laSalle

UNIVERSITAT RAMON LLULL

Escola Tècnica Superior d'Enginyeria La Salle

Treball Final de Màster

Màster Universitari en Enginyeria Informàtica i la seva gestió

HyPER: un control neuronal pel simulador TORCS

Alumne
Enric Costa Riera

Professor Ponent
Francesc Teixido Navarro

ACTA DE L'EXAMEN
DEL TREBALL FI DE CARRERA

Reunit el Tribunal qualificador en el dia de la data, l'alumne

D. Enric Costa Riera

va exposar el seu Treball de Fi de Carrera, el qual va tractar sobre el tema següent:

HyPER: un control neuronal pel simulador TORCS

Acabada l'exposició i contestades per part de l'alumne les objeccions formulades pels Srs. membres del tribunal, aquest valorà l'esmentat Treball amb la qualificació de

Barcelona,

VOCAL DEL TRIBUNAL VOCAL DEL TRIBUNAL

PRESIDENT DEL TRIBUNAL

Abstract

HyPER (Hybrid backPropagated Evolutive Runner) és un sistema d'aprenentatge artificial que entrenarà un cotxe en un simulador de forma que aquest sigui capaç de córrer en un circuit de forma competitiva. Aquest s'entrenarà mitjançant una xarxa neuronal que a la primera fase d'aprenentatge utilitzarà un algoritme genètic i la segona fase, que refinarà els resultats obtinguts a la primera, utilitzarà una variació de l'algoritme de *backpropagation*. Per acabar, es compararà el resultat amb altres controladors que s'han desenvolupat amb diverses tècniques d'aprenentatge.

Resum

Les curses de cotxes intel·ligents sobre simulador es poden veure als grans congressos sobre intel·ligència artificial internacionals, per exemple al GECCO. Tot va començar quan al 2008 un grup d'organitzadors d'aquests congressos van pensar que seria una forma divertida d'investigar en camps d'aprenentatge artificial muntar una competició on gent de tot el món s'enfrontessin entre ells en una cursa i actualment, al 2011, ha continuat amb la idea.

El que es vol fer en aquest projecte és un pilot intel·ligent que sigui capaç d'aprendre conduir per un circuit. Aquest ho haurà de fer d'una manera competitiva i eficient, minimitzant aquelles accions que el puguin perjudicar.

Això es farà utilitzant tots els coneixements d'intel·ligència artificial apresos durant la carrera i cercant a altres fonts literatura que ens ajudi a ampliar els coneixements de forma que amb aquests coneixements i una mica d'enginy es pugui arribar a desenvolupar alguna nova tècnica que ofereixi alguna millora substancial. Un cop s'hagi realitzat el sistema s'entrenarà en els mateixos circuits que es solen utilitzar en congressos internacionals per a posteriorment poder comparar els resultats obtinguts amb els d'altra gent que hagi afrontat el mateix problema.

La tècnica que es proposarà serà una xarxa neuronal híbrida, que entrenarà tant per *backpropagation*, com utilitzant un algoritme genètic. Amb això s'intentarà aconseguir els avantatges de cada tècnica alhora que s'eliminen els seus punts dèbils i així poder aconseguir uns resultats notables en la conducció del cotxe.

Finalment es valorarà tot el treball realitzat, ja sigui a nivell de desenvolupament del projecte com a nivell científic. De fet en aquestes valoracions es revisaran els objectius que es defineixin i es podrà veure si s'han complert o no. Per acabar, es deixaran obertes possibles línies de futur que poden donar peu a noves línies d'investigació.

Índex

Capítol 1

Introducció

La intel·ligència artificial és un tema que sempre ha interessat la humanitat, des de sempre s'ha volgut aconseguir una màquina que pensés per si sola. De fet, la primera màquina dotada d'intel·ligència artificial data de l'època dels grecs que van ser capaços de crear un sistema per a regular automàticament el flux d'aigua.

Si anem avançant per la història podem trobar molts altres exemples fins arribar al dia d'avui. Actualment una gran quantitat de màquines estan dotades d'intel·ligència: cases amb sistemes domòtics que són capaces de fer algunes tasques per elles mateixes, cotxes que aparquen sols, pàgines d'Internet que són capaces de sense demanar res oferir-nos anuncis a productes que realment ens interessen i així amb una infinitat de casos.

I no només ens trobem la intel·ligència artificial en la vida real, sinó que la literatura, cinema i altres arts han aconseguit dur-la a l'extrem. Pel·lícules que tracten de naus intel·ligents que poden inclús rebel·lar-se contra els tripulants, robots policies capaços de detenir a qualsevol, la gran quantitat de robots proposats per Isaac Asimov,...

Es pot veure que aquest és un camp molt explotat i, per això, amb aquest projecte es vol aportar un granet de sorra per a intentar apropar cada vegada més la realitat a la ciència ficció.

1.1 Marc i motivacions

L'origen del projecte ve de les línies de futur obertes en el treball final de carrera que consistia en realitzar un controlador [3] pel simulador TORCS que utilitzava un algoritme genètic per a optimitzar les constants d'un conjunt de funcions que indicaven com el cotxe havia de conduir. De fet el que es deia era:

' ... seria interessant utilitzar xarxes neuronals, que són especialment útils quan s'ha de tractar amb dades amb valors reals limitats i es requereix una sortida amb valor real. També es podria fer un sistema híbrid on un algorisme genètic entrenés la xarxa neuronal per tenir millors resultats'

Un altre motiu, també de gran pes, que va motivar a fer aquest projecte és que el controlador segueix l'estàndard de la **Simulated Car Racing Championship** que s'ha dut a terme en els grans congressos (GECCO, CIG, CEC...) de computació evolutiva i intel·ligència artificial durant els últims anys. Això ens va fer plantejar el repte de veure fins quin nivell podíem arribar respecte els experts en la matèria.

Figura 1.1: Logo del congrés GECCO de l'any 2011

1.2 Abast

Aquest projecte vol construir un sistema d'autoaprenentatge per a simular la conducció d'un pilot. Per a aconseguir-ho, es vol aprofundir en alguna tècnica que permeti fer-ho i cercar millores que puguin convertir la tècnica triada en una tècnica més potent.

A nivell teòric, es vol aprofundir en la tècnica de les xarxes neuronals, a més també es tractarà el tema dels algoritmes genètics, ja que en un moment donat serà la tècnica utilitzada per a entrenar la xarxa. D'aquesta forma es veurà el funcionament d'aquestes tècniques, usos, avantatges i inconvenients i s'analitzarà si ha sigut un encert utilitzar-les o no.

Com s'ha dit a l'apartat anterior, existeixen un gran nombre de competicions que utilitzen part del sistema utilitzat en aquest projecte per a treballar. Per això, amb els resultats que s'obtinguin es vol fer una comparativa del sistema que es realitzi amb d'altres per a intentar demostrar que s'està a l'alçada de competidors experts en el seu camp.

Un objectiu implícit en aquest projecte, és aconseguir l'expertesa en programació d'algoritmes en JAVA i en l'optimització d'aquests. El motiu es que el sistema que es vol fer ha d'avaluar a temps real la situació en la que es troba el cotxe i donar una resposta sense perdre temps. Per poder complir aquest objectiu s'haurà de seguir una metodologia correctament i donar èmfasi a les fases de disseny, per a no cometre errors fàcilment evitables que facin perdre una gran quantitat de temps.

En resum, els objectius són:

- Assolir uns coneixements elevats sobre tècniques d'intel·ligència artificial com les xarxes neuronal i els algoritmes genètics.
- Dissenyar, implementar i verificar el funcionament correcte d'un algoritme que sigui capaç de competir en una cursa de cotxes.
- Optimitzar l'eficiència del sistema construït de forma que un codi que fa el mateix, ho faci més ràpid.

1.3 Estructuració de la memòria

El capítol 1 d'aquest projecte vol ser una breu introducció del treball realitzat; posant-lo en context, especificant els objectius i finalment explicant com es farà. A continuació, en el capítol 2 s'explicarà la metodologia que s'ha seguit en el transcurs d'aquest projecte.

Al capítol 3 es farà una breu introducció en el món de la intel·ligència artificial, explicant els conceptes bàsics i la seva realització. Posteriorment s'aprofundirà en l'aprenentatge artificial i s'acabarà entrant en detall amb els algoritmes genètics i les xarxes neuronals, ja que seran les tècniques utilitzades en el desenvolupament de l'aplicació.

El capítol 4 explicarà el simulador TORCS i es donarà una breu guia sobre la seva instal·lació, ús,... Aquest capítol també descriurà a grans trets l'arquitectura sobre la que es treballarà, el protocol de comunicació i el format dels missatges que s'hi envien.

Amb els conceptes teòrics assolits, el capítol 5 explicarà la realització pràctica del projecte seguint la metodologia que s'explica en el capítol 2. Aquest capítol defineix diverses fases on les primeres consisteixen en la construcció del nucli intel·ligent del cotxe i aquestes s'integren al sistema global fins aconseguir un client intel·ligent capaç d'aprendre a conduir.

Per acabar, en el capítol 6 es farà una valoració dels resultats i a continuació, al capítol 7 s'extrauran les conclusions d'aquest projecte, ja siguin de caire metodològic, científic com personals. A més, en el capítol 8 es deixaran obertes algunes línies de futur que deixen a l'aire noves vies d'investigació.

Capítol 2

Metodologia

Aquest projecte, com es veurà més endavant, consta de diverses fases diferenciades entre sí. Per això s'ha decidit utilitzar una metodologia en espiral, fent servir una model de cascada en cada iteració.

Les fases d'un projecte són [1]:

- **Requeriments:** Consisteix en extreure els requisits i requeriments d'un software en una primera etapa per a crear-lo. Aquest punt és crucial, ja que una mala recollida de requeriments pot portar al fracàs del projecte al encarar-lo inicialment d'una forma incorrecta.
- **Anàlisi:** És entendre amb què estem tractant, és a dir, els requeriments. En aquesta fase habitualment es detecten les entitats rellevants, les seves propietats i les seves relacions.
- **Disseny:** Aquesta fase consisteix en dissenyar els components de la aplicació i descriu com es construirà la aplicació.
- **Especificació:** Es tracta de descriure el comportament dels components especificats en el disseny.
- **Implementació:** Consisteix en escriure el codi corresponent respectant el disseny i l'especificació.
- **Proves:** En aquesta fase s'ha de verificar que tot funciona com s'ha definit en els requeriments. És una fase vital, ja que en aquest cas aquesta fase indicarà si es pot avançar a la següent iteració o no.
- **Documentació:** Fase que consisteix en generar tots els manuals, documents i material necessari que servirà per a comprendre el funcionament del sistema.
- **Manteniment:** Consisteix en mantenir el programa treballant i solucionar els errors que es vagin trobant durant la vida d'aquest. Aquesta fase inclou

la detecció d'errors, com escoltar les deficiències que té el programa i atendre nous requeriments definits pels usuaris.

