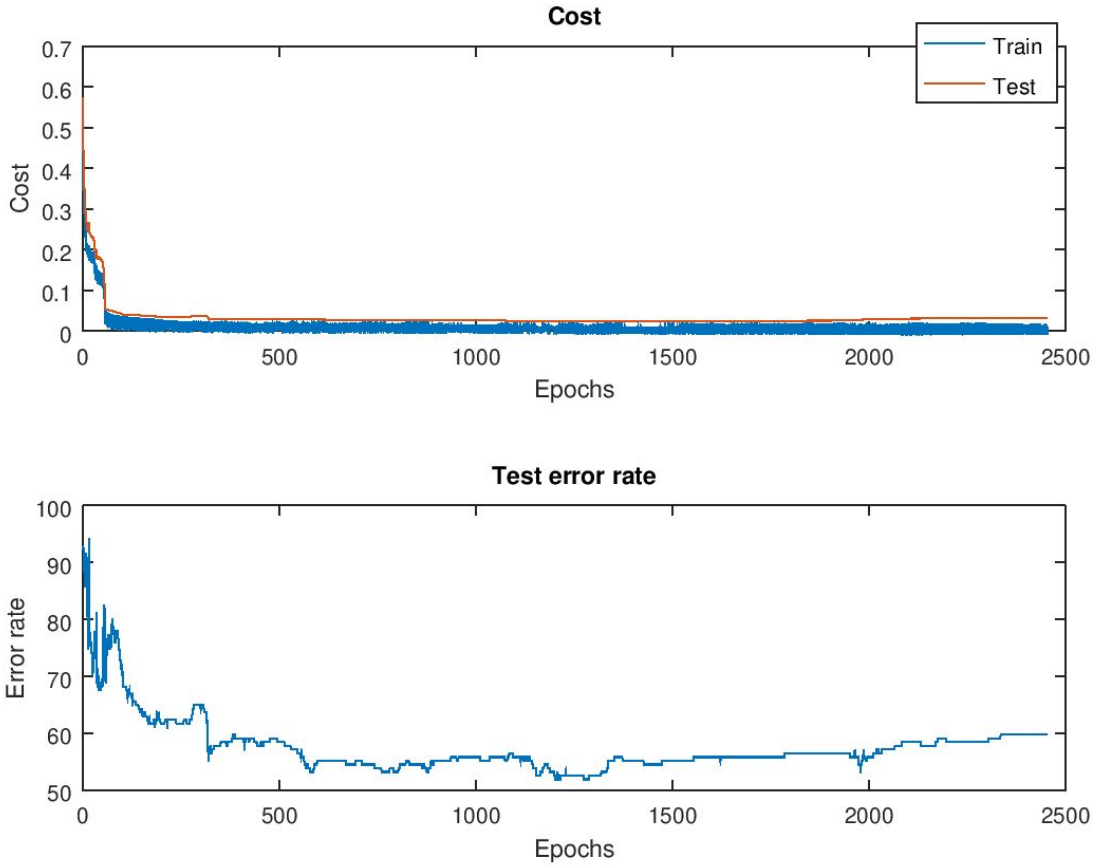


Figura 11 - Costo y error de la prueba con inicialización normal

Se destaca que desde aproximadamente la época número 800, el costo aún es alto, pero el entrenamiento deja de minimizarlo significativamente. Consecuentemente, la tasa de error de prueba no logra bajar del 55%, e incluso comienza a divergir.

Precisamente, luego de 2454 épocas, el entrenamiento se detiene por la condición de corte, terminando con un costo de 0.010529 y un error de 59.7%, sumamente más altos que en una prueba exactamente igual pero con inicialización de pesos `uniform_one`.

Se procede a analizar las capas ocultas realizando un histograma que cuenta la cantidad de neuronas de cada capa oculta según el promedio del valor de activación para todo el conjunto de entrenamiento.

