

# Algoritmos Genéticos

## Sistemas de Inteligencia Artificial - ITBA

Carlos Sessa

Lucas Pizzagalli

Nicolás Purita

### 1. INTRODUCCIÓN

Se implementa un motor de algoritmos genéticos para obtener los pesos para la red neuronal construida en el Trabajo Práctico Especial Número 2. La red neuronal resuelve la función que se puede observar en la figura 1.

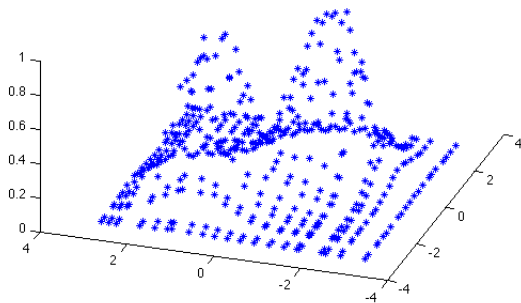


Figura 1: Distribución de puntos dada

En la figura 1 se puede observar que los puntos de la entrada pertenecen al intervalo  $[-3.5, 3.5]$  y la salida se encuentra en el intervalo  $(0, 1)$ .

El algoritmo genético se implementó en *Java* y la red neuronal fue realizada en *Matlab*. Utilizando el *MATLAB Compiler Runtime* se realizan las pruebas pertinentes para verificar el funcionamiento de la red.

### 2. DESARROLLO Y PROBLEMAS ENCONTRADOS

Se han implementado varios operadores genéticos, métodos de selección, condiciones de corte y demás configuraciones que se presentarán en las secciones a continuación. Con el objetivo de facilitar la configuración se ha creado un archivo de configuración en el que se debe detallar como se desea entrenar la red. Un ejemplo de dicho archivo se puede observar en 1.

#### 2.1 Operadores genéticos

##### 2.1.1 Mutación

Las mutaciones se realizan tomando un número aleatorio definido entre  $-1.5$  y  $1.5$ . Se han implementado las mutaciones del tipo clásicas y no uniformes, las cuales se les debe asignar cuando debe decrecer y cada cuantas generaciones debe hacerlo.

##### 2.1.2 Crossover

Se han decidido implementar los siguientes algoritmos de entrecruzamiento:

- cruce de un punto (clásica)
- cruce de dos puntos
- cruce anular
- cruce uniforme parametrizado
- Gene

##### 2.1.3 Backpropagation

Otro operador Backpropagation se agrega al set de operadores al tratarse de un problema que por sus características dicho algoritmo tiene una muy buena performance como ya pudo ser demostrado en coacciones anteriores. El mismo toma como parámetro la cantidad de épocas que debe entrenar la red.

### 2.2 Métodos de selección y reemplazo

Para ambos casos se han desarrollado los métodos descriptos a continuación, los cuales pueden ser combinados de a dos:

- Método de elitismo
- Método de torneos
- Método de Boltzman
- Método de selección universal estocástica

### 2.3 Distintas arquitecturas

Mediante el archivo de configuración se puede seleccionar la arquitectura que se desea utilizar para realizar las pruebas. Las arquitecturas a elegir son las siguientes:

1. [30 20 10 1]
2. [10 10 1]
3. [10 1]
4. [10 10 10 10 1]

5. [40 20 1]
6. [10 10 10 1]
7. [5 10 20 1]

## 2.4 Representación del individuo

Con el objetivo de representar al individuo, se ha elegido transformar la matriz a un vector de números *doubles* como se muestra en la imagen 2 y de esta forma poder aplicar los operadores pertinentes de una forma rápida y sencilla.

La representación más óptima del individuo fue representar a toda la red como un vector de *doubles*. Esta representación no se adapta a la representación de la matriz en *MATLAB*, por lo tanto se realiza una transformación del individuo. Esta transformación consiste en cambiar el vector de *doubles* por una matriz y viceversa.

### 2.4.1 Fidelidad del individuo

- **Complejidad:** Es completa ya que se puede representar todo el dominio del problema.
- **Coherencia:** Representa únicamente el dominio del problema, ya que ninguna representación puede pertenecer a un conjunto fuera del dominio.
- **Uniformidad:** Se puede concluir que esta representación es uniforme ya que es imposible representar dos individuos distintos con la misma cadena, en caso que esto sucediera esa representación sería exactamente la misma que la otra.
- **Sencillez:** Convertir matriz de pesos a un vector de *doubles* es una operación simple.
- **Localidad:** Es local dada que un cambio en un elemento del vector genera un cambio en el peso de la conexión que representa.

