# Sistemas de Inteligencia Artificial

---

Trabajo Práctico Especial N°2

Redes Neuronales


Integrantes:

- Alejandro Bezdjian, 52108
- Horacio Miguel Gómez, 50825
- Juan Pablo Orsay, 49373
- Sebastián Maio, 50386

# Tabla de contenidos

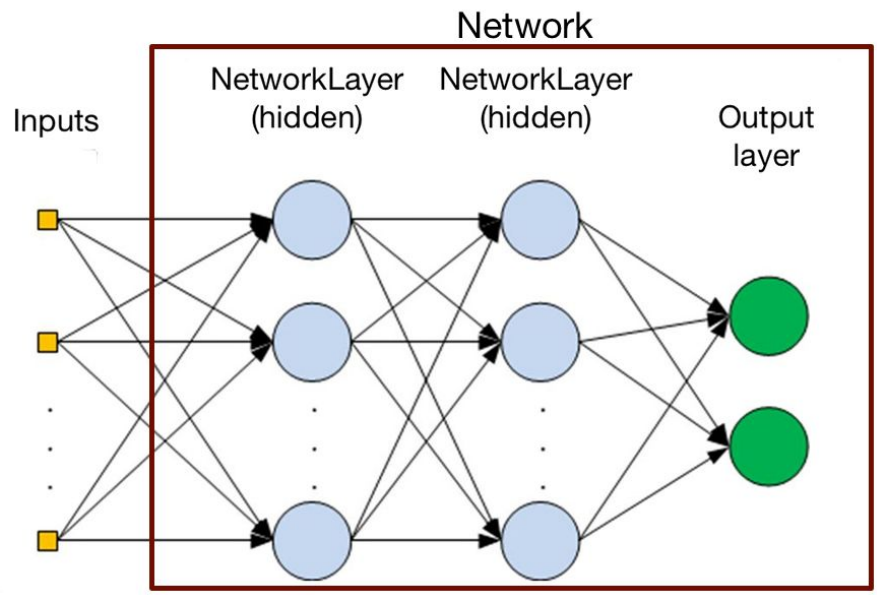
# Introducción

Una empresa de videojuegos precisa un desarrollo que pueda simular en su plataforma terrenos de diferentes partes del mundo a partir de mediciones de altura, latitud y longitud.
Para ello, se implementará una red neuronal multicapa con aprendizaje supervisado, la cual aproxime a la función de altura.

# Implementación

## Modelado

La implementación de la red se realizó en Python. Para la misma se modelaron las siguientes clases: Network, NetworkLayer y Neuron.




### Network

Network es la red, la cual contiene múltiples instancias de NetworkLayer, además es la encargada de la inicialización y entrenamiento de la red.

Internamente los datos entran a la primera capa (NetworkLayer) de la red y la salida es considerada la salida de la última capa.

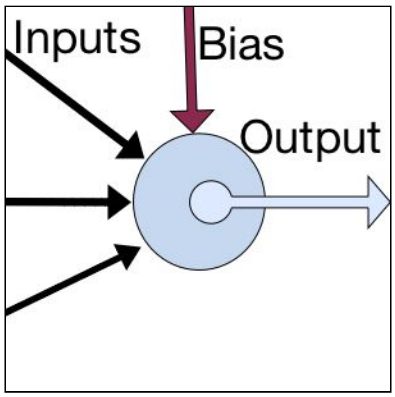
La forma recomendada de crearla es mediante archivos de configuración que son descritos en el anexo Config.

### NetworkLayer

Representa una capa de la red neuronal y contiene todas las neuronas pertenecientes a la red. A su vez, cada capa de la red, por diseño, contiene la función de activación de cada neurona, por lo que es posible especificar una función de activación distinta para cada capa.

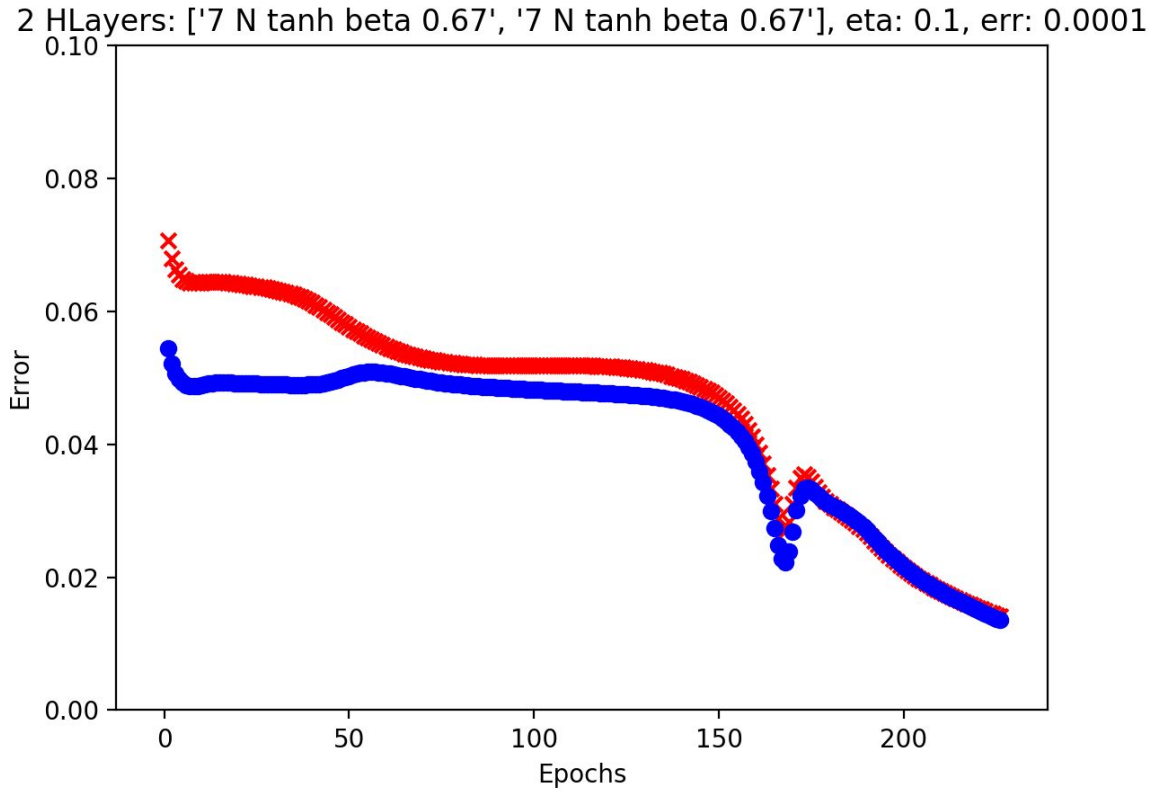
### Neuron

Cada neurona contiene un array con los pesos que corresponden a cada uno de los inputs que recibe junto con un bias (sesgo). Los mismos se utilizan para calcular la salida de la neurona con la función de activación que se configuró para esa capa.