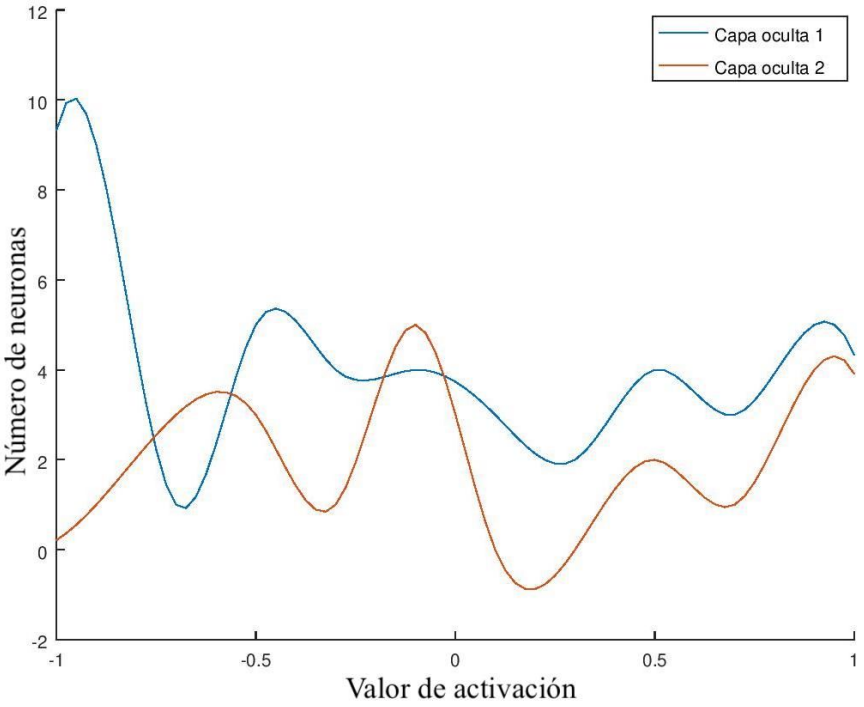


Figura 12 - cantidad de neuronas según su valor de activación promedio en todo el conjunto de entrenamiento, después de 800 épocas.

Se puede notar que en una etapa temprana del entrenamiento (a las 800 épocas el costo era de 0.0097 y el error de %53.24) hay una buena cantidad de neuronas, especialmente en la capa 1, con un valor de activación cercano a 1 o -1. Ésto de por sí es un indicador de que las capas comienzan a saturarse, lo cual también se puede corroborar notando que el error deja de bajar (e incluso por momentos aumenta) desde ésta época, y el costo disminuye muy lenta y erráticamente.

La saturación no es siempre un problema, pero en este caso lo que sucede es que se dá de manera temprana, en lugar de lo que es deseable, que es cuando se llega a un nivel de precisión satisfactorio en el entrenamiento.

Con los demás métodos de inicialización, salvo con `uniform_one`, se observaron resultados similares, especialmente cuando el número de unidades de la capa es más alto. Esto puede deberse a la forma en la que funcionan los métodos propuestos, ya que a más unidades los pesos están en promedio más cerca del 0, por lo que resultan muy pequeños y poco significativos durante el entrenamiento.

Momentum

Para las pruebas realizadas con el método momentum, los parámetros a configurar son: `alfa`, `learning_rate` y `batch_size`.

Se evalúa la red en los siguientes pasos para determinar la que mejor se acomode a lo pedido.

Paso 1 - Determinación de alfa

Se realizaron múltiples pruebas considerando variados `alfa`, los cuales se muestran en la siguiente tabla:

# Caso	`alfa`	`batch_size`	Error	Costo	Épocas	Tiempo (seg)
1	**0.9**	**5**	**<3%**	**$13.588x10^{-4}$**	**362**	**48.6138**
2	0.8	5	<3%	$13.626x10^{-4}$	384	49.7352
3	0.4	5	<3%	$13.227x10^{-4}$	521	67.4471

Tabla 5 - Resultados de pruebas con distintos `learning_rate`

Teniendo en cuenta lo recomendado teóricamente y los resultados de las pruebas se optó por elegir un `alfa` de 0.9. Si bien los resultados mostrados en la tabla 5 de los casos 1 y 2 son muy cercanos y competentes, dado que la diferencia en costo es casi despreciable, se busca aquel caso que necesite menos épocas para completarse y termine en el menor tiempo.

Se presentan las figuras 13 y 14 donde puede apreciarse que tanto el costo como el error disminuye de forma consistente.

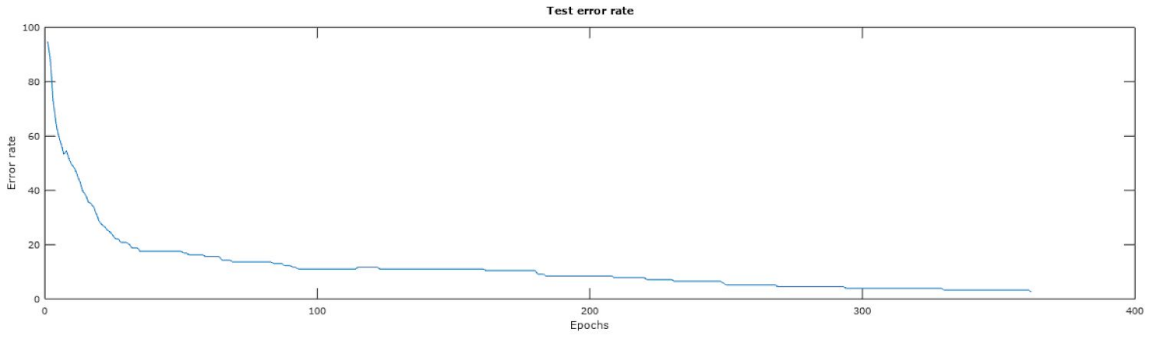


Figura 13 - Función de error en los puntos de prueba para el caso 1.

