

Diseño Evolutivo de Redes Neuronales con Evolución Diferencial

Alfredo Gutiérrez Alfaro

27 de junio de 2023

Tabla de contenidos

Introducción

Objetivo

Redes neuronales

Aprendizaje evolutivo

Desarrollo

Métricas

Experimentos

Resultados

- A la hora de implementar redes neuronales para resolver un problema requiere de personas expertas para diseñar la topología de la red y parámetros, entre otros elementos de diseño. [7]
- Para resolver esta problemática del diseño, en la literatura se puede encontrar el uso de metaheurísticas bioinspiradas, como lo es el uso del Particle Swarm Optimization (PSO), Ant-Colony y la Evolución Diferencial. [6, 7, 2]

- Implementar y analizar la evolución diferencial para el diseño de redes neuronales y cómo se desempeñan en las tareas de clasificación.
- Comparar resultados con una red neuronal "tradicional" entrenada con el descenso del gradiente

Redes Neuronales

Las redes neuronales son un modelo matemático que intentan replicar de cierta forma a las neuronas biológicas.

$$\sigma\left(\sum_{i=1}^m x_i w_i + b\right) = \sigma(x^T w + b) = \hat{y} \quad (1)$$

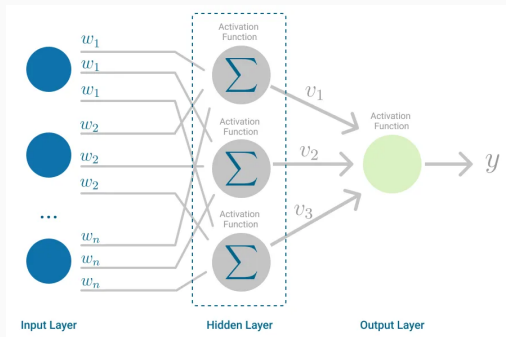


Figura 1: Ejemplo red neuronal [3]

Funciones de activación

Las funciones de activación introducen no linealidad dentro de una red neuronal, lo que le permite a la red realizar representaciones más complejas de los datos.

Nombre	Función
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$
Tanh	$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
Sinusoidal	$\sin(x)$
Linear	$f(x) = x$
Hard Limit	$f(x) = \begin{cases} 1 & \text{si } x \geq 0 \\ 0 & \text{si } x < 0 \end{cases}$
ReLU	$f(x) = \max(0, x)$
Leaky ReLU	$f(x) = \begin{cases} 0,1x & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases}$

Aprendizaje evolutivo

Los algoritmos evolutivos son un tipo de algoritmos de optimización heurísticos y aleatorizados, inspirados en la evolución natural. Simulan el proceso de evolución natural considerando dos factores clave: la reproducción variacional y la selección del más apto. [8]

La estructura básica de la mayoría de algoritmos evolutivos se puede resumir de la siguiente manera:

1. Generar un conjunto inicial de soluciones (llamado población).
2. Reproducir nuevas soluciones basadas en la población actual, mediante procesos como el cruce y la mutación.
3. Eliminar las peores soluciones de la población.
4. Repetir desde el paso 2 hasta que se cumpla algún criterio de parada.

La evolución diferencial es un algoritmo evolutivo que sirve para resolver problemas continuos. La ED utiliza una estrategia multiparental para generar posibles soluciones. El algoritmo se basa en la reproducción de uno o más individuos, reemplazando a los padres por hijos con mejor aptitud. [4]

Evolución diferencial

Pseudocódigo

Input : Number of individuals NP

Output: Optimized solution P

Generate $P = (x_1, x_2, \dots, x_{NP})$;

repeat

for $i = 1 \text{ to } NP$ **do**

 Compute a mutant vector v_i ;

 Create u_i by the crossover of v_i and x_i ;

if $f(u_i) < f(x_i)$ **then**

 Insert u_i into Q;

end

else

 Insert x_i into Q;

end

end

$P \leftarrow Q$;

until *stopping condition is satisfied*;

Evolución diferencial

Pseudocódigo

Input : Number of individuals NP

Output: Optimized solution P

Generate $P = (x_1, x_2, \dots, x_{NP})$;

repeat

for $i = 1 \text{ to } NP$ **do**

 Compute a mutant vector v_i ;

 Create u_i by the crossover of v_i and x_i ;

if $f(u_i) < f(x_i)$ **then**

 Insert u_i into Q;

end

else

 Insert x_i into Q;

end

end

$P \leftarrow Q$;

until *stopping condition is satisfied*;