## Aprendizaje

Para el aprendizaje de la red se implementó el algoritmo Backpropagation, con algunas variantes para intentar mejorar el aprendizaje (tanto en tiempo como en precisión). Para cada época utilizamos el entrenamiento incremental debido a que nos ayuda a evitar caer en mínimos locales para el tipo de problema planteado. Durante la etapa de aprendizaje donde se enseña a la red y calcula el error de entrenamiento, no se utilizaron todos los datos conocidos, sino solo parte de ellos. Con el resto de los datos, se testeó la capacidad de generalización de la red, introduciendo en la misma los datos de entrada, y una vez obtenidos los estimados, se comparaban con el valor real, obteniendo así el error de testeo. Para verificar que la red aprendía, graficamos los errores de entrenamiento y testeo.




En este gráfico observamos en rojo el error de entrenamiento y en azul el error de testeo. En el mismo se puede observar que la red había caído en un mínimo local (aproximadamente en la época 160) y que pudo salir del mismo. También podemos ver como luego de salir de ese mínimo local hubo una fluctuación en los errores, donde el error de testeo pasaba a ser mayor que el de entrenamiento y luego se vuelve a hacer menor. Este patrón ocurre muchas veces durante el entrenamiento de distintas redes, por lo que hay que ser muy cuidadoso con la condición de corte del entrenamiento. Usualmente se desea que el entrenamiento frene cuando el error de testeo supera al de entrenamiento para no sobre entrenar la red, sin embargo eso no se hubiese cumplido para este caso.

## Selección de datos de aprendizaje

Para lograr que la red aprenda correctamente, es importante la elección del dataset con que se entrena. Si los datos de entrenamiento no son representativos, la red no será capaz de aprender y generalizar.

A continuación veremos algunos resultados obtenidos con una red de 2 capas ocultas y 4 neuronas en cada capa, utilizando la función tangencial con $\beta = \frac{2}{3}$. Para todas las corridas se utilizaron los mismos pesos iniciales. Se tomó el mínimo error obtenido como dato de interés (sin importar si era o no un mínimo local).

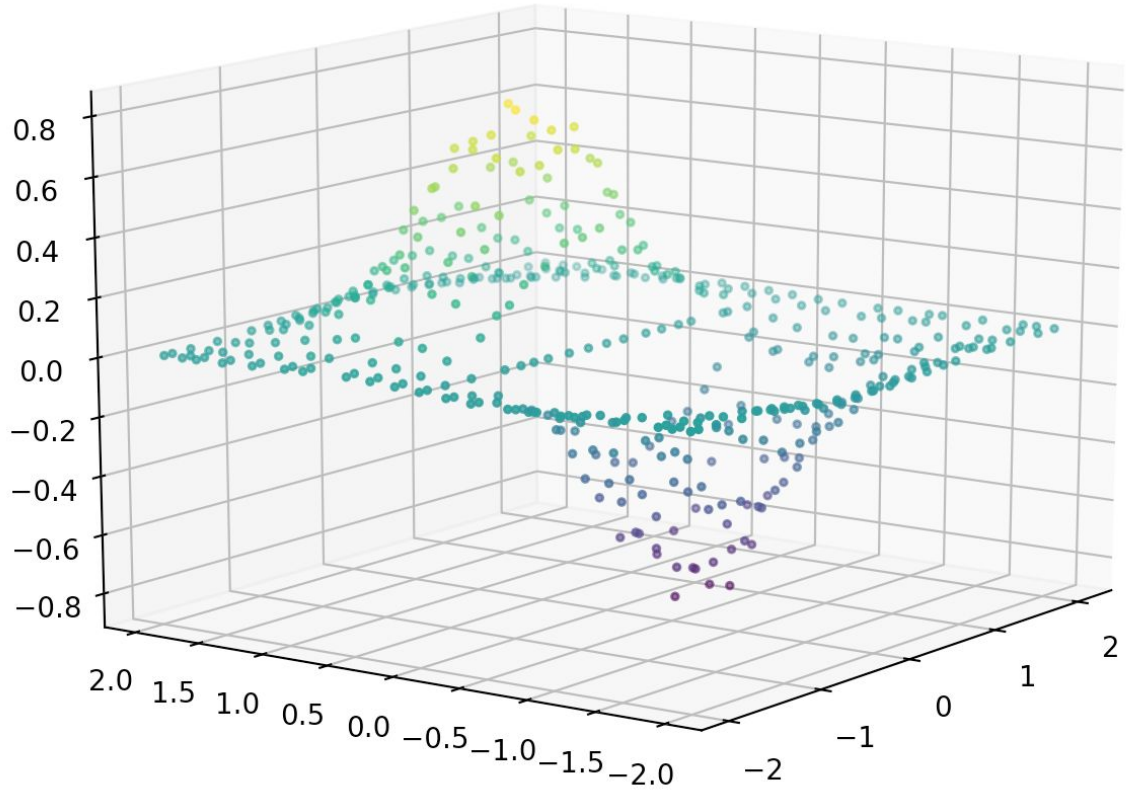
| $\eta$ | **Data Datos de entrenamiento** | **Datos de testeo** | **Épocas** | **Mínimo error** |
|---|---|---|---|---|
| 0.5 | Primer mitad, como venían | Segunda mitad, como venían | 2000 | 0.0347848653544 |
| 0.2 | Primer mitad, como venían | Segunda mitad, como venían | 2000 | 0.00420552751584 |
| 0.05 | Primer mitad, como venían | Segunda mitad, como venían | 2000 | 0.0238143816887 |
| 0.5 | Ordenados por z, tomando pares | Ordenados por z, tomando impares | 2000 | 0.188835162166 |
| 0.2 | Ordenados por z, tomando pares | Ordenados por z, tomando impares | 2000 | 0.188835162166 |
| 0.05 | Ordenados por z, tomando pares | Ordenados por z, tomando impares | 2000 | 0.0475945254823 |
| 0.5 | Ordenados por z, tomando impares | Ordenados por z, tomando pares | 2000 | 0.200275795184 |
| 0.2 | Ordenados por z, tomando impares | Ordenados por z, tomando pares | 2000 | 0.184881309443 |
| 0.05 | Ordenados por z, tomando impares | Ordenados por z, tomando pares | 2000 | 0.0439698759341 |
| 0.5 | Primeros 20 valores, como venían | Últimos 421 valores, como venían | 2000 | 0.0366820210271 |
| 0.2 | Primeros 20 valores, como venían | Últimos 421 valores, como venían | 2000 | 0.0350868838187 |
| 0.05 | Primeros 20 valores, como venían | Últimos 421 valores, como venían | 2000 | 0.0333646748939 |
| 0.5 | Últimos 20 valores, como venían | Primeros 421 valores, como venían | 2000 | 0.0386613375083 |
| 0.2 | Últimos 20 valores, como venían | Primeros 421 valores, como venían | 2000 | 0.0320534847978 |
| 0.05 | Últimos 20 valores, como venían | Primeros 421 valores, como venían | 2000 | 0.027874577911 |