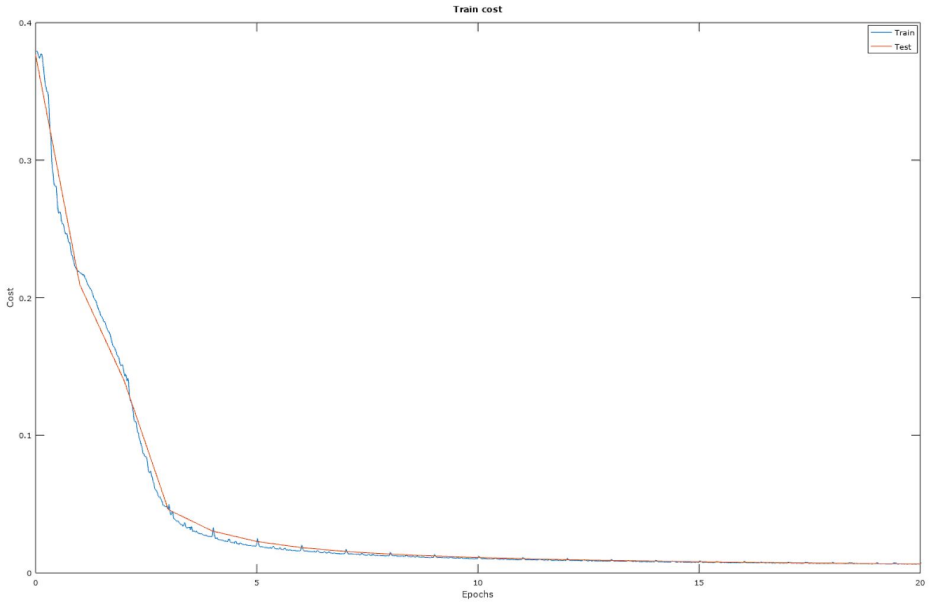


Figura 14 - Costo para el caso 1 en las primeras 20 épocas

Paso 2 - Determinación del learning rate

A partir de lo obtenido para el caso 1 de la tabla 5, se decidió fijar `alfa` en 0.9 y proceder a evaluar distintos valores de `learning_rate` haciéndolo variar desde 0.01 a 0.1. Se muestran a continuación los resultados de algunas pruebas realizadas con el fin de encontrar el mejor learning rate para resolver el problema.

# Caso	`learning rate`	`batch_size`	Error	Costo	Épocas	Tiempo (seg)
1	0.01	5	<3%	$13.588x10^{-4}$	521	48
2	**0.05**	**5**	**<3%**	**$13.342x10^{-4}$**	**93**	**12**
3	0.09	5	<3%	$8.8311x10^{-4}$	689	67
4	0.1	5	3.9%	$8.5387x10^{-4}$	1588	324

Tabla 6 - resultados variando `learning rate`

Luego de estas pruebas se obtuvo que la mejor configuración correspondía a un `learning_rate` de 0.05, finalizando en 93 épocas, 11.9578 segundos y con un costo de 13.342e-04. A continuación se muestran las figuras obtenidas:

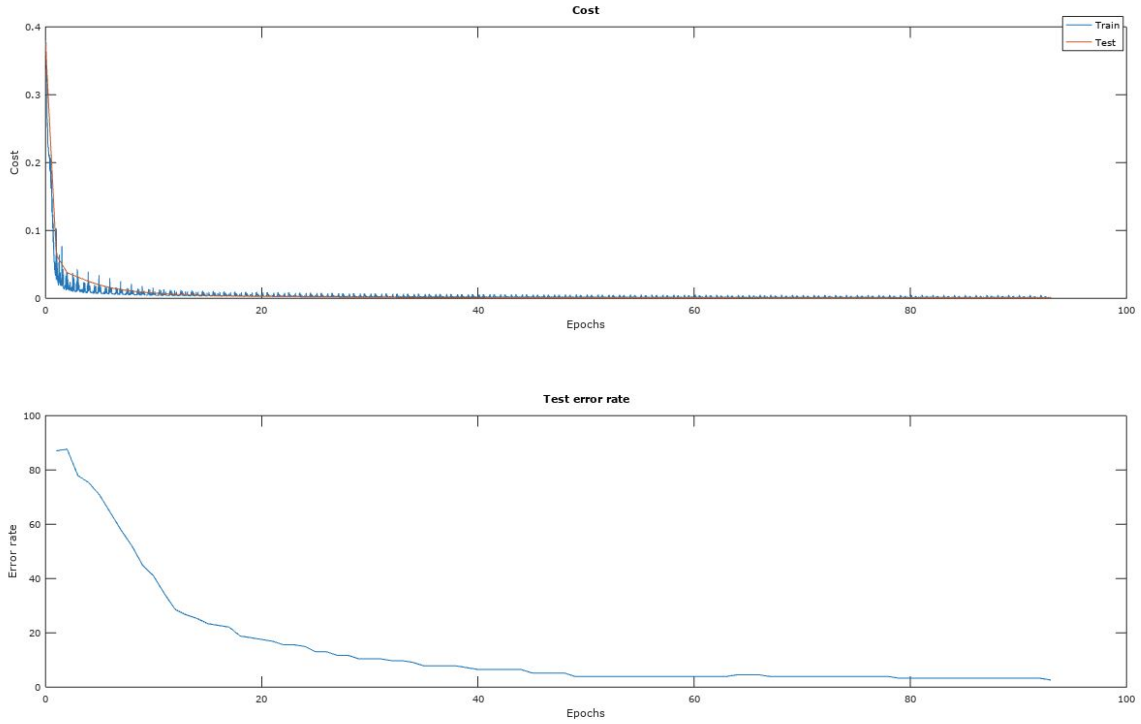


Figura 15 - Costo y error del caso 2 Momentum, paso 2.