Evolución diferencial

Pseudocódigo

Input : Number of individuals NP

Output: Optimized solution P

Generate $P = (x_1, x_2, \dots, x_{NP})$;

repeat

for $i = 1 \text{ to } NP$ **do**

 Compute a mutant vector v_i ;

 Create u_i by the crossover of v_i and x_i ;

if $f(u_i) < f(x_i)$ **then**

 Insert u_i into Q;

end

else

 Insert x_i into Q;

end

end

$P \leftarrow Q$;

until *stopping condition is satisfied*;

Evolución diferencial

Pseudocódigo

Input : Number of individuals NP

Output: Optimized solution P

Generate $P = (x_1, x_2, \dots, x_{NP})$;

repeat

for $i = 1$ **to** NP **do**

 Compute a mutant vector v_i ;

 Create u_i by the crossover of v_i and x_i ;

if $f(u_i) < f(x_i)$ **then**

 Insert u_i into Q ;

end

else

 Insert x_i into Q ;

end

end

$P \leftarrow Q$;

until *stopping condition is satisfied*;

Python Code

```
def fit(self, fitness, max_iter: int):
    self.dim = len(self.population[0])

    obj_all = [fitness(x) for x in self.population]
    best_vector = self.population[np.argmin(obj_all)]
    best_obj = min(obj_all)
    prev_obj = best_obj

    obj_iter = []
    for _ in range(max_iter):
        for i in range(self.pop_size):
            # Select three random individuals
            indexes = random.sample(
                [index for index in range(self.pop_size) if index != i], 3
            )

            candidates = [self.population[index] for index in indexes]
            mutated = self.mutation(candidates)
            mutated = self.check_bounds(mutated)

            trial = self.crossover(mutated, self.population[i])

            obj_target = fitness(self.population[i])
            obj_trial = fitness(trial)

            if obj_trial < obj_target:
                self.population[i] = trial

        obj_all = [
            fitness(x) for x in self.population
        ] # Update obj_all after modifying the population
        best_obj = min(obj_all)

        if best_obj < prev_obj:
            best_vector = self.population[np.argmin(obj_all)]
            prev_obj = best_obj
            obj_iter.append(best_obj)
            print(
                "Iteration: %d f([%s]) = %.5f"
                % (_, np.around(best_vector, decimals=5), best_obj)
            )

    return [best_vector, best_obj, obj_iter]
```

Metodología

Diseñar una red neuronal de tres capas con algoritmos evolutivos (evolución diferencial) con un esquema de codificación directo [6], que tiene en consideración los siguientes elementos:

1. Definir el número de neuronas en la capa oculta
2. Establecer funciones de activación
3. Generar conexiones sinápticas y pesos

Representación del problema

Definir el número total de neuronas en la red neuronal

$$Q = (M + N) + \frac{N + M}{2} \quad (2)$$

Definir el número de neuronas en la capa de entrada

$$H = Q - (M + N) \quad (3)$$

Obtener el tamaño de la dimensión del vector para representar el problema

$$\dim_d = [H * (N + 3)] + [M * (H + 3)] \quad (4)$$

Representación del problema

A continuación una muestra de cómo se representa la codificación para el diseño de las redes neuronales:

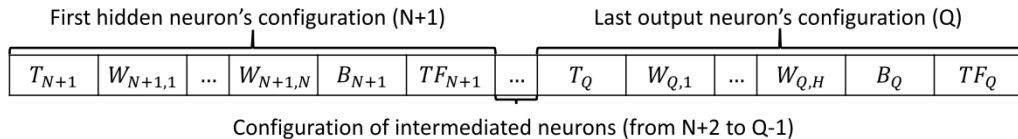


Figura 2: Esquema para representar los parámetros [2]

Para poder evaluar la red y utilizar una función a optimizar dentro de la evolución diferencial se hará uso de la exactitud y el error de esta:

$$Exactitud = \frac{VP + VN}{VP + VN + FP + FN} \quad (5)$$

$$error = 1 - exactitud \quad (6)$$

Se utilizaron las ecuaciones [2, 3, 4] para generar las neuronas (arquitectura) de una red neuronal para dos conjuntos de datos diferentes: Iris Plant [5] y Wine [1].

Posteriormente esta red fue codificadas para ser utilizadas por la evolución diferencial y así obtener la topología de la red así como sus parámetros para después evaluar la red con la exactitud.