Un resultado interesante es que ordenando los datos y tomando alternadamente sus valores no da un mejor resultado que tomando la misma cantidad de datos pero de la forma arbitraria dada. Esto es un poco anti intuitivo debido a que se esperaría que al tomar los datos ordenados y de forma alternada, la red pueda aprender tanto los outliers como los puntos regulares. Esto nos da un indicio que es conveniente que haya una cierta aleatoriedad en la selección de los datos.

# Selección de arquitectura

La arquitectura es un elemento clave debido a que varios aspectos de la red quedarán definidos gracias a ella, por ejemplo si la red será o no capaz de aprender el problema, el tiempo que tardará la red en predecir un valor para una nueva entrada y también el tiempo de entrenamiento aumentará mientras más capas y neuronas tenga la red.

Primero veamos la forma que tienen los datos originales:

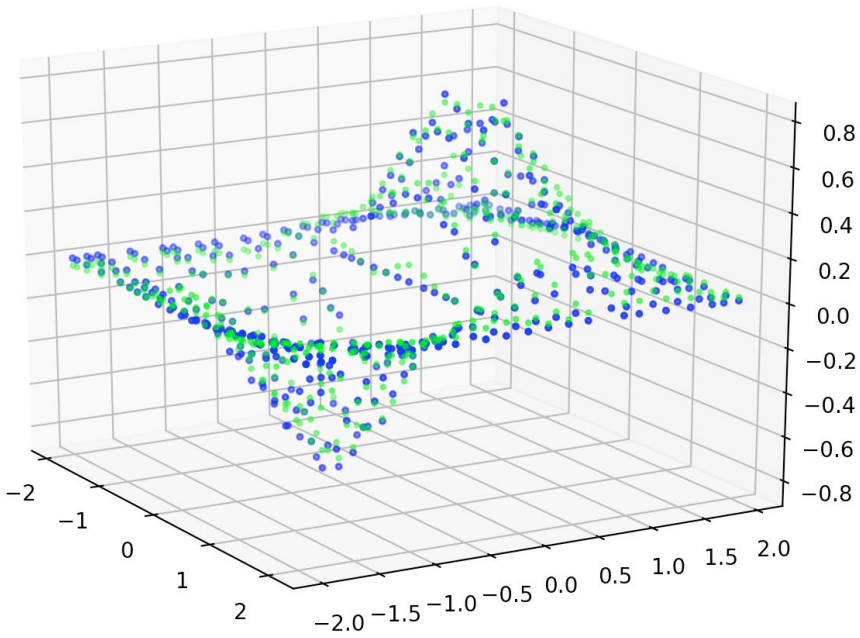



Se puede observar que se forman dos protuberancias en el terreno, una ascendente y otra descendente (como sabemos que es un terreno de un videojuego estas podrían ser, quizás, una montaña y el fondo de un lago).

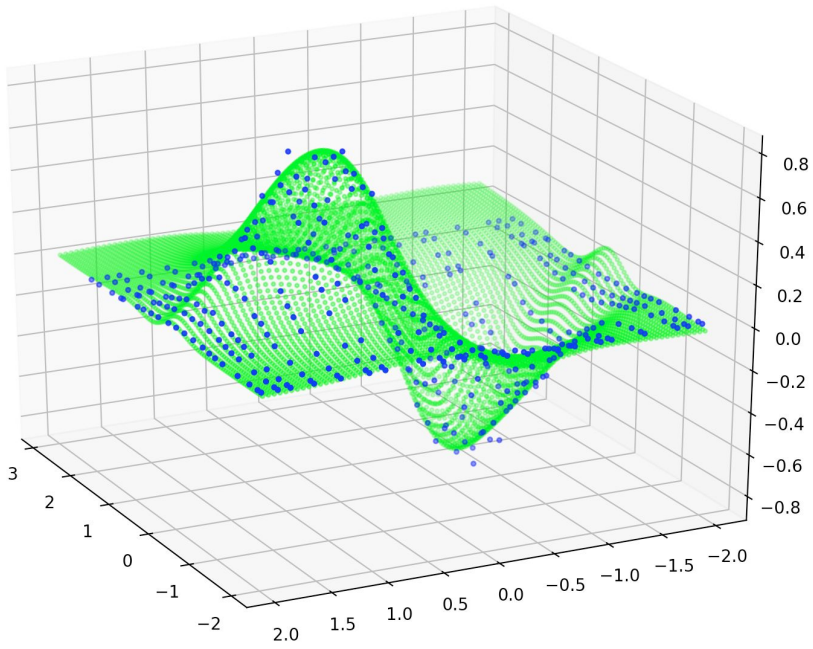
Teniendo en cuenta la figura deseada, deseamos que la red genere una superficie similar. En el anexo se mostrarán ejemplos de gráficas obtenidas con distintas arquitecturas.

Debido a que el aprendizaje tiene una fuerte dependencia de los valores de los pesos iniciales, se ejecutaron varias pruebas con pesos inicializados aleatoriamente y con variaciones del $\eta$. La información que nos interesa obtener de cada prueba y arquitectura es: tiempo de entrenamiento, tiempo que tarda en predecir, error obtenido para los inputs de prueba. Para todas las arquitecturas utilizamos una función lineal en la capa de salida.