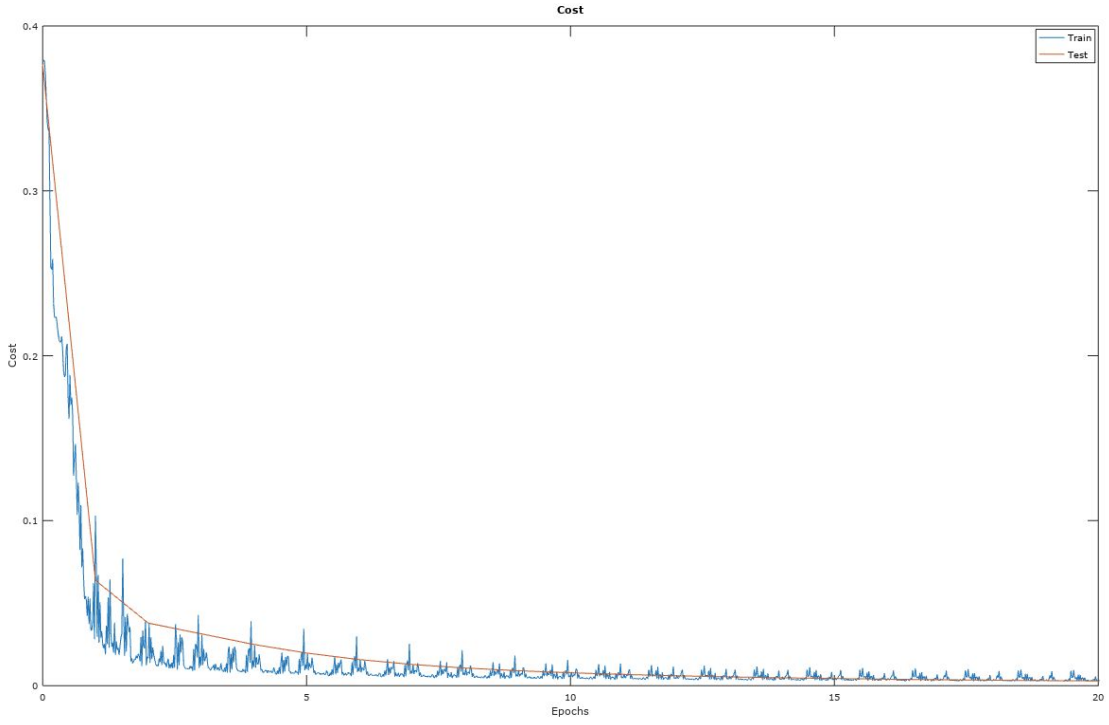


Figura 16 - Costo del caso 2 Momentum, paso 2

Paso 3 - Determinación del algoritmo

Se busca comparar la diferencia entre incremental, mini-batch y batch. Los resultados obtenidos para cada prueba se muestran en la siguiente tabla:

Caso	alfa	batch_size	Error	Costo	Épocas	Tiempo (seg)
Mini-batch	**0.9**	**5**	**<3%**	**$13.342x10^{-4}$**	**93**	**12**
Batch	0.9	batch	<3%	$10.086x10^{-4}$	10436	95
Incremental	0.9	1	<3%	$10.827x10^{-4}$	625	368

Tabla 6 - Comparación entre resultados obtenidos con distintos batch_size

Vemos en las siguientes figuras la comparación entre los costos de cada una para las primeras épocas:

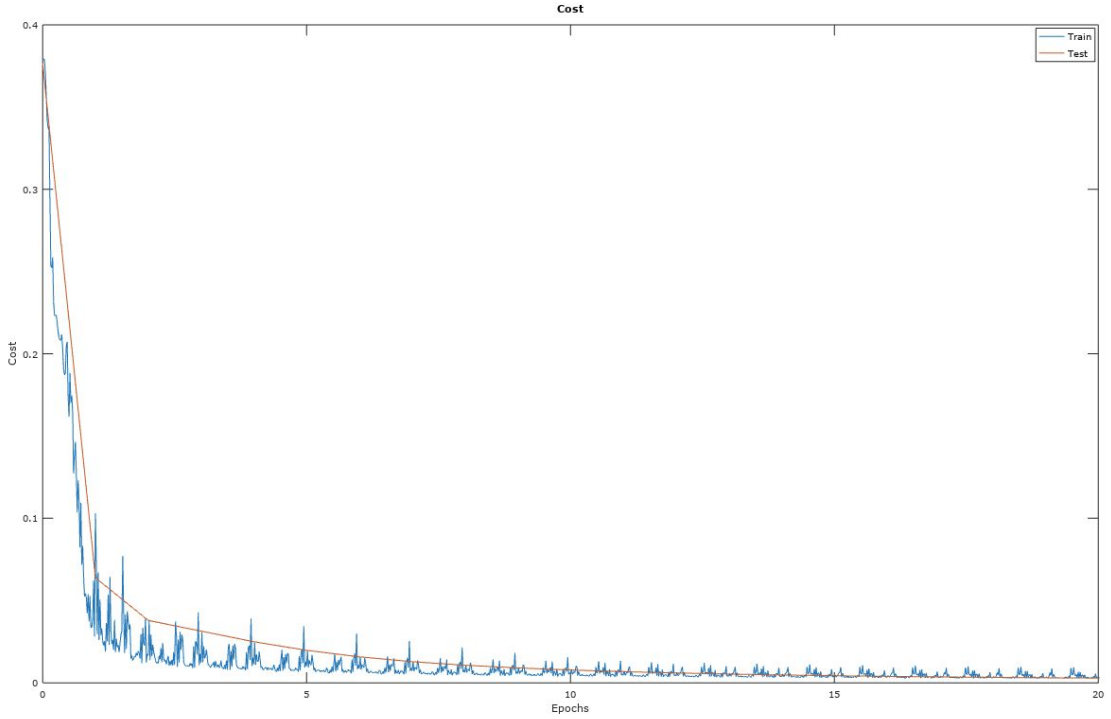


Figura 17 - Función de costo con mini-batch.
Oscilaciones pequeñas.

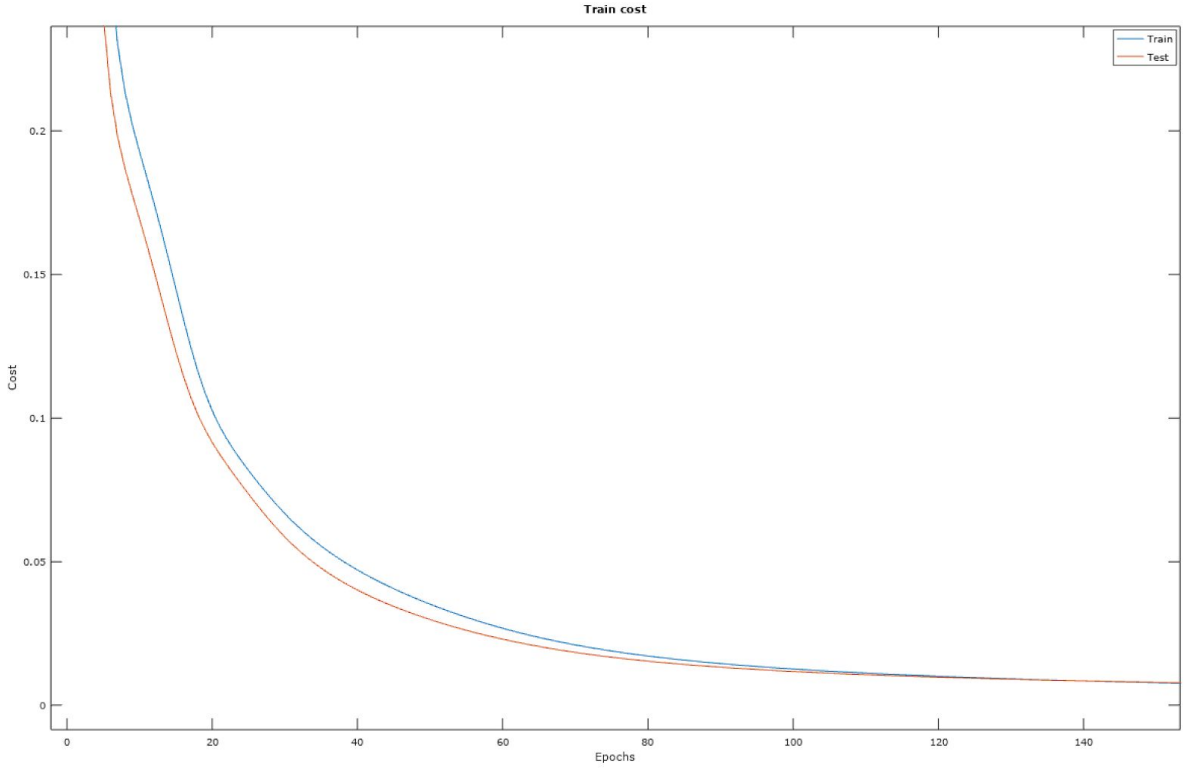


Figura 18 - Función de costo con batch.
No se presentan oscilaciones.