Red neuronal (Iris Plant)

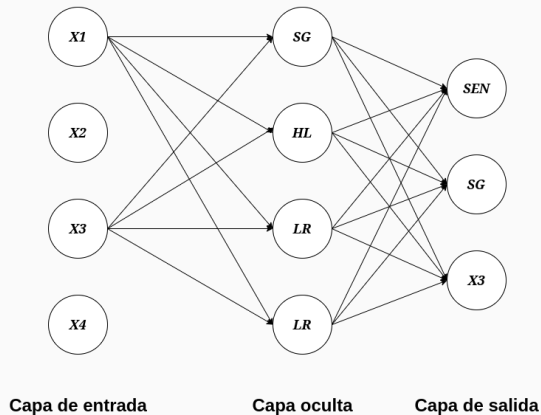


Figura 3: Arquitectura para Iris

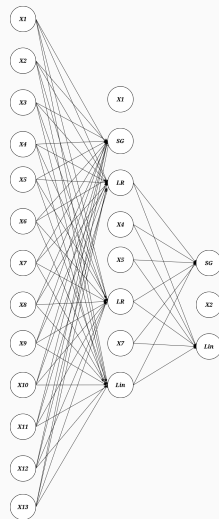
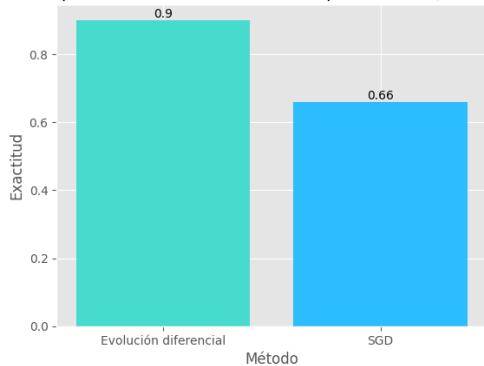


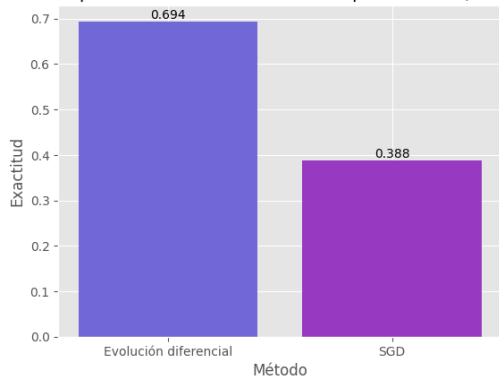
Figura 4: Arquitectura para Wine

Resultados



Comparación de redes neuronales por método (Iris Plant)








Comparación de redes neuronales por método (Wine)




- Las redes neuronales diseñadas por evolución diferencial llegan a tener mejores resultados que redes totalmente conectadas que utilizan el descenso del gradiente
- Las redes tienen mayor flexibilidad al ser diseñadas con una codificación directa

-  Aeberhard, S., Forina, M.: Wine. UCI Machine Learning Repository (1991), DOI: <https://doi.org/10.24432/C5PC7J>
-  Alba-Cisneros, O., Espinal, A., López-Vázquez, G., Sotelo-Figueroa, M.A., Purata-Sifuentes, O.J., Calzada-Ledesma, V., Vázquez, R.A., Rostro-Gonzalez, H.: Direct and Indirect Evolutionary Designs of Artificial Neural Networks. Springer Nature (1 2020). <https://doi.org/10.1007/978-3-030-35445-931>, https://doi.org/10.1007/978-3-030-35445-9_31

-  Bento, C.: Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis (1 2022),
<https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141>
-  Du, K.L., Swamy, M.N.S.: Search and Optimization by Metaheuristics. Birkhäuser (8 2016)
-  Fisher, R.A.: Iris. UCI Machine Learning Repository (1988), DOI:
<https://doi.org/10.24432/C56C76>

-  Garro, B.A., Vázquez, R.A.: Designing Artificial Neural Networks Using Particle Swarm Optimization Algorithms. Computational Intelligence and Neuroscience **2015**, 1–20 (1 2015). <https://doi.org/10.1155/2015/369298>, <https://doi.org/10.1155/2015/369298>
-  López-Vázquez, G., Ornelas-Rodríguez, M., Espinal, A., Soria-Alcaraz, J.A., Rojas-Domínguez, A., Puga, H., Carpio, M., Rostro-Gonzalez, H.: Evolutionary Spiking Neural Networks for Solving Supervised Classification Problems. Computational Intelligence and Neuroscience **2019**, 1–13 (3 2019). <https://doi.org/10.1155/2019/4182639>, <https://doi.org/10.1155/2019/4182639>

-  Zhou, Z.H., Yu, Y., Qian, C.: Evolutionary Learning: Advances in Theories and Algorithms. Springer (6 2019)