Para el problema en cuestión, se decidió que la mejor arquitectura debe tener al menos 2 capas y 4 neuronas en cada capa. Esto nos genera una red capaz de generar un terreno muy similar al que se requiere (el realismo es importante para un videojuego). La predicción de una salida para un input cualquiera se resuelve en el orden de 10^-5 segundos por lo que, si bien no es la arquitectura más rápida, es un tiempo totalmente aceptable para un videojuego.




En azul vemos los puntos originales, en verde los calculados por la red.




En azul vemos los puntos originales y en verde vemos la superficie calculada por la red.

# Mejoras sobre Backpropagation

## Momentum

Se agregó un término que pesa el descenso promedio y busca ayudar a no quedar atrapados en mínimos locales. La manera en la cual implementamos momentum fue:

$$\Delta W_{ij}(t+1) = \eta \delta_i^{\,m} V_j^{\,m-1} + \alpha \Delta W_{ij}(t)$$

En donde:

$\delta_i^{\,m}$: Error de la neurona i en la capa m

$V_j^{\,m-1}$: Salida de la neurona j en la capa m-1

$\alpha$: Coeficiente de momentum. $0 < \alpha < 1$

$\Delta W_{ij}(t)$: Delta peso de la conexión entre la neurona i en la capa "m" y la j en la capa "m-1".

Esto genera que la actualización de pesos no dependa únicamente del gradiente actual, sino también que del valor en el paso anterior, generando así una especie de inercia en la actualización del costo. De ahí el nombre momentum.

A su vez, hicimos que el momentum varíe cada cierta cantidad de épocas para buscar evitar quedarnos en un mínimo local. Esto se debe principalmente a que cuando la red no está entrenada, no es tan conveniente ni necesario usar momentum, pero conforme se achica el error y consolidan los datos en la red, es necesario incrementar el momentum para que sea efectivo en ayudarnos a salir de los mínimos. En general comenzamos con $\alpha = 0.5$ y terminamos en 0.9.

## Parámetros Adaptativos

Otra mejora sobre *backpropagation* es modificar $\eta$ según:

*Si una actualización de pesos particular no decrementa la función de costo, $\eta$ debería reducirse.*

*Si varios pasos en el aprendizaje han reducido la función de costo, entonces habriá que incrementar $\eta$ para no ser tan conservadores.*

*Se incrementa en una constante y se decrementa geométricamente, para permitir un rápido decaimiento.*

$$\Delta\eta = \begin{cases} +a & si \;\; \Delta E < 0 \;\; consistentemente \\ -b\eta & si \;\; \Delta E > 0 \\ 0 & en\ otro\ caso \end{cases}$$

Queda entonces el problema de decidir a qué llamamos "consistentemente" y que valores de **a** y **b** tomar. Para nosotros, consistentemente quiere decir que una red hace varias épocas tiene una tendencia marcada de cambio de error. Esto presenta un problema que notamos empíricamente: si tomamos los últimos **k** errores de aprendizaje podría suceder que haya uno de esos valores que no sigan la tendencia. Por ejemplo, supongamos que el error después de una época es de 0.4 y que tenemos los siguientes errores previos: [0.9, 0.85, 0.7, 0.6, 0.5, 0.4, 0.5, 0.3], vemos que existe una tendencia de disminución, sin embargo si solo tomamos el último valor previo (0.3) nos parecería ser que el error está aumentando y tomaríamos decisiones erróneas. Este problema se torna aún más complicado de decidir si miramos más pasos hacia atrás. Es por eso que se decidió aplicar una regresión lineal para los últimos k errores de testeo y calcular la pendiente de la función lineal obtenida y en base a ésta tomar una decisión. Para la regresión lineal se utilizó la librería scikit de python.

Una vez conseguido el valor de la pendiente se debía tomar una decisión sobre qué valor nos haría tomar la decisión de cambiar el $\eta$, ya que si lo hacemos muy sensible el $\eta$ tiende a disminuir de forma violenta. De forma empírica obtuvimos buenos resultados aumentando el $\eta$ cuando la pendiente era mayor a -10^-7 y disminuir el $\eta$ cuando la pendiente era mayor a 10^-7. Las constantes utilizadas además fueron a=0.01, b=0.1.

## Adaptative Annealing

Para evitar dar saltos alrededor del mínimo, y lograr converger mejor, otra mejora sobre backpropagation que se implementó fue adaptive annealing. La idea es gradualmente ir reduciendo la tasa de aprendizaje ($\eta$).
Para ello se utiliza la siguiente fórmula
$\eta(t) = \eta(0)/(1 + t/T)$
La cual llevada a codigo, resultó ser:

```
def _adapt_eta_annealing(self, previous_errors):
    if len(previous_errors) >= self._adaptive_annealing_k:
        self.eta = self._original_eta / (1 + ((len(previous_errors) - 1) / self._adaptive_annealing_k))
```

Este método es llamado en cada paso del entrenamiento, achicando el $\eta$ conforme avanza el algoritmo (época a época).

# Conclusiones

Se pudo implementar una red neuronal multicapa capaz de predecir alturas de un terreno dadas las coordenadas x e y. Para su entrenamiento se utilizó el algoritmo de backpropagation y sus variantes.

Entrenar a la red con una única variante de backpropagation puede ser contraproducente. Hay ciertos momentos que queda atrapada en un mínimo local y es necesario utilizar momentum, parámetros adaptativos o manualmente aumentar el $\eta$ en algunas épocas hasta que logre salir del mismo.