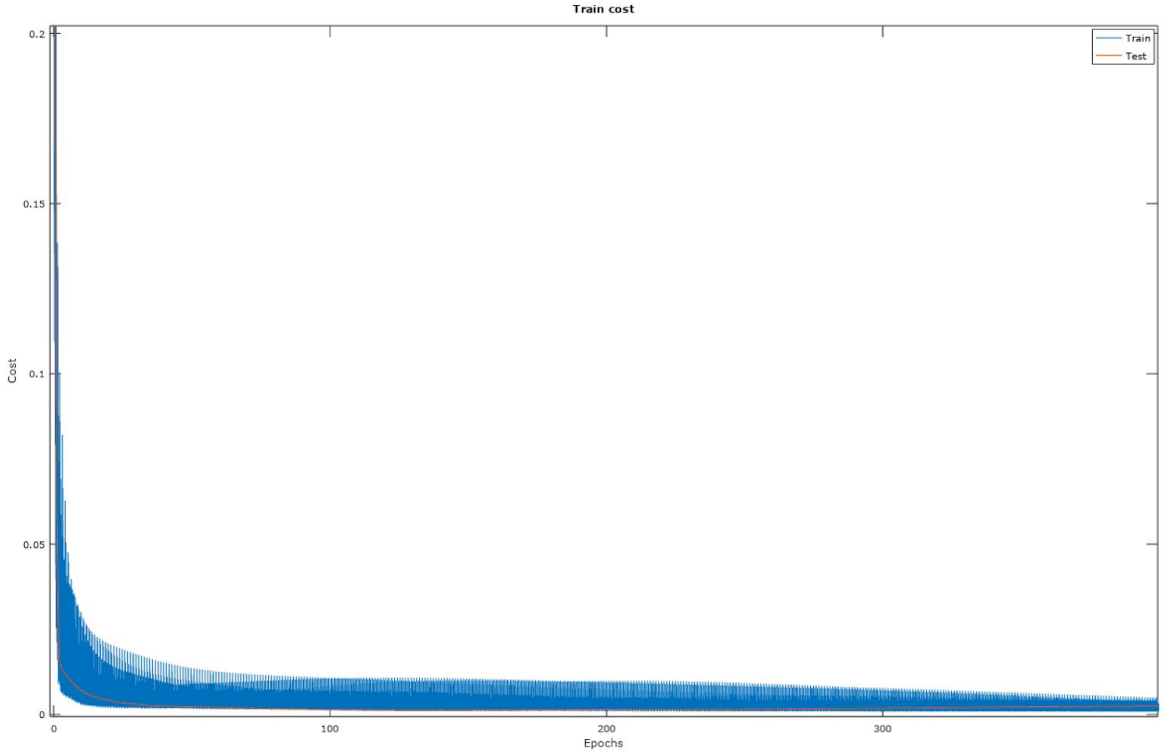


Figura 19 - Función de costo con incremental.
Notable cantidad de oscilaciones.

A partir de estos resultados obtenidos en la tabla 6 y de las figuras 17, 18 y 19 es sencillo ver la diferencia entre cada una. En la prueba incremental se pueden apreciar múltiples picos en comparación a los de batch y mini-batch, siendo batch la curva más suave de las tres.

Se puede ver que el costo de la prueba de mini-batch es notablemente mayor al de los dos restantes y además se logra completar en un tiempo notablemente menor. Estas razones no dejan duda alguna de que el caso de prueba con mini-batch es el óptimo, dado que se le da importancia a que la red tenga la tendencia a generalizar y no a memorizar.

Es interesante notar que el resultado preferible corresponda a un entrenamiento con configuración mini-batch y no batch ni incremental, pues se corresponde con los que se puede encontrar en la literatura sobre el tema. Una metodología batch suele ser demasiado conservadora como para llegar al objetivo mientras que una metodología incremental puede que sea demasiado errática. En cambio, se suele decir que mini-batch combina lo mejor de ambos mundos, avanzando pasos grandes y acertados hacia el objetivo, como lo respaldan los resultados.

Adaptive

Este algoritmo permite muchos valores parametrizables presentados anteriormente como: `cost_interval`, `inc_steps`, `lr_increase` y `lr_decrease_factor`. Por lo tanto se realizan pruebas con distintos valores buscando aquel caso que se muestre óptimo.
En la siguiente tabla se muestran algunos valores representativos de las pruebas realizadas que lograron cumplir la condición de que el error sea menor al 3%, todas con un `learning_rate` de 0.01.

Caso	batch_size	cost_interval	inc_steps	lr_increase	lr_decrease	Costo	Épocas	Tiempo (seg)
1	5	5	3	0.1	0.1	8.2649×10^{-4}	234	31.222
2	5	10	3	0.1	0.001	8.6368×10^{-4}	289	36.5901
3	5	10	4	0.01	0.001	8.5815×10^{-4}	249	31.6767
4	5	5	4	0.005	0.01	7.0883×10^{-4}	1616	217.057
5	1	10	4	0.01	0.001	8.0584×10^{-4}	173	100.785
6	1	5	5	0.01	0.01	9.1356×10^{-4}	351	211.484
7	**1**	**5**	**10**	**0.001**	**0.0001**	**12.726×10^{-4}**	**178**	**108.675**

Tabla 7 - Resultado de pruebas con el algoritmo adaptive

A partir de estos resultados, son destacados tanto el caso 3 como el caso 7, teniendo el caso 3 un menor tiempo, pero el caso 7 un costo notablemente más alto y una cantidad de épocas menor. Es por estas razones que se determina como la mejor prueba la del caso 7.