Degut a la natura del sistema que s'ha d'implementar en aquest projecte, algunes les fases no es poden dur a terme o bé s'han agrupat a d'altres. Així que per cada fase del projecte es trobaran les etapes de recollida de requeriments, disseny, implementació i proves (figura 2.1 descrites al capítol 5. No obstant, no s'indicarà la documentació en el capítol 5, ja que aquest document és la documentació del projecte.

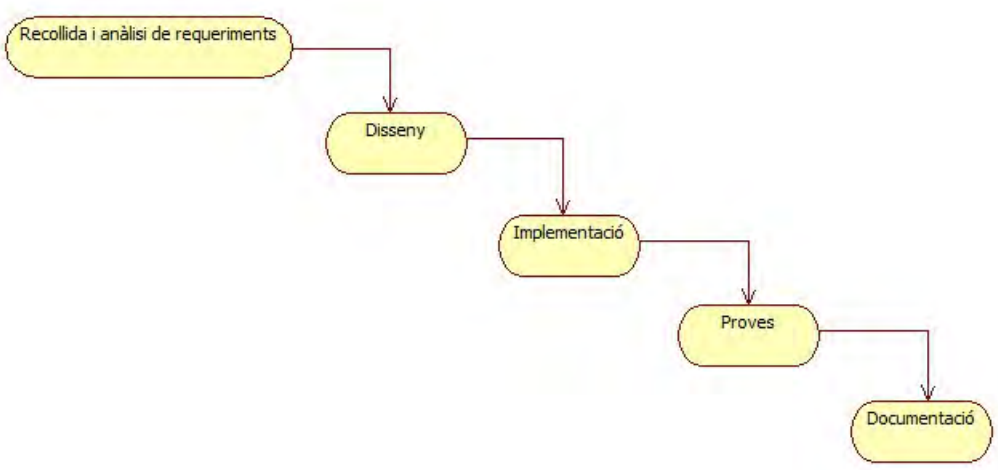

Figura 2.1: Metodologia en cascada

Però com s'ha dit al principi d'aquest capítol, en aquest projecte s'utilitzarà una metodologia en espiral [2]. Aquesta metodologia consisteix en, com mostra la figura 2.2 reiterar durant les vegades que sigui necessari la cascada de etapes explicada anteriorment.

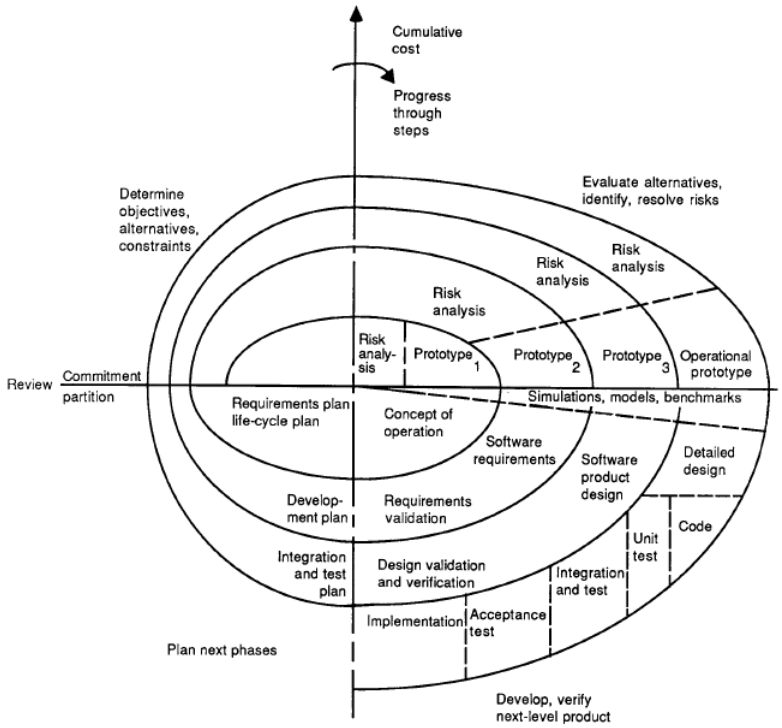

Figura 2.2: Model en espiral del desenvolupament de software [2]

En aquest model és fa l'anàlisi dels riscos de forma clara i explicita de forma que es redueixen els riscos del projecte, a més, és possible tenir en compte millores i nous requeriments sense trencar amb la metodologia, ja que aquest cicle de vida no és estàtic. No obstant, aquest model augmenta molt el cost temporal de projecte, un inconvenient que no ha afectat a l'hora selecció de la metodologia a utilitzar.

Capítol 3

Conceptes

3.1 Intel·ligència artificial

'Una màquina és capaç de pensar?' es va formular el matemàtic Alan Turing a l'any 1950. Segons ell, un sistema intel·ligent era aquell que podia imitar el comportament humà.

Actualment s'anomena intel·ligència artificial a la branca de la informàtica que se n'encarrega de desenvolupar sistemes que intenten reproduir els processos de la intel·ligència humana, de fet, es pot definir un sistema intel·ligent com a aquell que emula les capacitats racionals i és capaç de prendre les decisions, segons un coneixement adquirit, més correctes en cada circumstància.

3.1.1 Tipus de intel·ligència artificial

La intel·ligència artificial es divideix en dues escoles [10]:

- **Intel·ligència artificial convencional:** Es coneix també com intel·ligència artificial simbòlico-deductiva. Està basada en l'anàlisi formal i estadístic del comportament humà vers els problemes.
- **Intel·ligència artificial computacional:** També coneguda com a intel·ligència subsimbòlico-inductiva. Aquesta implica el desenvolupament o l'aprenentatge interactiu, com per exemple modificacions interactives de paràmetres en sistemes connexionistes que s'explicaran més endavant. En aquest cas l'aprenentatge es basa en dades empíriques.

3.1.2 Camps de la intel·ligència artificial

La intel·ligència artificial es pot classificar segons el seu àmbit en les següents categories [10]:

- **Programació automàtica:** Aquest camp pretén generar software automàticament, és a dir, un programa intel·ligent fa un altre programa només sabent les entrades i les sortides que es volen. Es poden definir dues formes de generar software:
  - **Sistema deductiu:** Es genera el codi a partir d'especificacions.
  - **Sistema inductiu:** Es genera el codi a partir d'exemples d'ús.
- **Raonament automàtic:** Aquest àmbit es basa en que un sistema informàtic sigui capaç de resoldre un problema que requereixi de raonament.
- **Aprenentatge artificial:** Es centra en sistemes que millorin automàticament amb l'experiència.
- **Processament del llenguatge natural:** Aquest camp té com objectiu investigar en sistemes per a fer possible la comunicació entre sistemes informàtics i els éssers humans mitjançant el llenguatge natural.
- **Conversió de llenguatge natural a codi:** La finalitat d'aquest camp és que a partir d'unes instruccions en llenguatge natural, un sistema informàtic sigui capaç de generar un programa que resolgui el problema presentat.
- **Interpretació d'imatges:** Aquest camp estudia la interpretació de imatges per a aconseguir extreure coneixement d'aquestes.
- **Sistemes agents o multiagents:** Aquest camp seria comparable a tenir diversos processos en un sistema operatiu, que fossin intel·ligents i interactuessin entre ells. D'aquests n'hi ha de dos tipus:
  - Cooperatius
  - Competitius

3.2 Aprenentatge artificial

L'aprenentatge artificial és un dels àmbits de la intel·ligència artificial. Aquest investiga la creació de sistemes que milloren automàticament amb l'experiència. L'objectiu principal és millorar la automatització de processos d'enginyeria de forma que els sistemes intel·ligents realitzin el mateix treball que un ésser humà optimitzant temps.

3.2.1 Classificació de l'aprenentatge artificial

Els algorismes d'aprenentatge artificial es classifiquen segons les tècniques d'aprenentatge:

- **Aprenentatge supervisat:** Aquest tipus d'aprenentatge serveix per a trobar una funció de correlació a partir d'unes dades d'entrenament. Aquest tipus d'aprenentatge s'utilitza, per exemple, en la detecció d'escriptura manual.
- **Aprenentatge no supervisat:** Es pretén trobar estructures dins de les dades d'entrada i agrupar-les o classificar-les. El resultat que donarà el sistema serà una agrupació amb el conjunt de valors que s'ha utilitzat com a entrenament.
- **Aprenentatge semi-supervisat:** Aquest aprenentatge és una barreja entre els dos explicats anteriorment, amb l'objectiu de aconseguir uns resultats que siguin similars als de l'aprenentatge supervisat sense necessitat que un ésser humà hagi d'introduir les dades d'entrenament manualment.
- **Aprenentatge per reforçament:** Està enfocat a sistemes capaços d'interactuar amb el seu entorn, els resultats de les accions fetes en aquest entorn es quantifiquen segons l'èxit d'aquestes.

3.2.2 Paradigmes de l'aprenentatge artificial

L'aprenentatge artificial es divideix en diversos paradigmes, cadascun d'ells utilitza un conjunt de tècniques. A pesar que tots segueixen diferents mètodes d'aprenentatge, tots acaben formulant un problema, determinant la representació, entrenant-se i avaluant el coneixement adquirit. Els diversos paradigmes són:

- **Aprenentatge artificial connexionista:** L'objectiu d'aquest paradigma és dissenyar elements neuronals de processament paral·lel, d'una forma que s'intenta imitar el comportament del cervell humà. S'associa a les xarxes neuronals.
- **Aprenentatge artificial analògic:** També conegut com aprenentatge basat en casos. Aquest intenta recordar la resolució d'antics problemes semblants

al que se'ns presenta per a aconseguir una solució tal com ho faria un ésser humà.

- **Aprenentatge artificial evolutiu:** Aquest paradigma engloba un conjunt de tècniques que simulen la evolució natural. La característica principal és que es tracta d'un aprenentatge col·laboratiu on a partir de poblacions d'individus on cada individu representa una possible solució d'un espai de cerca, s'ha d'arribar a una solució final. El conjunt de les tècniques que s'han anomenat el formen:
  - **Estratègies evolutives:** S'utilitzen operadors com l'encreuament i la mutació per anar generant descendència. No obstant, l'operador de selecció és determinista i la mida de la població dels pares és diferent de la mida de la població dels fills.
  - **Algorismes genètics:** Al contrari que en les estratègies evolutives s'utilitza una selecció probabilística i la probabilitat de l'operador mutació és més baixa. Aquests codifiquen les solucions en cadenes binàries o reals i no treballen directament sobre l'espai de cerca.
  - **Programació genètica:** És una variació dels algorismes genètics on l'objectiu és construir programes sense ser dissenyats, ni implementats per l'ésser humà. A pesar d'això funciona igual que els algorismes anteriors.
- **Aprenentatge artificial inductiu:** l'objectiu de l'aprenentatge inductiu és generar un model a partir d'exemples específics, és a dir, a partir d'un conjunt d'exemples s'anomena el conjunt d'entrenament.
- **Aprenentatge artificial deductiu:** Aquest paradigma representa el coneixement com a regles en forma lògica. No interacciona amb cap font de dades, ja que simplement funciona amb el conjunt de regles especificat.