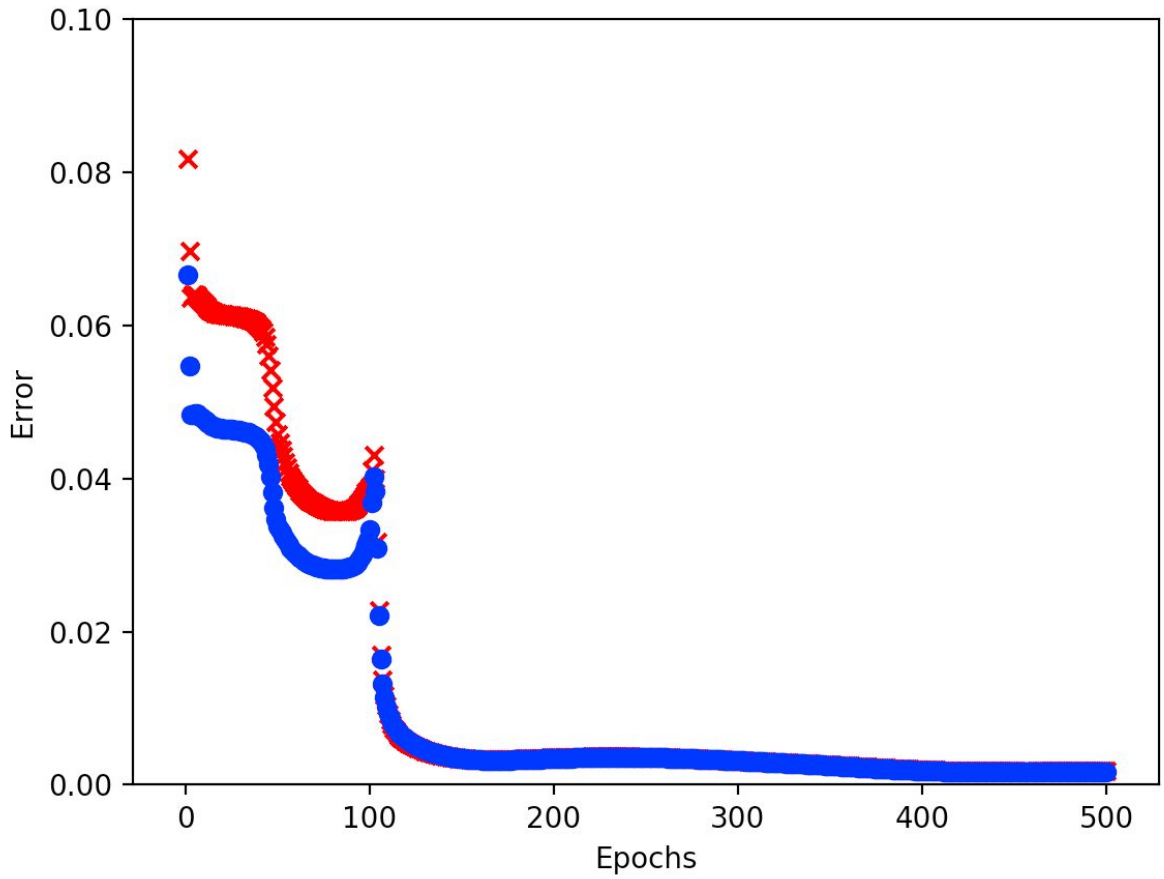
Además queda claro que la selección de los valores de aprendizaje es importante y que no con cualquier conjunto de valores se logra que la red llegue a un nivel de generalización satisfactorio.

Se vió que la aleatoriedad en los datos de la entrada ayuda a la red a aprender, aunque eso parezca poco intuitivo.

# Anexo

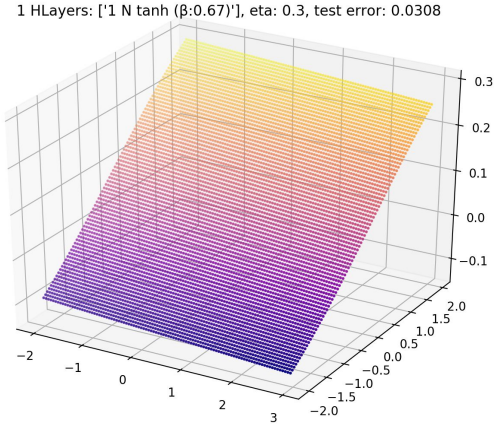
## Resultados

En los gráficos siguientes se puede visualizar, como durante el aprendizaje, si bien se alcanzó mínimos locales, se logró salir de los mismos y continuar con el aprendizaje. (Azul - error de test || rojo - error de training)

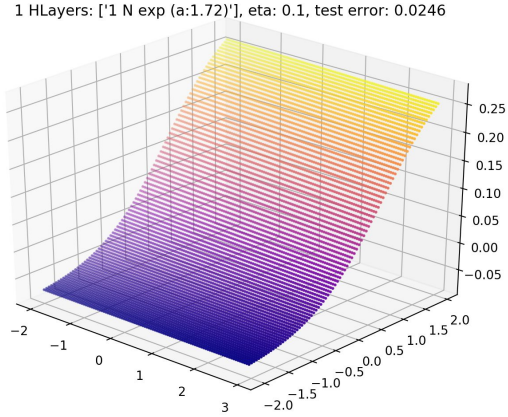


# Resultados obtenidos probando arquitecturas

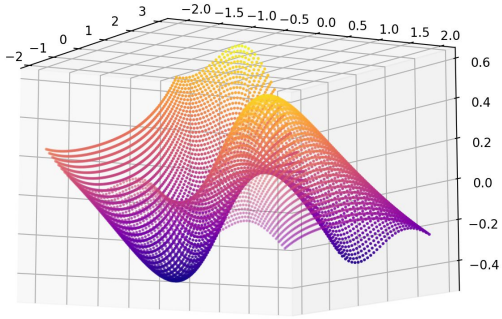
1 HLayers: ['1 N tanh (β:0.67)'], eta: 0.3, test error: 0.0308



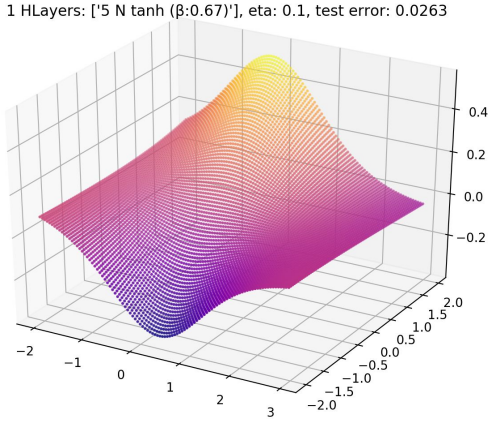
1 HLayers: ['1 N exp (a:1.72)'], eta: 0.1, test error: 0.0246



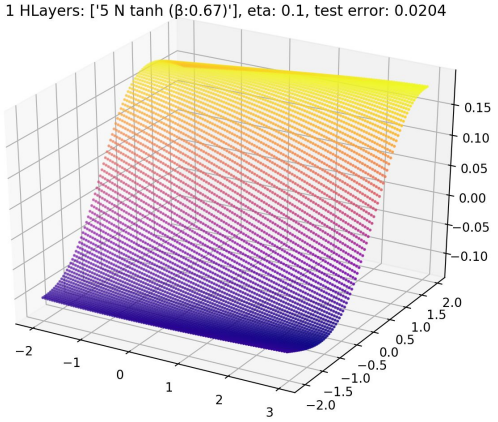
1 HLayers: ['5 N tanh (β:0.67)'], eta: 0.1, test error: 0.0065