## 3. RESULTADOS Y CONCLUSIONES

Los resultados se presentaban en dos secciones distintas. En la primera no se utilizaba el operador *backpropagation* y la segunda sí.

Con el fin de obtener resultados comparables entre sí se define un contexto base para todas las pruebas de cada sección. Luego en cada prueba se reemplaza el caso base con lo que se desea. Debe tenerse en cuenta que el hecho de fijar algunos parámetros puede hacer que los distintos métodos y operadores a comparar se vean beneficiados o perjudicados por su funcionamiento en conjunto.

Cabe destacar que aparte de los operadores pedidos por la cátedra se implementó un operador de cruce adicional al que denominamos *Gene*. *Gene* funciona de manera similar al cruce clásico pero en vez de realizar el cruce en base a un *locus* lo hace reemplazando capas enteras.

### 3.1 Sin backpropagation

Primero se intenta resolver el problema sin usar el operador *backpropagation*. El contexto base usado en esta sección es el siguiente:

- Tamaño de la población es de 52 individuos.

- La brecha generacional (*G*) es de 0.5.
- La arquitectura elegida para realizar las pruebas es la de [10 10 1].
- El operador de mutación inicial es *Clásico* con una probabilidad de 0.5 de mutar.
- El operador de cruce inicial es *Clásico* con una probabilidad de 1 de cruce.
- El operador *Backpropagation* se encuentra desactivado es decir que se ejecute con probabilidad 0.
- El método de selección es *Elite*.
- El método de reemplazo es *Elite* al igual que el de selección.
- Hay dos condiciones de corte, ya sea por *Máxima cantidad de generaciones* (500) y/o *Contenido* (donde la última verifica que si en 50 épocas no mejoró más de un 0.01 el fitness del individuo finaliza su ejecución).

#### 3.1.1 Mutación clásica

Primero se prueba variar la probabilidad de la mutación clásica. Los resultados se pueden observar en la tabla 1.

#### 3.1.2 Mutación no uniforme

Luego se prueba variar la probabilidad de la mutación no uniforme. Los resultados se pueden observar en la tabla 2.

#### 3.1.3 Cruce

Por último se corre el algoritmo usando distintos cruces. Los resultados se encuentran en la tabla 3.

#### 3.1.4 Análisis de resultados

Después de probar con distintas configuraciones, notamos que sin el operador de *backpropagation* no conseguimos buenos resultados.

Entendemos que esto sucede porque el individuo es muy largo haciendo que el algoritmo tarde mucho en encontrar un resultado útil. En la próxima sección analizaremos qué pasa cuando agregamos el operador *backpropagation*.

## 3.2 Con backpropagation

En esta sección se prueban distintas configuraciones utilizando el operador *backpropagation*. Se puede observar en la tabla 4 que la mejor configuración para los métodos de selección es con *Elite* y *Boltzman*. Para el reemplazo se decidió como métodos a *Elite* y *Ruleta*. Cabe destacar que el mejor operador de cruce fue *Gene* y de mutación el *Clásico*. Es importante remarcar que la diferencia entre la utilización del operador de *backpropagation* y la no utilización, provoca una gran mejora.

## 4. POSIBLES MEJORAS

### 4.1 Mejoras a backpropagation

Como posible mejora se concluye que el operador *backpropagation* podría disminuir su probabilidad de acción a medida que se alcanza un fitness deseado. El motivo por el cual se desea disminuir esa probabilidad es para que comience a tener más influencia los operadores de mutación y cruce. El deseo de que la mutación y cruce tenga más influencia a lo

largo de las generaciones es porque se observó que el fitness en un punto se asintotiza al error cuadrático medio obtenido por *backpropagation* en el Trabajo Práctico Especial 2.