3.3 Algorismes genètics

En la natura tots els individus d'una població competeixen entre sí en la cerca de recursos com aigua, aliments i refugi. Inclús els membres d'una mateixa espècie competeixen moltes vegades en la cerca de companyia. Aquells individus que tenen més èxit en sobreviure i en atraure un gran nombre de companys tenen més probabilitat de generar un major nombre de descendents. En canvi, aquells individus amb un índex més baix de supervivència produiran un menor nombre de descendents. Això significa que els gens dels individus millor adaptats es propaguen amb més facilitat, de forma que la combinació de bons gens durant moltes generacions pot acabar creant una població de superindividus que tinguin una adaptabilitat molt més gran que es seus avantpassats. D'aquesta forma, les espècies evolucionen aconseguint característiques que els converteixen en éssers millor adaptats en l'entorn en el que viuen.

Els algorismes genètics [9] són mètodes adaptatius que, principalment, s'utilitzen per resoldre problemes de cerca i optimització. Aquests estan basats en el procés genètic dels organismes vius. Durant el transcurs de les generacions, les poblacions evolucionen en la natura d'acord els principis de la selecció natural i la supervivència dels més forts. Així, imitant aquests processos, els algorismes genètics són capaços de crear solucions per a problemes del món real i la evolució d'aquestes solucions acaben tendint cap als valors òptims del problema depenent de la representació que s'hagi utilitzat.

Aquests algorismes treballen amb una població d'individus, on cada un d'ells representa una solució factible a un problema donat. A cada individu se li assigna una puntuació en relació a lo bona que sigui aquella solució al problema; això en comparació a la natura representaria el grau d'efectivitat per aconseguir els recursos necessaris per a sobreviure. Quan més gran sigui aquesta puntuació, major serà la probabilitat de que aquest sigui seleccionat per sobreviure i també tindrà possibilitats de creuar els seus gens amb un individu seleccionat de la mateixa forma. Aquests creuament crearan descendent dels anteriors individus que compartiran algunes característiques dels seus pares. Ara bé, quan menor sigui la adaptació d'un individu, menor serà la probabilitat de que aquest individu sigui seleccionat per a la reproducció, de forma que durant el transcurs de les generacions els seus gens desapareixeran.

D'aquesta manera es produeix una nova població de possibles solucions que substituirà la anterior i verificarà la propietat de que globalment aquesta població és millor que la població anterior. Així durant diverses generacions, les bones característiques s'aniran propagant a través de les poblacions resultants; i afavorint el creuament entre individus cada vegada millor adaptats es veu que les solucions es concentraran en les zones més òptimes de l'espai de cerca. De fet, si l'algorisme genètic ha estat ben dissenyat, la població convergirà cap a la millor solució del problema.

La potència dels algorismes genètics prové del fet de que es tracta de una tècnica

```
inicialitzacio()
avaluacio()
for each GENERACIO do
  seleccio()
  crossover()
  mutacio()
  avaluacio()
end for
```

Figura 3.1: Pseudocodi d'un algorisme genètic

robusta i poden tractar amb èxit una gran quantitat de problemes de diferents àrees, inclús en aquelles que altres mètodes tindrien dificultats. Encara que aquest algorisme no assegura que la solució trobada sigui la òptima, sempre es troben solucions d'un nivell acceptable en un temps relativament competitiu en relació amb altres algorismes d'optimització combinatòria. No obstant, en el cas de que existeixin tècniques especialitzades per resoldre un problema, el més probable és que el superin en velocitat com en eficàcia, ja que el camp dels algorismes genètics es relaciona amb aquells problemes per als quals no existeixen tècniques específiques. De fet, encara que existeixin, es poden crear millores de aquestes tècniques fent un híbrid amb algorismes genètics.

3.3.1 Estructura

L'estructura d'un algorisme genètic és la que es mostra a la figura 3.1.

Bàsicament consisteix en generar una població inicial i per cada generació fer quatre passos:

- **Selecció:** Es seleccionen els individus seguint alguns criteris i es genera una nova població
- **Crossover:** S'encreuen els gens de dos individus.
- **Mutació:** Algun individu pot canviar aleatòriament algun dels seus gens.
- **Avaluació:** Cada individu de la població s'analitza segons la funció a optimitzar.

3.3.2 Representació

Com s'ha dit, cada individu conté una possible solució de l'espai de cerca, però el problema és quina forma té aquest individu per a que contingui tota la informació

necessaria. Hi ha dues formes per representar-los:

- Representació binària
- Representació real

Representació binària

Tradicionalment s'utilitza la representació binària que consisteix en que cada individu és representat mitjançant 0's i 1's i tots són codificat amb el mateix nombre de bits. Per dimensionar el nombre de bits que es necessiten per ser capaços de representar tot el domini, s'haurà de determinar la precisió que és vol en les solucions. Per exemple, si es necessités representar un rang de valors de l'1 al 6 amb una precisió de dos decimals es requeriria una codificació que permetés $(limit\_superior - limit\_superior) * 10^{\#decimals}$, és a dir 500 combinacions diferents, per això en necessitarien 9 bits:

$$256 = 2^8 < 500 < 2^9 = 512$$

Per saber el valor d'un individu, simplement es necessita convertir a decimal el seu valor com a cadena binària i fer-li la següent transformació:

$$x = limit\_inferior + x' * \frac{limit\_superior - limit\_inferior}{2^{bits}-1}$$

on x serà el valor de l'individu, límit_inferior, serà el valor més petit del rang que es té, límit_superior serà el valors més gran del domini, x' serà el valor en decimal de la cadena binària i bits serà el nombre de bits que té la cadena binària. Així si es tingués la cadena '101010101' representaria el valor 4.33 ja que:

$$x = 1 + 341 * \frac{5}{511} = 4.33$$

És evident llavors que els individus '000000000' i '111111111' representaran els límits del domini a representar, que en aquest cas serien 1 i 6.

No obstant aquesta representació dificulta el treball a mesura que es complica el domini a representar. Per una banda com més precisió es necessiti, més bits es necessitaran a la cadena, de forma que el cost serà directament proporcional al nombre d'aquests; aquesta representació, també, complica molt el tractament de representacions de dominis multidimensionals ja que s'haurà de fer un algorisme de conversió capaç d'extreure tots els valors codificats en una cadena, a més de l'increment de bits que això implica.

Representació real

La representació real codifica cada individu amb un vector de reals, i tots els cromosomes tindran aquest vector de la mateixa longitud. Cada gen d'un individu tindrà el rang especificat i el conjunt de tots serà el que formarà una solució. En aquest cas la precisió de les solucions es basarà en la quantitat de bits que l'arquitectura del computador permeti representar un real, tot i així s'aconsegueix molta més precisió que amb la representació binària.

Actualment aquest tipus de codificació és el que s'utilitza perquè no necessita cap algorisme de conversió del valor de un gen al valor real i és capaç de cobrir dominis molt més grans (de forma que es té molta més precisió com s'acaba de dir). Amb aquesta codificació també és molt més simple treballar amb dominis multidimensionals, ja que per cada dimensió nova que es necessiti simplement s'haurà de afegir un altre real al vector que formen l'individu.

3.3.3 Inicialització

Principalment totes les possibles solucions es generen aleatòriament per formar la població inicial, òbviament aquestes han de formar part del rang i seguir la representació que s'ha especificat per a cada individu. La mida de la població ve especificada per l'usuari però ha de ser el suficientment gran per a que el resultat final pugui arribar a ser precís. A pesar de que com s'acaba de dir, la població es genera aleatòriament cobrint tot l'espai de cerca, a vegades es pot forçar la població inicial per a que les possibles solucions formin part de l'àrea on es troben les solucions més optimes.

3.3.4 Selecció

La selecció [17] és la forma que té l'algorisme genètic de triar quins individus formaran part de la nova població. Hi han diverses formes de fer la selecció:

- **Ruleta:** Mètode de selecció aleatori on els més forts tenen més probabilitats de ser seleccionats.
- **Torneig:** Mètode de selecció on diversos individus competeixen per sobreviure.

Ruleta

Com el seu nom indica, simula una tirada d'una ruleta de forma que la probabilitat de que li toqui a un individu formar part de la següent població és directament proporcional a la seva avaluació, és a dir, com millor sigui un individu, més probabilitat

té de sortir elegit. És el mètode més estocàstic. Aquest mètode de selecció conta de quatre passos:

- Avaluar cadascun dels individus de la població actual.
- Trobar el valor total de avaluació:

$$F = \sum_{i=1}^{poblacio} eval(v_i)$$

- Trobar la probabilitat de selecció de cadascun dels individus:

$$p_i = eval(v_i)/F$$

- Calcular la distribució de cada individu:

$$q_i = \sum_{j=1}^{i} p_j$$

Per exemple, la taula 3.1 mostra un conjunt d'individus amb el retorn de la seva avaluació, la probabilitat de ser seleccionats que tenen i la seva distribució. També mostra el total de la suma dels valors de avaluació per a que es pugui veure més endavant com el total tendeix a créixer.

Cromosoma	**Avaluació**	**Probabilitat**	**Distribució**
A	1	0.0625	0.0625
B	3	0.1875	0.25
C	4	0.25	0.5
D	8	0.5	1
Total	**16**	**1**	**-**

Taula 3.1: Exemple de població

Un cop fet això s'ha de simular la tirada de la ruleta tantes vegades com individus hi hagi a la població. Cada vegada seleccionarem un cromosoma per a la nova població d'aquesta manera:

- Generar un numero aleatori entre 0 i 1.
- Si $r < q_1$ aleshores seleccionem el primer cromosoma, sinó hem de seleccionar el primer cromosoma que trobem que compleixi $r < q_i$.

Seguint amb l'exemple anterior si ara en simulen quatre tirades aleatòries generant quatre nombres aleatoris entre 0 i 1 com podrien ser el 0.555, el 0.0027, el 0.632 i el 0.3241; com es mostra a la taula 3.2 quedarà l'individu D, l'A, un altre cop el D i el C, d'acord amb els passos que s'acaben d'explicar.