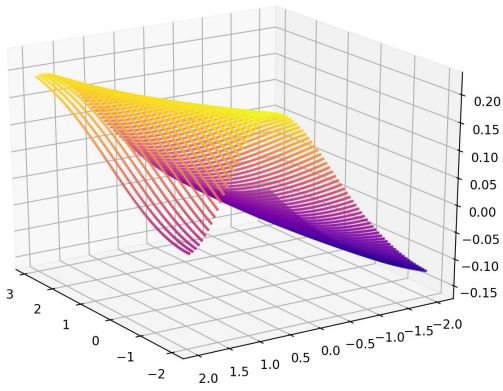
1 HLayers: ['5 N tanh (β:0.67)'], eta: 0.1, test error: 0.0263



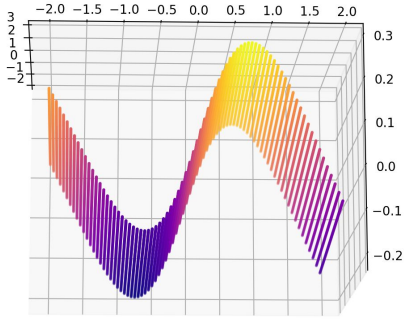
1 HLayers: ['5 N tanh (β:0.67)'], eta: 0.1, test error: 0.0204



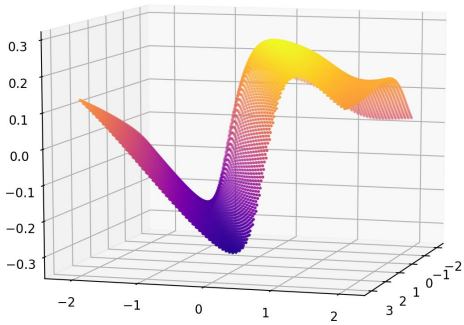
1 HLayers: ['5 N exp (a:1.72)'], eta: 0.1, test error: 0.0205



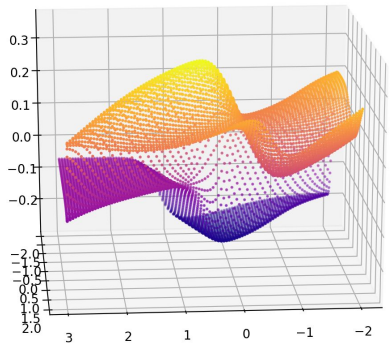
1 HLayers: ['6 N tanh (β:0.67)'], eta: 0.1, test error: 0.0161



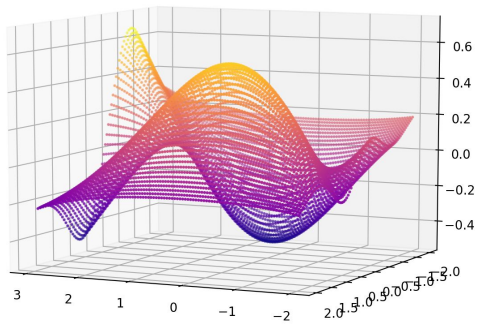
1 HLayers: ['5 N exp (a:1.72)'], eta: 0.1, test error: 0.018



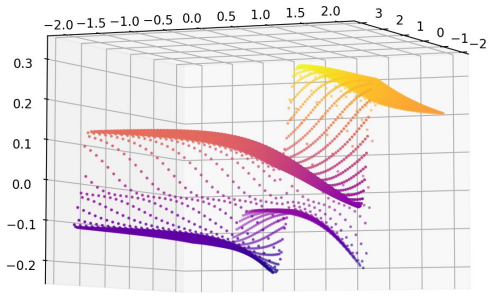
1 HLayers: ['6 N exp (a:1.72)'], eta: 0.1, test error: 0.0204



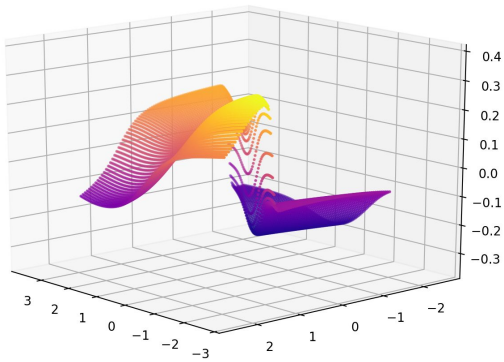
1 HLayers: ['7 N tanh (β:0.67)'], eta: 0.1, test error: 0.0069

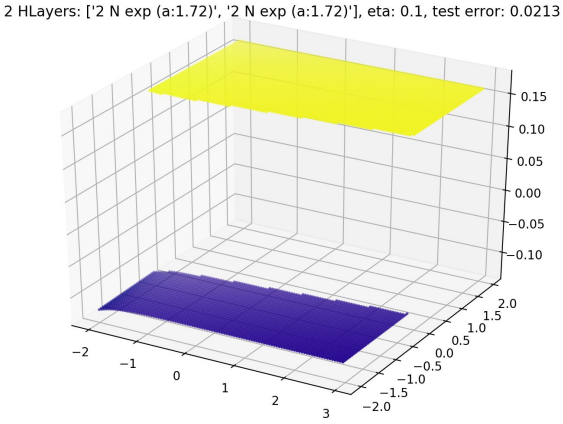


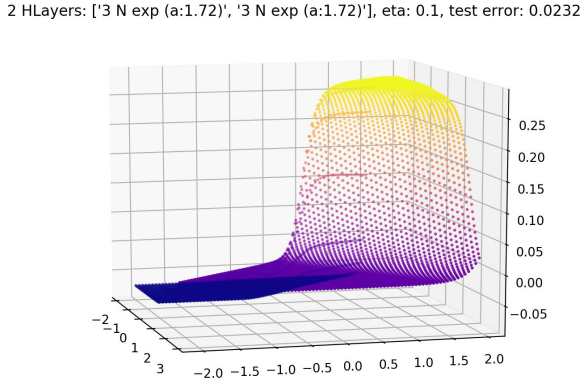
2 HLayers: ['3 N exp (a:1.72)', '3 N exp (a:1.72)'], eta: 0.1, test error: 0.019