Se muestra con más detalle el resultado del caso 7:

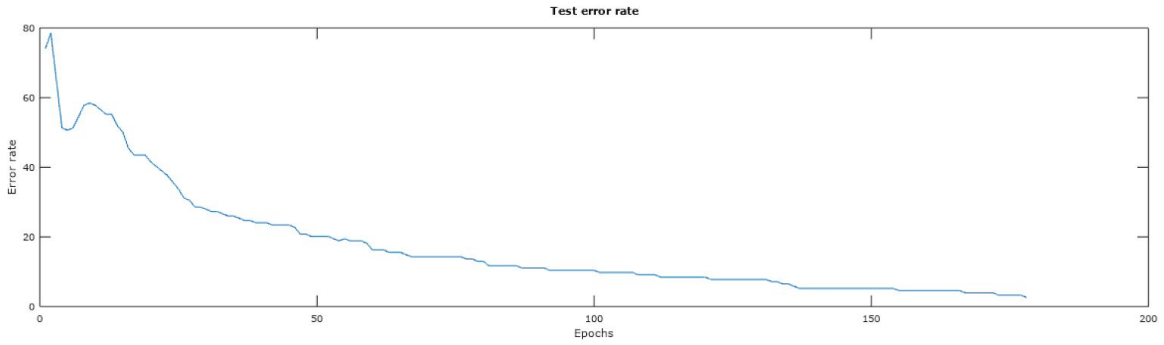


Figura 20 - Error del caso 7 Adaptive

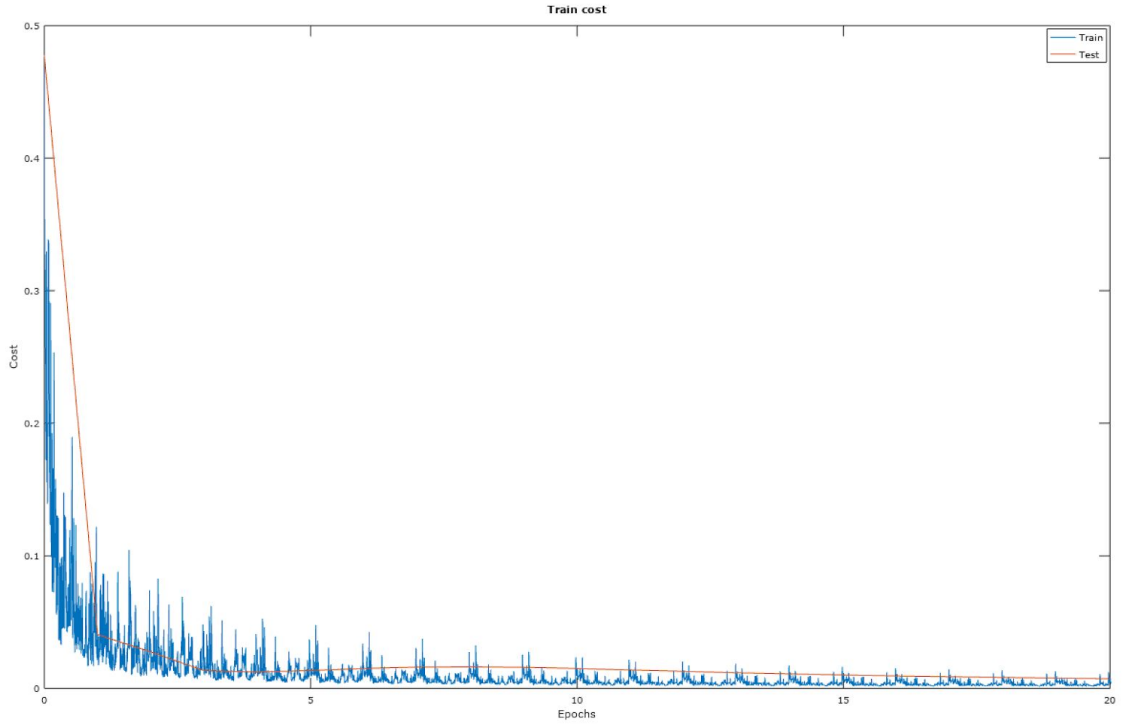


Figura 21 - Función de costo para las primeras 20 épocas del caso 7

Aún habiendo obtenido este resultado, podemos notar que no supera a resultados obtenidos con otras funciones evaluadas y probadas anteriormente, como por ejemplo con `momentum`, puesto que tarda más tiempo y tiene menor capacidad de generalización (Ver caso mini-batch de la tabla 6).

Resultados

Luego de elegir la mejor configuración y de comparar los distintos métodos de entrenamiento, tomando en cuenta las prioridades que se plantearon en las consideraciones iniciales, se determina que la mejor red para la resolución del problema es la siguiente:

Configuración

- 2 capas ocultas: 40 y 20 neuronas respectivamente.
- Función de activación: tangente hiperbólica.

Entrenamiento

- Backpropagation con `momentum`.
- `learning_rate: 0.05`
- `alfa: 0.9`
- `batch_size: 5`

Esta configuración de entrenamiento logra un alto grado de generalización en un tiempo reducido en contraposición a los resultados obtenidos para los mejores casos `Vanilla` y `Adaptive`, que son cualitativamente inferiores en ambos aspectos.

De esta manera fue posible lograr un error de testeo menor al 3% (usando los parámetros de `tolerance` y de `test_rate` mencionados previamente) con un costo de entrenamiento de 13.342×10^{-4} . Los resultados pueden variar según la semilla que se utilice, pero en términos generales, ésta fue la configuración que mejor cumplía con el objetivo propuesto de forma consistente.