Cromosoma	Avaluació	Probabilitat	Distribució
D	8	0.38	0.38
A	1	0.05	0.43
D	8	0.38	0.81
C	4	0.19	1
Total	**21**	**1**	**-**

Taula 3.2: Exemple de població després de la selecció per ruleta

Cal dir que d'aquesta manera un cromosoma pot ser seleccionat més d'una vegada, i els millors individus són els que surten més cops, mentre que els pitjors tendeixen a desaparèixer.

Torneig

En el mètode de selecció per torneig no es deixen les seleccions tan a l'atzar, ja que no es perd el millor individu pel camí. En aquest cas, s'elegeixen grups de cromosomes a l'atzar i entre ells el més fort substitueix als altres triat en la nova població. Per exemple, si tinguéssim el mateix exemple d'abans, i féssim grups de dos com ara l'individu A i l'individu D, i per altra banda, l'individu B amb l'individu C, llavors l'A guanyaria el D i el C al B, de manera que la població final seria:

Cromosoma	Avaluació
D	8
C	4
C	4
D	8
Total	**24**

Taula 3.3: Exemple de població després de la selecció per torneig

És una prova empírica que amb la selecció el valor d'avaluació total tendeix a anar creixent.

3.3.5 Encreuament

L'encreuament és un operador que selecciona i combina dos cromosomes als que s'anomenen pares i genera dos descendents, als que s'anomenen fills. La idea de l'encreuament és que el nou individu, el fill, pot tenir unes característiques millors que els dos pares si els dos pares li donessin el millor de cadascun. El paràmetre de probabilitat d'encreuament va donat per l'usuari.

En aquest cas hi ha diverses maneres de fer un encreuament, i casi totes serveixen tant per individus amb gens binàris con per a individus amb gens reals; encara que

es veurà que hi ha alguns casos que només serveixen per als reals. El que s'haurà de tenir en compte és que el punt d'encreuament, que en l'encreuament binàri podrà ser a qualsevol gen, mentre que l'encreuament real serà entre valors reals. Aquestes maneres de fer-ho són les següents:

- **Per un punt:** En aquesta operació es selecciona aleatòriament un punt d'encreuament en un rang del número de gens d'un individu. Un cop fet això, la primera part del primer pare s'ajuntarà amb la segona part del segon per crear un fill, així com la primera part del segon pare la segona part del primer per crear l'altre fill.

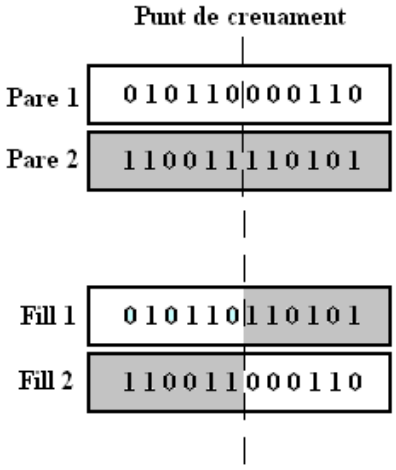

Figura 3.2: Encreuament amb un punt de tall

- **Per diversos punts:** Aquest operador selecciona aleatòriament diversos punts de encreuament en un rang del nombre de gens d'un individu. Un cop es fa això aleshores els pares s'intercanvien les parts per crear els fills nous fills nous.

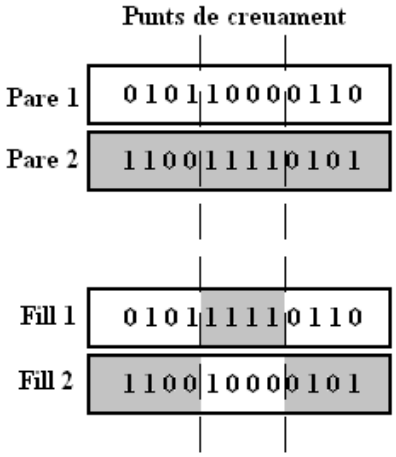

Figura 3.3: Encreuament amb dos punts de tall

- **Uniforme:** [**24**] Aquesta operació baixa a nivell de gen per fer l'encreuament. Primer es defineix la probabilitat d'un gen a ser encreuat i després es mira gen a gen, si s'encreuarà o no. Així veiem que aquest cas ja no segueix el sistema d'encreuament a nivell de segments com els casos anteriors.

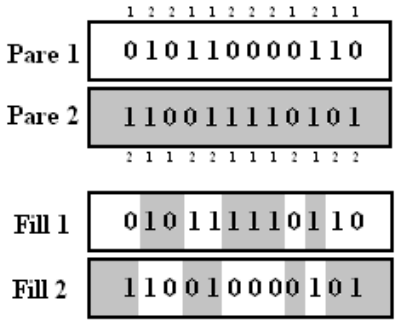

Figura 3.4: Encreuament uniforme

- **Aritmètic:** Aquest operador només serveix per a casos on els individus siguin reals. Es fa una combinació lineal entre els dos pares per generar els dos fill seguint la següent equació:

$$fill_1 = a * pare_1 + (1 - a) * pare_2$$
$$fill_2 = (a - 1)pare_1 + a * pare_2$$

On a és un factor aleatori que marca els pesos dels individus i que es decideix abans de l'operació d'encreuament.

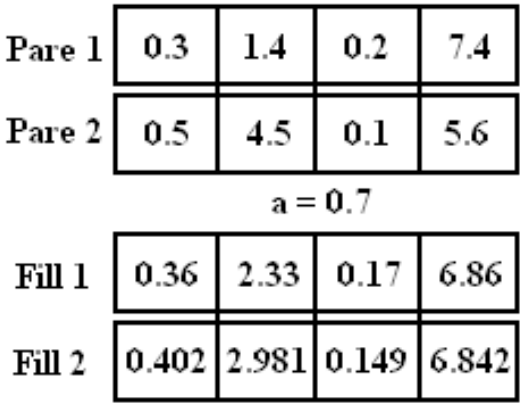

Figura 3.5: Encreuament aritmètic

- **Heurístic:** En aquest cas la operació d'encreuament utilitza els valors dels individus avaluats de dos pares per determinar com s'han d'encreuar. Els fills es creen segons les següents equacions:

$$fill_1 = millorpare + r * (millorpare - pitjorpare)$$
$$fill_2 = millorpare$$

on r es un número aleatori entre 0 i 1. Es probable que el fill_1 no sigui factible, això pot venir donat si la r es seleccionada de forma que diversos gens es surtin del seu rang. Per aquesta raó, s'hauria d'afegir a aquesta manera de fer l'encreuament una manera de repetir l'encreuament si el resultat no es factible, ja sigui repetint la generació de la r o fent que el primer fill sigui el pitjor fill.

3.3.6 Mutació

La mutació és un operador genètic que altera un o més gens en l'individu inicial. Això fa que després d'aquesta operació hi hagi un gen nou, que pugui canviar completament el valor de l'individu. El que es pretén fer amb això és que alguns individus canviïn aleatòriament, per evitar l'estancament en un possible màxim relatiu creant així un valor que a vegades pot ser superior i es pugui trobar una solució més òptima. Encara que el paràmetre de que succeeixi una mutació és definit per l'usuari, normalment es posa un valor baix, ja que un paràmetre alt en mutació fa que l'algorisme genètic es converteixi en una cerca aleatòria.

La mutació es fa de maneres diferents depenent si els individus són binaris o són reals.

Mutació binària

La mutació binària consisteix en simplement invertir el valor d'un gen, és a dir, si el gen val 1 passa a valer 0 i si val 0 passa a valer 1. La figura 3.6 mostra com quedaria un gen després de mutar.

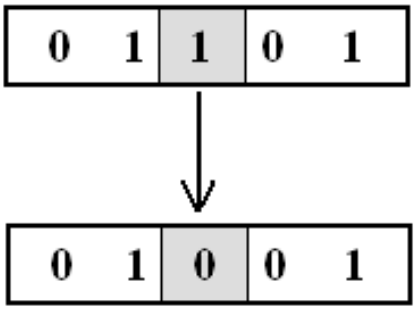

Figura 3.6: Mutació binària

Mutació real

En aquest cas ens trobaríem amb diverses formes de fer la mutació [25]:

- **Uniforme:** Donat un cromosoma, un gen mutarà de forma que rebrà un valor nou aleatori entre el límit superior i el límit inferior que pugui valer aquell gen.
- **No uniforme:** **[16]** Donat el gen d'un cromosoma, aquest tindrà més probabilitats de mutar quan més baixa sigui la generació on s'està. Amb això s'aconsegueix que s'explori de forma més global l'espai de cerca a l'inici i de forma més local al final.
- **Gaussiana:** L'operació de mutació gaussiana afegeix o resta al gen que tenim una unitat en relació amb la distribució normal. Si el gen se surt dels limits amb aquesta operació es torna al seu valor original.
- **Basada en paràmetres:** Donat el gen que ha de mutar es calcula un nombre $u \in [0, 1]$ i es calcula la quantitat que s'afegirà o restarà al gen a mutar de la següent forma [6]:

$$\vec{\delta} = \begin{cases} (2u)^{\frac{1}{n_m+1}} & \text{si } u \leq 0.5 \\ 1 - [2(1-u)]^{\frac{1}{n_m+1}} & \text{altrament} \end{cases}$$

- **Límit:** El gen d'un cromosoma seleccionat per a mutar passa a ser el màxim o el mínim del rang d'aquest gen. Si es el màxim o el mínim es decideix aleatòriament.

3.3.7 Elitisme

L'elitisme és un mètode utilitzat per assegurar-se que el o els millors individus de la població no es perden. Aquest consisteix en realitzar l'etapa de selecció en dues parts:

- Es seleccionen directament n cromosomes amb millor avaluació per formar part de la població final
- Es fa la selecció seguint els criteris explicats anteriorment per als individus restants de la població

La introducció d'elitisme en un algorisme genètic augmenta la velocitat de cerca de un punt òptim sempre i quan la funció d'avaluació no tingui punts òptims relatius. Si aquesta funció presenta màxims i mínims relatius, aleshores la efectivitat de l'algorisme empitjora més com més n'hi hagi.

Amb poblacions d'una mida petita es poden aconseguir efectes similars a l'elitisme fent reinicialitzacions periòdiques en l'algorisme genètic cada cop que aquest convergeixi a una solució. Cada cop que arribi aquesta situació llavors s'haurà de salvar els millors individus que formaran part de la nova població mentre que la resta s'haurà de reinicialitzar com si fos el principi de l'algorisme. Aquest mètode pot ser beneficiós, ja que al reinicialitzar la població afegeixes més diversitat, requisit indispensable per a que funcioni l'algorisme amb poblacions petites.