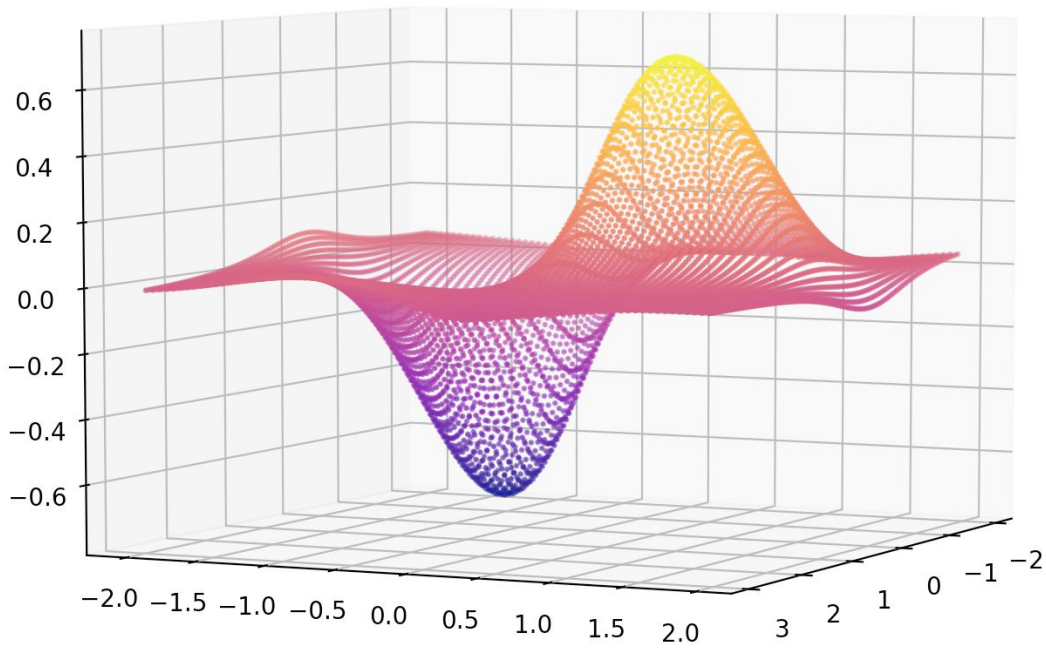
1 HLayers: ['7 N exp (a:1.72)'], eta: 0.1, test error: 0.0169






2 HLayers: ['4 N tanh (β:0.67)', '4 N tanh (β:0.67)'], eta: 0.2, test error: 0.0013



## Tabla de métricas

Para la obtención de las métricas se ejecutaron 50 corridas para cada arquitectura y cada $\eta$. Esto es para promediar el efecto que producen los pesos aleatorios de las capas. Cada arquitectura se entrenó 300 épocas. Se tomaron los errores mínimos obtenidos en cualquier corrida correspondiente a la misma arquitectura y $\eta$.

| Arquitectura | $\eta$ | Avg train epoch time [s] | Avg prediction time [s] | Train error | Test error |
|---|---|---|---|---|---|
| 1 capa 1 neurona c/ tanh | 0.5 | 0.0101 | 1.4e-05 | 0.04353578 | 0.03577855 |
| 1 capa 1 neurona c/ tanh | 0.2 | 0.0097 | 1.4e-05 | 0.0325732 | 0.02488577 |
| 1 capa 1 neurona c/ tanh | 0.05 | 0.0097 | 1.4e-05 | 0.03179772 | 0.02423627 |
| 1 capa 1 neurona c/ exp | 0.5 | 0.0122 | 1.8e-05 | 0.03847281 | 0.02600741 |
| 1 capa 1 neurona c/ exp | 0.2 | 0.012 | 1.7e-05 | 0.03202012 | 0.02377497 |
| 1 capa 1 neurona c/ exp | 0.05 | 0.0151 | 1.8e-05 | 0.03172445 | 0.02383026 |
| 1 capa 5 neurona c/ tanh | 0.5 | 0.0279 | 4.2e-05 | 0.04360958 | 0.0358624 |
| 1 capa 5 neurona c/ tanh | 0.2 | 0.0277 | 4.2e-05 | 0.03150736 | 0.02490031 |
| 1 capa 5 neurona c/ tanh | 0.05 | 0.0277 | 4.3e-05 | 0.00444531 | 0.00372065 |
| 1 capa 7 neurona c/ tanh | 0.5 | 0.0373 | 5.6e-05 | 0.04360958 | 0.0358624 |
| 1 capa 7 neurona c/ tanh | 0.2 | 0.0374 | 5.5e-05 | 0.03150733 | 0.02483044 |
| 1 capa 7 neurona c/ tanh | 0.05 | 0.0365 | 5.8e-05 | 0.00440126 | 0.00360745 |
| 2 capas 4 neuronas c/ tanh | 0.5 | 0.045 | 6.3e-05 | 0.0221924 | 0.02059892 |
| 2 capas 4 neuronas c/ tanh | 0.2 | 0.0448 | 7.2e-05 | 0.00311044 | 0.0031376 |
| 2 capas 4 neuronas c/ tanh | 0.05 | 0.0437 | 6.6e-05 | 0.00329216 | 0.00262524 |

Por lo obtenido en las métricas, vemos que utilizar la tangente hiperbólica como función de activación en las capas nos da resultados que

tienden a tener un error mayor comparado con la exponencial, pero es más rápida de calcular tanto en el entrenamiento como en las predicciones. Podríamos decir que el tiempo es despreciable entre ambas e inclinarnos a elegir la función exponencial para obtener resultados más precisos, sin embargo, viendo las gráficas, observamos figuras con formas completamente distintas a las del gráfico esperado. Como el enunciado de este trabajo tiene en cuenta que la red se utilizará para un videojuego, debemos elegir una arquitectura que use la tangente hiperbólica para obtener un resultado más realista. También observamos gráficamente la necesidad de utilizar 2 capas, esto se ve tanto en el error obtenido como en el gráfico.