Se muestra por último en la figura 22 los puntos del terreno conocidos versus los puntos generalizados por la red.

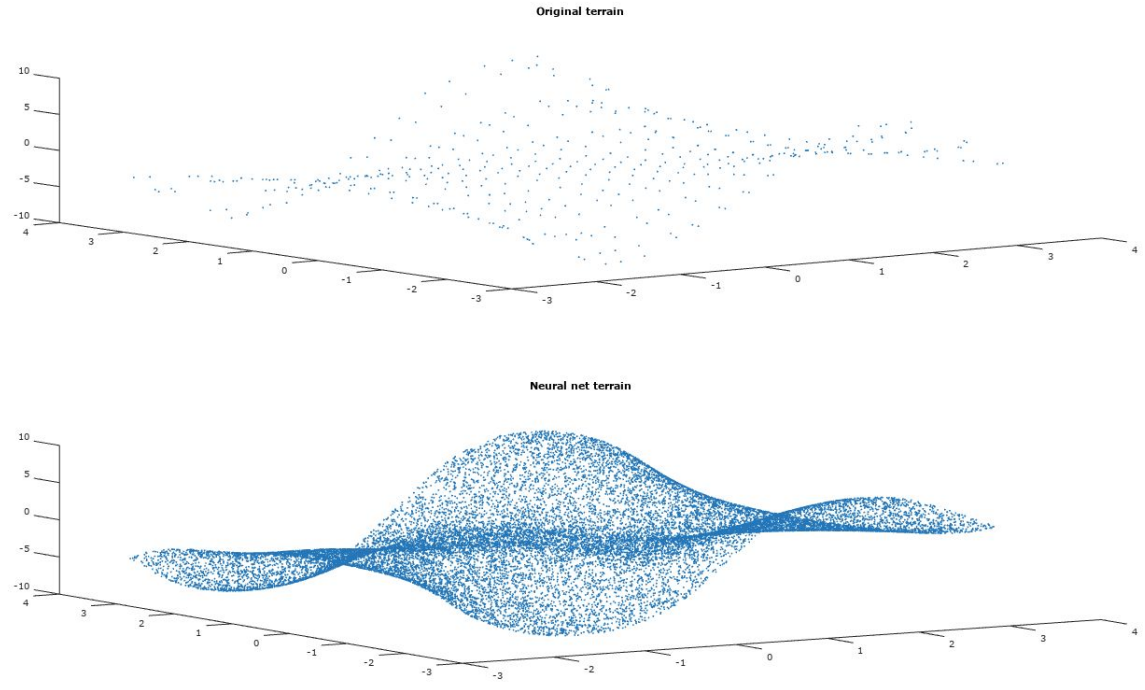


Figura 22 - Representación de los puntos conocidos vs. los puntos representados por la red

Conclusiones

A lo largo del informe se evidenció la dificultad que representa encontrar una red neuronal *óptima* para resolver un problema dado, incluso uno relativamente simple. La gran cantidad de parámetros y combinaciones de los mismos hace fundamental el criterio del diseñador a la hora de elegir entre las innumerables configuraciones que se pueden probar. Como solución a ésto, resultó útil y efectiva la idea de separar el análisis en dos etapas, desacoplando la selección de la configuración de la red de la elección de la forma de entrenamiento. De ésta manera, se redujeron drásticamente las combinaciones de parámetros posibles, buscando no afectar en gran medida la calidad final de la red elegida.

El análisis separado, sin embargo, también trajo un inconveniente. En la etapa del análisis de entrenamiento fue claro que se debió haber evaluado con un learning rate mayor a 0.01 durante la etapa de selección. Un learning rate superior pudo haber permitido que configuraciones de red con incluso menos neuronas hayan podido resolver el sistema bajo nuestros criterios, ya que como se pudo observar en los resultados de selección, en general la tasa de error disminuía de forma consistente pero con cambios tan pocos significativos que incluso con los generosos valores de `min_cost_change` y `max_tries` se daban por terminadas las evaluaciones como fracaso.

Queda pendiente de un análisis más profundo la influencia de las distintas inicializaciones de pesos. Si bien para éste problema en particular no dieron los resultados esperados, son ampliamente usadas en la industria y la academia. Posiblemente se deba analizar un problema de otras características para aprovechar los beneficios de dichas técnicas.

Queda pendiente también analizar de forma más profunda el efecto de diferentes tipos de normalización como también la opción de normalizar. Para esto puede ser apropiado utilizar en la capa de salida una función lineal como función de activación.

Cabe enfatizar también que el criterio de red neuronal óptima es altamente sensible al problema y contexto en cuestión, donde sin temor a equivocarse podrían priorizarse otras características como el tiempo de entrenamiento o mayor más grado de precisión para con los resultados de la red. Asimismo no resultan sencillos tampoco los criterios para determinar que una red se encuentra en un mínimo local para así poder dar por terminada una evaluación de entrenamiento, ya que es posible que simplemente se encuentre en un *plateau*.