3.4 Xarxes Neuronals

El cervell humà és un dels principals èxits de l'evolució biològica, ja que és un gran processador que se n'encarrega d'infinitat d'informació procedent de tots els sentits que rep constantment a gran velocitat. Aquesta informació es combina i es compara amb d'altres ja emmagatzemades i se'n donen respostes. A més, També s'aprèn d'aquestes dades i es milloren les habilitats sense tenir instruccions prèvies.

Tanta és la potència del cervell humà que científics i enginyers porten treballant molt de temps per intentar aconseguir sistemes que s'aproximin cada vegada més a aquests. Un dels sistemes que imiten el funcionament d'un cervell són les xarxes neuronals, que a pesar de que aquests no son més que aproximacions molt llunyanes de les neurones biològiques, són molt interessants degut a la seva capacitat d'aprendre i d'associar patrons semblants, amb el que se'ns permet afrontar problemes de difícil solució per a la programació tradicional.

De fet, les xarxes neuronals artificials [18] són mètodes d'aproximació de funcions, ja siguin de valors continus com discrets, que s'utilitzen per a classificar en classes un conjunt donat de dades d'entrada.

3.4.1 El perceptró

Concepte de perceptró

Un perceptró és la mínima expressió de xarxa neuronal i va ser inventat l'any 1957 per Rosenblatt [21]. A partir d'un perceptró es poden configurar moltes topologies de xarxa neuronal.

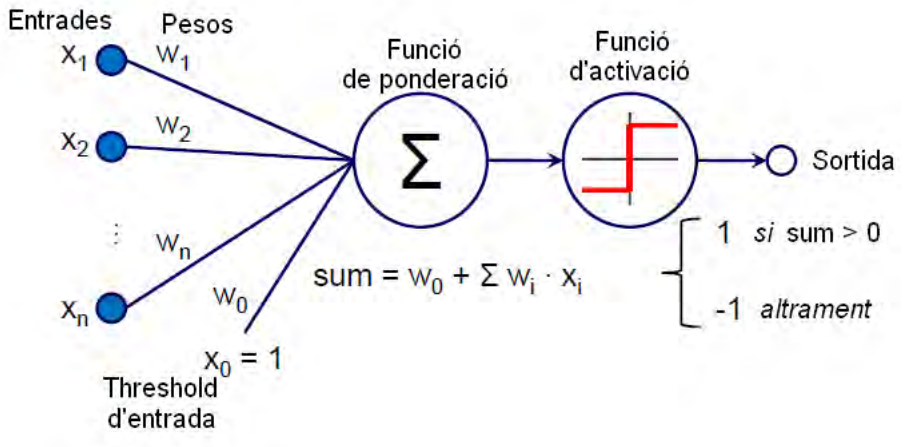

Figura 3.7: Esquema d'un perceptró simple [19]

Com es pot veure en la figura 3.7, un perceptró està format [4] per:

- **Entrades:** Són tots aquells valors que arriben a la neurona i que es computaran amb l'objectiu de donar una

- **Pesos:** Son valors entre -1 i 1 que serveixen per a ponderar les entrades i que se'ls doni més o menys pes per a obtenir el millor resultat.

- **Llindar (threshold) d'entrada:** A més de les entrades que rep la neurona, s'afegeix una entrada x_0 que sempre val 1. El que s'obté amb aquesta entrada és un *offset* a la funció lineal que genera la neurona.

- **Funció de ponderació:** És un sumatori de les entrades del perceptró ponderades amb els seus pesos.

- **Funció d'activació:** És una funció que donat el valor de sortida de la funció de ponderació dona la sortida del perceptró. Aquesta pot ser binaria, de forma que donarà un 1 si la funció de ponderació és més gran que 0 i -1 altrament; o pot ser també real donada una funció sigmoide o lineal a partir de la sortida de la funció de ponderació.

El perceptró agafa un conjunt d'entrades de valor real i en calcula una combinació lineal, aleshores si el resultat és major que el llindar definit, aquesta donarà com a sortida un 1 i altrament un -1. De fet donat un conjunt d'entrades x_1 fins x_n la sortida es computa de la següent forma:

$$o(x_1, ..., x_n) = \begin{cases} 1 & \text{si } w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n > 0 \\ -1 & \text{altrament} \end{cases}$$

On cada w_i és un valor real constant, o pes, que determina la contribució de l'entrada x_i a la sortida del perceptró. Cal veure que hi ha el pes w_0 que no pondera cap entrada; això és degut a que $-w_0$ és el llindar que $w_1x_1 + w_2x_2 + ... + w_nx_n$ haurà de superar per a que la sortida del perceptró sigui 1. De fet, cal imaginar que existeix un x_0 de forma que es pot expressar la inequació com:

$$\sum_{i=0}^{n} w_ix_i > 0$$

No obstant, el valor dels pesos d'un perceptró s'han d'aprendre d'alguna forma. Per a aconseguir-ho hi ha diversos mètodes d'aconseguir-ho.

Mètodes d'entrenament

Regla de l'entrenament del Perceptró (Perceptron Training Rule)

Una forma d'entrenar un perceptró és començant inicialitzant a l'atzar els seus pesos a l'atzar i iterativament anar comprovant el resultat obtingut amb l'exemple d'entrenament i modificar els pesos sempre que hi ha un error. Això es repeteix fins que s'aconsegueix no tenir cap error de classificació.

Cal tenir en compte que els pesos canviaran seguint la *Perceptron Training Rule* que revisa cada pes w_i associat a cada entrada x_0 tenint en compte que:

$$w_i \leftarrow w_i + \Delta w_i$$

On:

$$\Delta w_i = \eta(t - o)x_i$$

En aquesta formula t és la sortida ideal de l'exemple d'entrenament, o és la sortida generada pel perceptró i η és una constant petita (per exemple 0.05) que és el ritme d'entrenament que modera la velocitat amb el que els pesos canvien.

Gradient Descent Technique i Delta Rule

A pesar que la *Perceptron Training Rule* pot entrenar molt bé un problema linealment separable, pot cometre l'error de convergir si el problema no és linealment separable. Aquí va aparèixer la *Delta Rule* [26], descrita per Widrow i Hoff l'any 1960, que aproxima la millor solució dins del problema no separable.

La idea d'aquest mètode és utilitzar el *gradient descent technique* per trobar els pesos dins l'espai de cerca de pesos que millor funcionen amb el conjunt d'entrenament. Cal tenir en compte que aquesta forma d'aproximar els pesos és important perquè és la base de l'algoritme de *Backpropagation* que s'explicarà més endavant.

La *Delta Rule* ve donada d'agafar la sortida de un perceptró sense haver avaluat el llindar, és a dir:

$$o(\vec{x}) = \vec{w}\Delta\vec{x}$$

D'aquesta manera es pot calcular l'error quadràtic respecte el conjunt d'entrenament de la forma:

$$E(\vec{w}) = \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

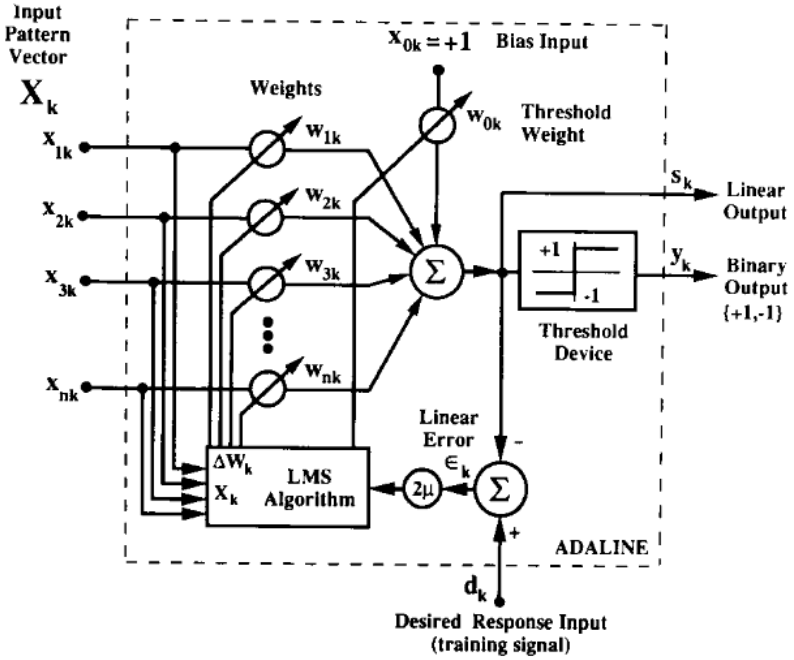

Figura 3.8: Esquema d'un perceptró amb un sistema d'aprenentatge mitjançant l'error trobat a partir dels mínims quadrats (LMS) [27]

On D és el conjunt d'entrenament, t_d és la sortida objectiu per les entrades donades i o_d és la sortida lineal de les entrades donades de l'exemple d. Aquest error s'utilitza per a trobar la nova Δw_i que en aquest cas passa a ser:

$$\Delta w_i = -\frac{\partial E}{\partial w_i} = \eta \sum_{d \in D} (t_d - o_d) x_{id}$$

La figura 3.9 mostra un pseudocodi de l'algoritme *gracient descent* que serveix per a trobar els millors pesos en un problema linealment no separable.

Init each w_i to some random value
while $\neg$ termination condition **do**
  Init each Δw_i to zero
  for each $\langle \vec{x}, t \rangle$ in training_examples **do**
    Input the instance $\vec{x}$ to the unit anc compute the output o
    for each linear unit weight w_i **do**
      $\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$
    end for
  end for
  for each linear unit weight w_i **do**
    $w_i \leftarrow w_i + \Delta w_i$
  end for
end while

Figura 3.9: Pseudocodi de l'algorisme *Gradient-Descent* [18]

3.4.2 Xarxes neuronals multicapa

Concepte de xarxa neuronal multicapa

El problema dels perceptrons simples és que només són capaços de resoldre problemes linialment separables. Per exemple es pot separar una *and* i una *or* amb un perceptró, però el problema és com separar una *xor* amb una sola línia?