# Config

Se puede configurar el comportamiento al correr el programa con el archivo de configuración situado en data/config.json

## Ejemplo

```
{
 "training_epochs": 1000,
 "input_network": "net_4_4_learning_test",
 "input_strategy": "first_half",
 "print_progress_every": 10,
 "trained_network_name": "net_4_4_learning_test_trained",
 "plot": ["error", "network_and_original"]
}
```

## Schema

En el siguiente JSON-Schema se describe el formato.

```
{
 "$schema": "http://json-schema.org/schema#",
 "id": "",
 "type": "object",
 "required": ["training_epochs", "input_network", "input_strategy",
"trained_network_name", "print_progress_every"],
 "properties": {
   "training_epochs": {
     "description": "La cantidad de epocas a entrenar",
     "type": "integer",
     "minimum": 0,
     "default": 0
   },
   "input_network": {
     "description": "Nombre del archivo de configuracion de red a utilizar.
Situado en .data/ y sin el .json",
     "type": "string"
   },
   "input_strategy": {
     "description": "Tipo de estrategia de selección de puntos de
entrenamiento",
     "enum": ["first_half", "second_half", "z_ascending", "z_descending"]
   },
   "print_progress_every": {
     "description": "Se muestra por stdout el progreso del entrenamiento con
este step",
```

```
      "type": "integer",
      "minimum": 0,
      "default": 10
    },
    "trained_network_name": {
      "description": "Nombre del archivo de configuracion de red saliente.
Situado en .data/ y sin el .json",
      "type": "string"
    },
    "plot": {
      "description": "Tipos de gráfico que se graficarán",
      "type": "array",
      "items": {
        "enum": ["network", "network_and_original", "error"]
      }
    }
  }
}
```

# Network Config

La configuración de la red neuronal se utiliza tanto para tener un estado inicial como para guardar el estado del sistema (config + estado). Es por eso que figuran parámetros como “epochs” que significa la edad de esta configuración.

## Ejemplo

```
{
  "network": {
    "epochs": 9000,
    "inputs": 2,
    "eta": 0.044197921801251365,
    "momentum": 0.0,
    "adaptive_bold": {
      "a": 0.01,
      "b": 0.1,
      "k": 3
    },
    "adaptive_annealing": null,
    "layers": [
      {
        "neurons": 4,
        "activation_function": {
          "type": "htan",
          "a": 1.7159,
          "beta": 0.6666666666666666
```

```
      },
      "neuron_weights": [
        {
          "bias": -0.20591978614651005,
          "weights": [
            0.9951387554652638,
            -0.09647242698418093
          ]
        },
        {
          "bias": -1.620504635974132,
          "weights": [
            -0.4517066178967171,
            -1.5930918132537881
          ]
        },
        {
          "bias": -1.313505379223587,
          "weights": [
            -1.003994621959052,
            1.3465042061892947
          ]
        },
        {
          "bias": 0.07201060071288082,
          "weights": [
            -0.13467716488985693,
            3.0807589290710284
          ]
        }
      ]
    },
    {
      "neurons": 4,
      "activation_function": {
        "type": "htan",
        "a": 1.7159,
        "beta": 0.6666666666666666
      },
      "neuron_weights": [
        {
          "bias": -0.5420010310065684,
          "weights": [
            -2.325918807251931,
            0.5521875391275459,
            -1.0754865131118738,
            0.8039083786726217
          ]
        },
        {
```

```
          "bias": -0.3408498992249129,
          "weights": [
            -1.4109295208712231,
            -0.317085397928441,
            0.4379965126899695,
            -0.7548456777541185
          ]
        },
        {
          "bias": 0.18966041257569854,
          "weights": [
            -0.24946632240631925,
            -0.5017496905886178,
            0.8054690579745496,
            -1.0414250214960896
          ]
        },
        {
          "bias": -0.8533115001377312,
          "weights": [
            -2.684718983911386,
            0.7034074921435034,
            0.714024560811789,
            0.6369634983334246
          ]
        }
      ]
    },
    {
      "neurons": 1,
      "activation_function": {
        "type": "linear"
      },
      "neuron_weights": [
        {
          "bias": -0.02204568207302905,
          "weights": [
            0.42455555298076975,
            -0.7779626626471162,
            0.142853126571317,
            0.22885414387344083
          ]
        }
      ]
    }
  ]
 }
}
```

## Schema

En el siguiente JSON-Schema se describe el formato.

```
{
 "$schema": "http://json-schema.org/schema#",
 "id": "",
 "type": "object",
 "required": ["inputs", "eta", "layers"],
 "properties": {
   "inputs": {
     "description": "Cantidad de inputs que tiene la red",
     "type": "integer",
     "minimum": 1
   },
   "eta": {
     "description": "El coeficiente de entrenamiento",
     "type": "number",
     "minimum": 0
   },
   "momentum": {
     "description": "El coeficiente de momento",
     "type": "number",
     "minimum": 0,
     "maximum": 1,
     "default": 0
   },
   "momentum_delta": {
     "description": "Delta a agregarle al coeficiente de momento",
     "type": "number",
     "minimum": 0,
     "maximum": 0.1,
     "default": 0.005
   },
   "momentum_epoch_increase": {
     "description": "Cada cuantas épocas se le quitará momentum_delta a
momentum",
     "type": "integer",
     "minimum": 1,
     "default": 50
   },
   "epochs": {
     "description": "Edad de la red (sólo estado)",
     "type": "integer",
     "minimum": 0,
     "default": 0
   },
   "adaptive_bold": {
     "description": "Configuracion de parametros adaptativos",
     "type": ["object", "null"],
```

```
    "required": ["a", "b", "k"],
    "properties": {
      "k": {
        "description": "Número de pasos previos para comprobar la progresión
del error (aumento o disminución)",
        "type": "integer",
        "minimum": 1
      },
      "a": {
        "description": "Valor para incrementar a eta en la disminución
consistente del error: eta = eta + a",
        "type": "number"
      },
      "b": {
        "description": "Valor para multiplicar y decrementar eta actual: eta =
eta - b * eta",
        "type": "number"
      }
    }
  },
  "adaptive_annealing": {
    "description": "Coeficiente de annealing",
    "type": ["integer", "null"]
  },
  "layers": {
    "description": "Configuración de las capas de la red",
    "type": "array",
    "items": {
      "description": "Configuración de las neuronas de la capa",
      "type": "object",
      "required": ["neurons", "activation_function"],
      "properties": {
        "neurons": {
          "description": "Cantidad de neuronas",
          "type": "integer",
          "minimum": 1
        },
        "activation_function": {
          "description": "Funcion de activación",
          "type": "object",
          "required": ["type"],
          "properties": {
            "type": {
              "enum": ["htan", "logistic", "linear", "sign", "step"]
            },
            "a": {
              "type": "number"
            },
            "beta": {
              "type": "number"
```

```
            }
          }
        },
        "neuron_weights": {
          "description": "Estado de cada neurona",
          "type": "array",
          "items": {
            "type": "object",
            "required": ["bias"],
            "properties": {
              "bias": {
                "type": "number"
              },
              "weights": {
                "type": "array",
                "items": {
                  "type": "number"
                }
              }
            }
          }
        }
      }
    }
  }
}
```