## 4.2 Mejoras a criterios de selección y reemplazo

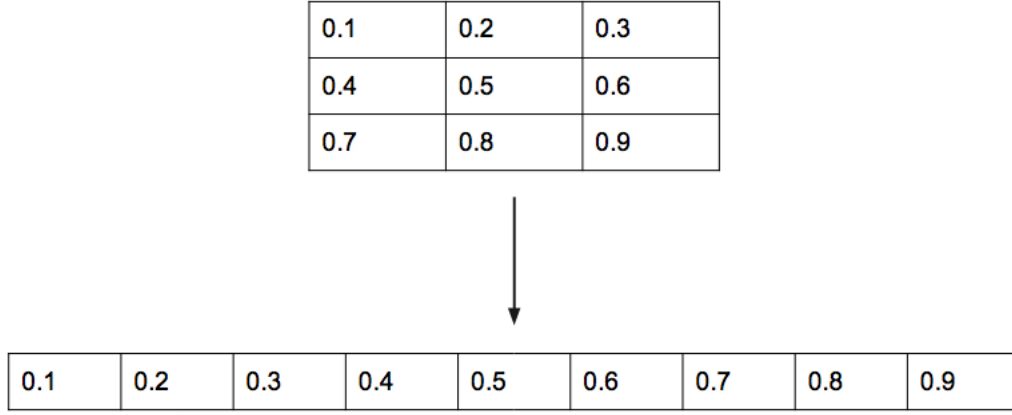
En algunas corridas nos pasó que perdimos al mejor individuo entre generaciones. Sería conveniente agregar un parámetro a los criterios de selección y reemplazo para que siempre dejen el individuo con mejor aptitud.

## 4.3 Mutación evolutiva

Se podría probar evolucionar la probabilidad de mutación de cada individuo de la población. En vez de tomarse una probabilidad de mutación global, se agrega al genotipo de cada individuo un gen que codifica la probabilidad de mutación del resto del cromosoma. En otras palabras, en vez de fijar la probabilidad de mutación arbitrariamente, la probabilidad de mutación tiene libertad de moverse según mejor le convenga a cada individuo para mejorar su fitness.

## 5. CONCLUSIONES

Finalmente, se llega a la conclusión de que, debido a la naturaleza del problema que se desea resolver, es necesaria la utilización del operador *backpropagation* para llegar a resultados próximos a los deseados.



**Figura 2: Transformación de la matriz en vector**

Fitness	Generación	Corte	$p_m$
0.22501570790437642	60	Contenido	0.05
0.21842640558999668	60	Contenido	0.1
0.22113522454399956	67	Contenido	0.3
0.22050756611447994	72	Contenido	0.5

**Tabla 1: Configuración simple variando la  $p_m$**

Fitness	Generación	Corte	$p_m$	decrecimiento (%)	dec. gen.
0.21222349890474348	50	Content	0.6	5	30
0.22795744299992385	77	Content	0.6	10	30
0.21740760693242447	65	Content	0.6	15	30

**Tabla 2: Configuración simple método de mutación No Uniforme**

**Listing 1: Archivo de configuración simple**

```

popSize = 52
generationGap = 0.6
architecture = 2

mutation = Classic
mutationProbability = 0.05

Backpropagation.probability = 0.15
Backpropagation.iterations = 30

crossover = Classic
crossoverProbability = 1

selection = Elite / Roulette
Elite.toSelect = 16
Roulette.toSelect = 15

replacement = Elite / Boltzman
replacement.Elite.toSelect = 15
replacement.Boltzman.toSelect = 6
replacement.Boltzman.maxTemperature = 100
replacement.Boltzman.minTemperature = 7

```

Fitness	Generación	Corte	$p_m$	Cruce
0.22501570790437642	60	Contenido	0.05	Clásico
0.20662153685787393	50	Contenido	0.05	Gene
0.21418198155450324	50	Contenido	0.05	Multiples puntos con 2
0.21674758534799746	70	Contenido	0.05	Uniforme con 0.3 prob
0.20662153685787393	50	Contenido	0.05	Anular

**Tabla 3: Configuración simple variando el método de cruce**

Nombre .properties	Fitness	Generación	Corte
bp_seb_rer_cg_mc_ec.properties	16.720276062651436	69	Contenido
bp_ser_reb_mc_cc_emc.properties	11.941655508072634	58	Contenido
bp_se_re_mc_cc_emc.properties	4.825691593990156	51	Contenido
bp_seb_rer_cmp_mc_emc.properties	12.57897997517028	36	Contenido
bp_seb_rer_mc_ca_emc.properties	13.560979676949982	50	Contenido
bp_seb_rer_mnu_cc_emc.properties	12.992838764610976	36	Contenido

**Tabla 4: Distintas Configuraciones con backpropagation**

`replacement.Boltzman.decrement = 0.8`

`ending = MaxGeneration / Content`  
`MaxGeneration.iterationToEnd = 100`  
`Content.improvement = 0.01`  
`Content.iterationToImprove = 15`