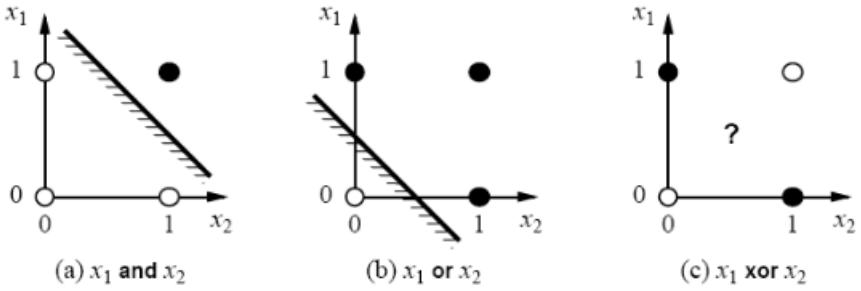

Figura 3.10: Representació de la separació d'una *and*, una *or* i de com no es pot separar una *xor* [19]

La resposta és que no es pot. Per aquest motiu es proposen les xarxes neuronals multicapa. Aquestes són xarxes amb diversos perceptrons que interactuen entre elles i que estan dividides en capes de forma que les sortides d'una capa sempre seran les entrades de la següent capa. Les capes, que estan formades per diversos perceptrons, [4] es poden diferenciar en tres tipus:

- **Capa d'entrada:** Només tenen una entrada que correspon amb una entrada de la xarxa neuronal. Les neurones d'aquesta capa solen donar com a sortida la pròpia entrada.
- **Capa oculta:** Aquesta capa rep les sortides de la capa anterior i cada neurona les processa com a un perceptró simple. El conjunt de sortides de les neurones d'aquesta capa són les entrades de la següent. En una xarxa pot no haver cap capa oculta, haver-n'hi una o més.
- **Capa de sortida:** Aquestes capes treballen igual que les capes ocultes amb la diferència que enlloc d'enviar la sortida a la següent capa, la sortida d'aquesta capa és la sortida de la xarxa neuronal.

La figura 3.11 mostra la representació gràfica d'una xarxa multicapa.

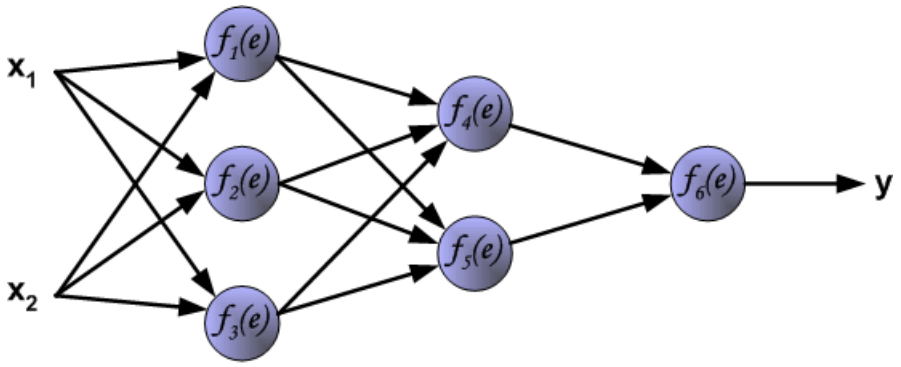

Figura 3.11: Esquema d'una xarxa neuronal multicapa [19]

De fet, per a resoldre la *xor*, el que s'aconseguiria amb una xarxa multicapa formada amb una capa oculta de dos perceptrons seria un resultat que aconseguiria separar correctament els punts com es mostra en la figura 3.12.

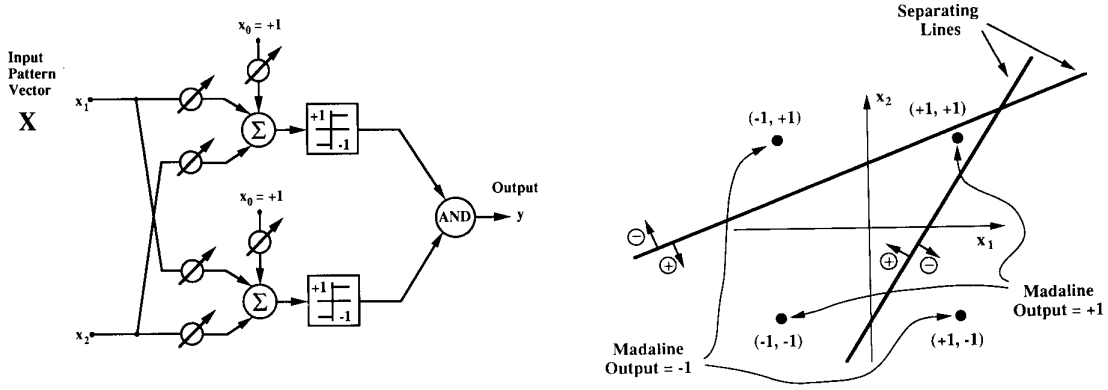

Figura 3.12: Xarxa neuronal proposada per Widrow per a resoldre el problema de la *xor* amb la representació de la solució donada [27]

Les xarxes neuronals multicapa també necessites un entrenament per a aconseguir els pesos correctes. Per tal d'aconseguir-ho tenim diversos mètodes.

Mètodes d'entrenament

Backpropagation

L'algoritmne de *Backpropagation* aprèn els pesos d'una xarxa neuronal multicapa, donada una xarxa amb un nombre fixat de neurones i de connexions, Utilitza l'algoritme de *gradient descent* explicat anteriorment per minimitzar l'error quadràtic entre la sortida de la xarxa i la sortida teòrica.

Com que en aquest cas es tenen xarxes que poden tenir més d'una sortida, aleshores l'error haurà de tenir en compte totes aquestes sortides. Aquest error s'ha de propagar per totes les neurones de les capes anteriors de forma que tots els pesos s'ajustin. L'error és calcula de la següent forma:

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

On *outputs* és el conjunt de sortides de la xarxa, i t_{kd} i i_{kd} són les sortides teòriques i reals de la xarxa de cada sortides k de l'exemple d.

La figura 3.13 mostra el pseudocodi de l'algoritme de *Backpropagation*.

Init all networks weights to small random numbers $(-0.05, 0.05)$
while $\neg$ termination condition **do**
  for each $\langle \vec{x}, t \rangle$ in training_examples **do**
    Input the instance $\vec{x}$ to the network and comput the output o_u of every unit u in the network
    for each neuron output unit k, calculate its error term δ_k **do**
      $\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$
    end for
    for each hidden unit h, calculate its error term δ_h **do**
      $\delta_h \leftarrow o_h(1 - o_h) \sum w_{kh} \delta_h$
    end for
    Update each network weights w_{ji}
    $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$
    where
    $w_{ji} \leftarrow \eta \delta_j x j i$
  end for
end while

Figura 3.13: Pseudocodi de l'algorisme *Backpropagation* [18]

Mètodes d'optimització globals

Una alternativa al *gradient descent* o al *backpropagation* són els algoritmes d'optimització globals per a trobar els pesos.

Es pot utilitzar un algoritme genètic per a evolucionar els pesos correctament, d'aquesta manera es podria codificar dins d'un cromosoma els pesos i potser la topologia de la xarxa i utilitzar les tècniques necessàries per a trobar la solució òptima, a més la funció de *fitness* a utilitzar podria ser la funció de l'error quadràtic.

Capítol 4

TORCS

Aquest capítol descriu com funciona el software de la competició *Simulated Car Racing Competition 2011* [15] i explica com configurar correctament el servidor i el client de forma que es pugui competir.

El simulador TORCS (The Open Racing Car Simulator) és un simulador GPL de cotxes de competició dissenyat per a que els desenvolupadors puguin fer córrer els seus propis conductors programats. Aquest està considerat un dels millors simuladors de software lliure pel seu realisme en el maneig dels vehicles.

El principal motiu pel quan s'ha triat aquest simulador és que TORCS s'utilitza en un ampli ventall de congressos sobre inteligència artificial. Va començar l'any 2008 al WCCI (*World Congress on Computational Intelligence*), i va continuar, apareguent en el CIG (*Computer Intelligence in Games*, CEC (*Congress on Evolutionary Computation* i en el GECCO (*Genetic and Evolutionary Computation Conference*) fins actualment, l'edició del 2011.

Per simplificar el desenvolupament de *bots*, l'organització de les competicions proporcionen una API que serveix per a que es pugui programar en el sistema operatiu que es vulgui i triar entre C++ i JAVA com a llenguatge de programació. També s'ha modificat l'estructura, muntant així una arquitectura client-servidor, com es mostra a la figura 4.1, de forma que el controlador quedi separat del simulador, així al programador no li caldrà entendre com funciona TORCS en sí.

El software de la competició consta de dos mòduls: el client i el servidor. El servidor es un mòdul que s'acobla a TORCS i fa possible la comunicació amb el client, també implementa tots els sensors que donen informació sobre la cursa, així com les accions a realitzar amb el controlador. Per altra banda el client tindrà implementat tota la part heurística de la conducció. Així doncs, cal remarcar que mentre el servidor no estarà dotat de intel·ligència, el client sí.

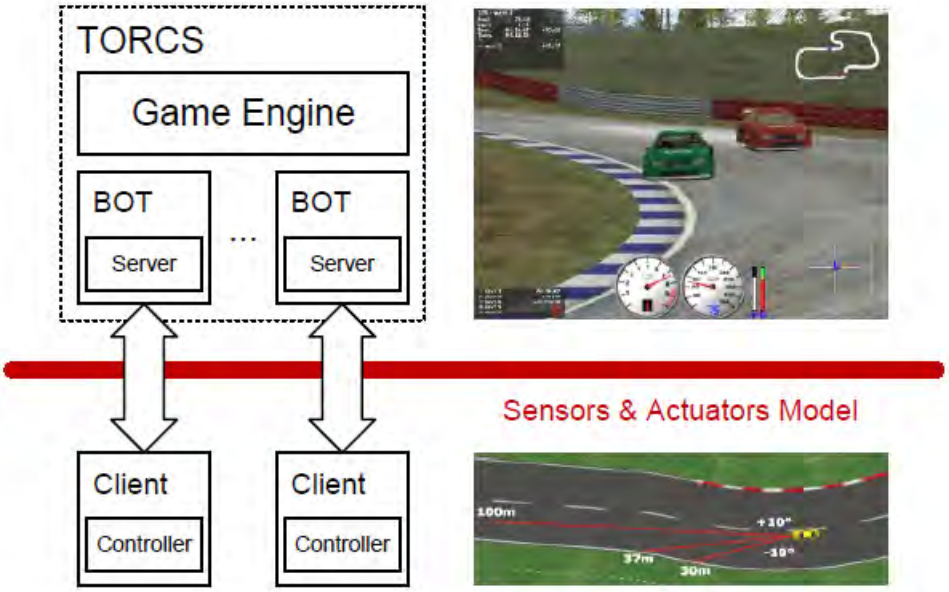

Figura 4.1: Esquema de l'arquitectura del simulador amb el client.[15]

4.1 Execució i configuració del servidor

Un cop s'hagi instal·lat el simulador i s'ha instal·lat el servidor, aleshores ja es poden fer córrer *bots*. Quan es vol fer-lo córrer s'ha d'iniciar TORCS i començar una cursa, aleshores s'ha d'executar el client i es veurà com el cotxe comença a córrer pel circuit. En el simulador TORCS hi ha diversos modes de cursa, no obstant el client només podrà córrer en dos d'ells:

- **Pràctica:** Mode que només permet un *bot* alhora a la cursa.
- **Cursa ràpida:** Mode que permet multiples *bots* alhora a la cursa.

Però abans de començar una cursa s'hauran de configurar els següents paràmetres:

- S'ha de triar el circuit on es vol córrer.
- S'ha d'afegir el *bot* **championship2011serverX** com a competidor. De fet la X indica l'identificador del participant de forma que hi pot haver diversos *bots* alhora.
- S'ha de definir quantes voltes o quants quilòmetres durarà la cursa.
- S'a de triar si en correrà en mode 'Normal' o en mode 'Visualització de resultats'

La forma més ràpida i senzilla de configurar aquests paràmetres és fent-ho directament des de la finestra del simulador, a la finestra de configuració s'hi accedeix des del menú principal seguint la següent ruta:

Race → Quick Race o Practice → Configure Race

Un cop s'ha entrat a la opció configuració, la primera finestra dóna la opció de seleccionar el circuit que es vol. Un cop s'ha seleccionat el circuit s'ha de fer clic a acceptar per accedir a la següent finestra. Aquesta permet seleccionar tots els *bots* que participaran a la cursa. La següent finestra fa seleccionar la longitud de la cursa; aquesta és pot dir en quilòmetres o en voltes. Finalment si es vol es pot fer que la velocitat del simulador sigui 20 vegades la velocitat normal seleccionant el mode 'Visualització de resultats', d'aquesta manera només es mostren els resultats i es pot simular una volta en 3 segons.

El simulador també es pot configurar directament des dels fitxers de configuració. Aquests son uns fitxers XML que es guarden a *practice.xml* o *quickrace.xml* depenent del mode de simulació que es vol triar.

Per seleccionar un circuit s'ha de trobar la secció *Tracks* al fitxer XML que ha de contenir una cosa com:

```
<section name="1">
        <attstr name="nom" val="NOM_CIRCUIT" />
        <attstr name="categoria" val="CATEGORIA_CIRCUIT" />
</section>
```

on *NOM_CIRCUIT* s'ha de substituir pel nom del circuit i *CATEGORIA_CIRCUIT* per la categoria. Tota la informació dels circuits es pot trobar a *./torcs/tracks/categoria/circuit*.

Si hem de triar els corredors que participen es farà buscant la secció *Drivers* al fitxer i posant tantes vegades aquest fragment d'XML com corredors es vulguin.

```
<section name="N">
        <attstr name="idx" val="IDX" />
        <attstr name="modul" val="NOM" />
</section>
```

on *N* serà el enèsim corredor que posem, *IDX* es substituirà amb l'identificador del corredor i *NOM* es canviarà pel nom que es vulgui posar al corredor. El llistat dels corredors es pot trobar a *torcs/drivers/*.

Per canviar la longitud de la cursa o triar si volem triar si es vol veure la simulació o només es resultats, s'ha de canviar les següents línees:

```
...
<attnum name="distance" unit="km" val="DIST" />
...
<attnum name="laps" val="VOLTES" />
...
<attnum name="display mode" val="MODE" />
...
```

on *DIST* serà el número de quilòmetres de la cursa o 0 si la cursa esta definida per voltes. Així *VOLTES* serà el número de voltes o 0 si la cursa està definida en quilòmetres. Finalment *MODE* serà *Normal* o *results only*.

Un cop ja s'ha definit tot per córrer s'ha de fer mitjançant la finestra:

Race → Quick Race o Practice → New race

quan s'hagi fet això es veurà que a la finestra apareix un missatge que diu *Initializing Driver wcci2008competition....* Això indica que s'està esperant a que s'executi el client i es connecti amb el servidor. Durant la cursa, aquesta es pot interrompre prement ESC i en el seleccionant 'Abort Race' en el menú; fent-ho d'aquesta manera el servidor desconnectarà al client correctament passant-li la informació correcta, cosa que no passa si es tanca el servidor de qualsevol altra manera.

4.2 El client

En codi del client en JAVA ens ve donat en un paquet anomenat *champ2011client* i dintre d'aquest es troba tot el codi font i els executables. Per a executar-lo amb el nostre controlador implementat, s'ha d'anar al directori *classes* i escriure:

$$javachamp2011client.Clientchamp2011client.TheControllerhost :< ip > port :< p > id :< client.id > maxEpisodes :< me > maxSteps :< ms > verbose :< v > track :< trackname > stage :< s >$$

on *champ2011client.TheController* és la implementació del nostre controlador, $< ip >$ és la adreça IP de la màquina on es troba el servidor (per defecte *localhost*), $< p >$ és el port del servidor (per defecte el 3001), $< client - id >$ és el nom identificador del *bot* (per defecte *championship2011*), $< me >$ és el nombre màxim d'episodis d'aprenentatge (per defecte 1), $< ms >$ és el nombre màxim de iteracions que té cada episodi (per defecte 0), $< v >$ controla si es vol que surti la informació més detallada o no (per defecte *off*), $< trackname >$ és el nom del circuit on es correrà (per defecte *unknonwn* i $< s >$ és un enter que representa la fase actual de la competició (per defecte és *unknown*.

Si el que es vol és compilar el codi s'ha d'anar al directori *src* i escriure:

$$javac - d../classeschamp2011client/ * .java$$

4.3 Interfície dels sensors i de les accions

Com s'ha dit abans, una de les funcions del servidor és implementar tots els sensors i les accions que es transmeten del servidor al client i del client al servidor respectivament.

Els sensors donen informació sobre la carrera, el cotxe i en general sobre l'estat del joc:

- **angle**. *[π, -π] (rad)* Angle entre la direcció del cotxe i la carretera.
- **curLapTime** *[0, -] (s)* Temps actual de la volta que s'està corrent.
- **damage**. *[0, -]* Danys actuals que ha sofert el cotxe.
- **distFromStart**. *[0, -] (m)* Distància entre el cotxe i la línea de sortida seguint la carretera.
- **distRaced**. *[0, -] (m)* Distància total correguda des de l'inici de la carrera.
- **focus**. *[0, 200]* Vector de 5 indicadors que tornen la distancia amb el límit de la carretera o amb el cotxe d'un oponent en un rang de 200 metres. Aquests sensors agafen entre 1 i 5 graus en la direcció que el client especifica en un marge de -$\pi/2$ a $\pi/2$. Si s'activa la opció *noisy* aleshores s'afegeix una desviació al valor de retorn d'aquest sensor.
- **fuel**. *[0, -] (l)* Gasolina restant.
- **gear**. *[-1, 1, 0,..., 7]* Marxa engranada. -1 és la marxa enrere, 0 es el punt mort, i de l'1 al 7 són les velocitats.
- **lastLapTime**. *[0, -] (s)* Temps que s'ha tardat en recórrer la última volta.
- **opponents**. *[0, 200] (m)* Vector de 36 sensors que detecten la distància a la que estan els oponents amb un abast de 200 metres. Cada sensor esta separat de l'altre per 10º, de manera que es cobreix de π a -π. Si s'activa la opció *noisy* aleshores s'afegeix una desviació al valor de retorn d'aquest sensor.
- **racePos**. *[1, 2,...]* Posició en la carrera respecte els altres competidors.
- **rpm**. *[2000,7000] (rpm)* Revolucions per minut del motor del cotxe.
- **speedX**. *[-] (km/h)* Velocitat respecte l'eix longitudinal de la carretera.
- **speedY**. *[-] (km/h)* Velocitat respecte l'eix transversal de la carretera.
- **speedZ**. *[-] (km/h)* Velocitat respecte l'eix Z de la carretera.

- **track**. *[0, 200] (m)* Array de 19 sensors que detecten la distància a a que estan els extrems de la carretera amb un abast de 200 metres. Cada sensor esta separat de l'altre per 10º, de manera que es cobreix de $\pi/2$ a $-\pi/2$. Quan el cotxe està fora de la carretera, aquests valors no son fiables. Si s'activa la opció *noisy* aleshores s'afegeix una desviació al valor de retorn d'aquest sensor.
- **trackPos**. *[-]* Distancia normalitzada entre el cotxe i el centre de la carretera de forma que 0 indicarà el centre de la carretera, -1 l'extrem dret i 1 l'extrem esquerre. Valors més grans d'1 i de -1 volen dir que el cotxe està fora de la carretera.
- **wheelSpinVel**. *[0, -] (rad/s)* Array de 4 sensors que representen la velocitat angular de cada una de les rodes.
- **z**. *[-] (m)* Distancia del centre de gravetat del cotxe amb la superfície del circuit respecte l'eix Z.

Les accions representen les ordres que el client pot donar al cotxe:

- **accel**. *[0, 1]* Pedal de l'accelerador, 0 és no pitjar-lo i 1 és pitjar-lo a fons.
- **brake**. *[0, 1]* Pedal del fre, 0 és no pitjar-lo i 1 és pitjar-lo a fons.
- **clutch**. *[0, 1]* Pedal de l'embrague, 0 és no pitjar-lo i 1 és pitjar-lo a fons..
- **gear**. *[-1, 0, 1,...,7]* Marxa que el cotxe ha d'engranar.
- **steering**. *[-1, 1]* Valor de gir del volant. -1 i 1 indica que es gira el màxim que es pot cap a l'esquerra o dreta respectivament.
- focus *[-90,90]* Direcció del*focus* en graus.
- **meta**. *[0, 1]* 0 indica no fer res, 1 reiniciar la cursa.

Capítol 5

Desenvolupament del projecte

5.1 Overview de l'aplicació

En aquest capítol es vol desenvolupar un sistema d'aprenentatge seguint la metodologia explicada al capítol 2. Per aquest motiu s'ha dividit el projecte en 5 fases ben separades com es pot veure a la figura 5.1.

Figura 5.1: Diagrama del desenvolupament de les fases en les que s'ha dividit el projecte.

D'aquesta forma s'anirà construint les unitats més petites i incorporant-les a bloc més grans que utilitzaran les seves funcionalitats. Com es pot veure en el diagrama de blocs (figura 5.2), el client contindrà la xarxa neuronal que alhora contindrà l'algoritme genètic i el *backpropagation*.

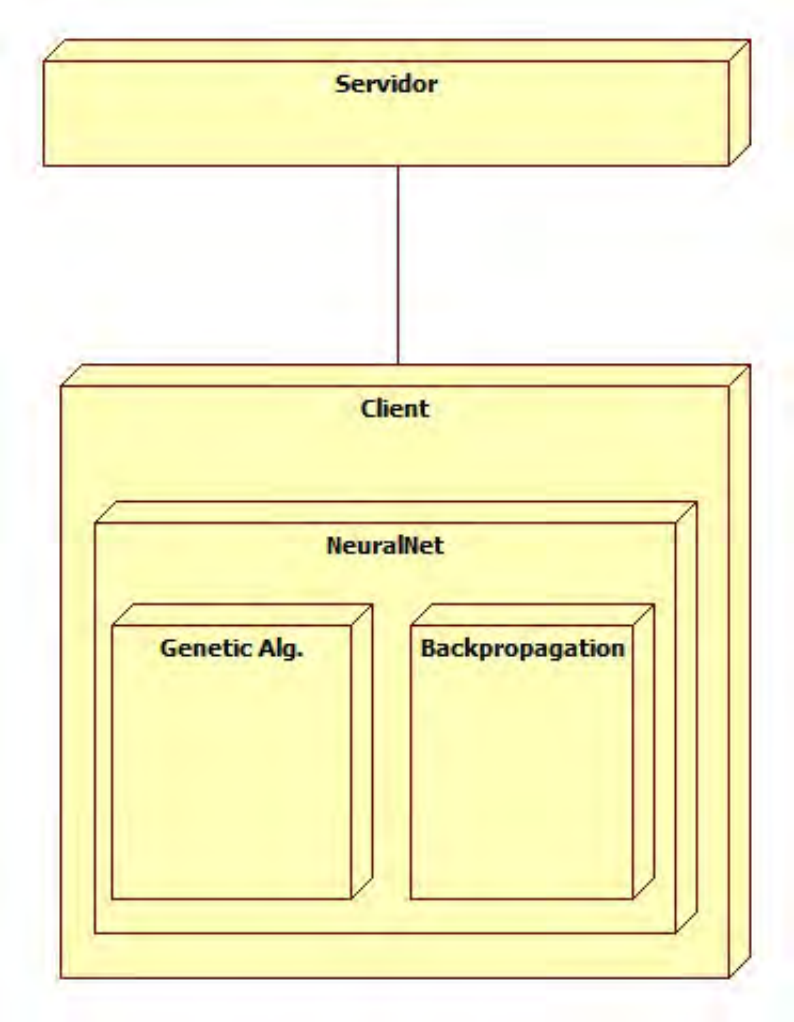

Figura 5.2: Diagrama de blocs del sistema.

5.2 Fase 1: Implementació de l'algoritme genètic

5.2.1 Recollida de requeriments i anàlisi de l'algoritme genètic

En aquesta fase es vol realitzar un algoritme genètic que sigui capaç de optimitzar una funció. Amb l'objectiu d'aconseguir-ho s'han definit un conjunt de requeriments que són els que marcaran les característiques del sistema a realitzar:

- L'algoritme genètic ha de ser adaptable a qualsevol problema.
- L'algoritme ha de ser escalable en quant a dimensió del problema.
- El millor element d'una població ha de persistir, per tant haurà de ser per torneig.
- L'encreuament es farà només per un punt.
- La mutació serà completament aleatòria.

Aquestes especificacions són les que ens guiaran durant el transcurs del disseny i desenvolupament de l'algoritme genètic.

5.2.2 Disseny de l'algoritme genètic

Abans de començar amb el disseny de les classes primer s'ha dissenyat l'algoritme genètic segons els que s'ha explicat en el capítol 3. És a dir que es definirà com es representarà el coneixement, com s'inicialitzarà la població i quin tipus d'operador genètic es farà servir.

Com que el problema consisteix en aconseguir la millor configuració de pesos per una xarxa neuronal, s'ha decidit que s'utilitzarà una representació on els cromosomes són vector de valors reals, on cada valor real correspondrà a cada gen. La mida dels cromosomes serà configurable per a poder adaptar-se a qualsevol problema. La inicialització dels cromosomes serà totalment aleatòria generant un nombre entre -1 i 1 per cada gen, d'aquesta forma ens assegurem que en una població el suficientment gran hi haurà individus repartits per tot el domini del problema.

Els operadors binaris s'ha triat que l'encreuament serà per un punt aleatori; quan dos elements siguin triats per encreuar-se aleshores després s'haurà de triar per quin punt encreuar-los. La mutació serà uniforme on si un gen és elegit per ser mutat aquest se li sumarà o restarà un valor de -0.1 a 0.1 amb l'objectiu d'aconseguir possibles millors resultats però sense que el *fitness* del cromosoma que conté el gen variï notablement.

Un cop especificats els requeriments s'ha realitzat el diagrama de classes que es mostra en la figura 5.3:

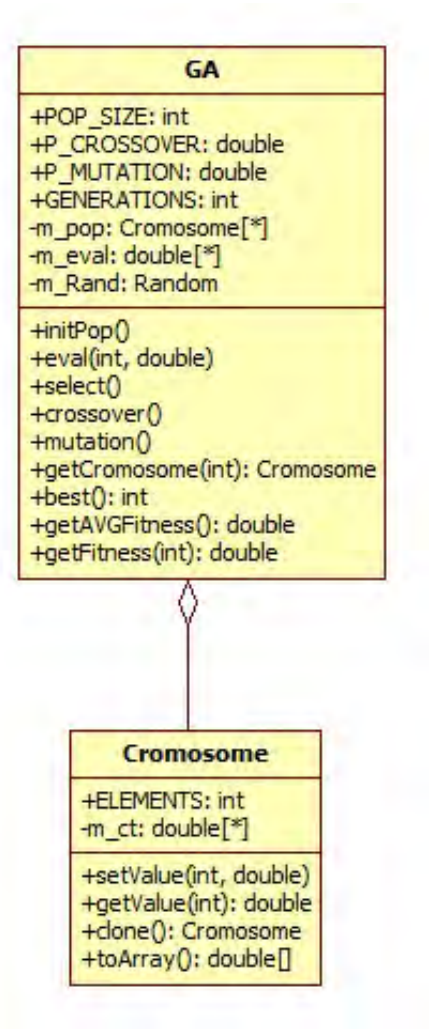

Figura 5.3: Diagrama de classes de l'algoritme genètic.

La classe **Cromosome** encapsula el que són els valors de cada *set* de dades. Conté una constant anomenada **ELEMENTS** que indica la quantitat de dades que conté un cromosoma i les següents funcions:

- **setValue(int,double):** És un mètode que se n'encarrega de posar el valor indicat a la posició passada per paràmetres.
- **getValue(int):** Retorna el valor de la dada a la posició passada per paràmetres del cromosoma.
- **clone():** Retorna un cromosoma idèntic al que fa l'acció.
- **toArray():** Retorna el cromosoma en forma d'array.

La classe **GA** és l'encarregada de configurar i fer les accions de l'algorisme genètic. Els paràmetres de l'experiment que es poden configurar són:

- **POP_SIZE:** És un enter que indica la mida de la població d'individus a avaluar.
- **GENERATIONS:** És un enter que indica el nombre de iteracions que durarà l'experiment.
- **P_CROSSOVER:** És un valor entre 0 i 1 que indica quina és la probabilitat de que un individu sigui seleccionat per a fer l'encreuament.
- **P_MUTATION:** És un valor entre 0 i 1 que indica quina és la probabilitat de que un individu sigui seleccionat per a fer la mutació.

A més, aquesta classe conté els següents mètodes:

- **initPop():** És l'encarregat de generar una població aleatòria quan comença l'experiment.
- **eval(int,double):** Aquest mètode permet assignar un valor com a *fitness* a la posició de l'element avaluat. En aquesta classe no s'implementa cap funció a avaluar, ja que serà l'usuari d'aquesta classe qui se n'encarregui quina serà la funció a maximitzar o minimitzar.
- **select():** Se n'encarrega de fer la selecció de torneig sobre la població existent.
- **crossover():** Se n'encarrega de realitzar l'encreuament sobre alguns elements de la població existent.
- **mutation():** Se n'encarrega de realitzar la mutació sobre alguns elements de la població existent.
- **getCromosome(int):** És un mètode que retorna el cromosoma que es troba a la posició passada per paràmetres.
- **best()**: Retorna la posició de l'element que millor *fitness* té.
- **getAVGFitness():** Retorna la mitjana aritmètica de tots els *fitness* dels elements que es troben a la població existent.
- **getFitness(int):** Retorna el *fitness* de l'element indicat en el paràmetre.

5.2.3 Implementació de l'algoritme genètic

Tal com s'ha explicat anteriorment en el capítol 3 ja que s'ha seguit al peu de la lletra la definició d'un algorisme genètic. Com es mostra en la figura 5.4, l'algorisme genera una població aleatòria a l'inici i l'avalua, en aquest moment i durant el nombre de generacions determinades es seleccionaran, encreuaran, mutaran i avaluaran un altre cop els cromosomes o individus de la població.

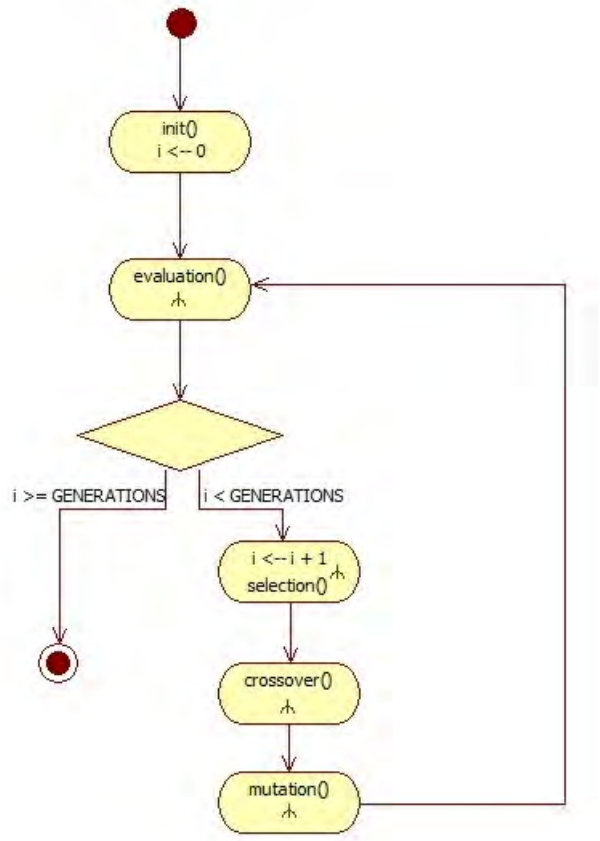

Figura 5.4: Diagrama d'activitats d'un algorisme genètic.

Selecció

La selecció, com s'ha definit als requeriments, es fa per torneig. Aquesta tècnica consisteix en triar grups de cromosomes a l'atzar (en el cas d'aquest projecte els grups són de tres cromosomes) i el que tingui un *fitness* més gran substituirà els altres dos en la població on es troben. La decisió de triar aquest mètode és que així es garanteix la supervivència del més fort i no es permet que un cop de mala sort deixi la població sense el seu millor individu. Es mostra a la figura 5.5.

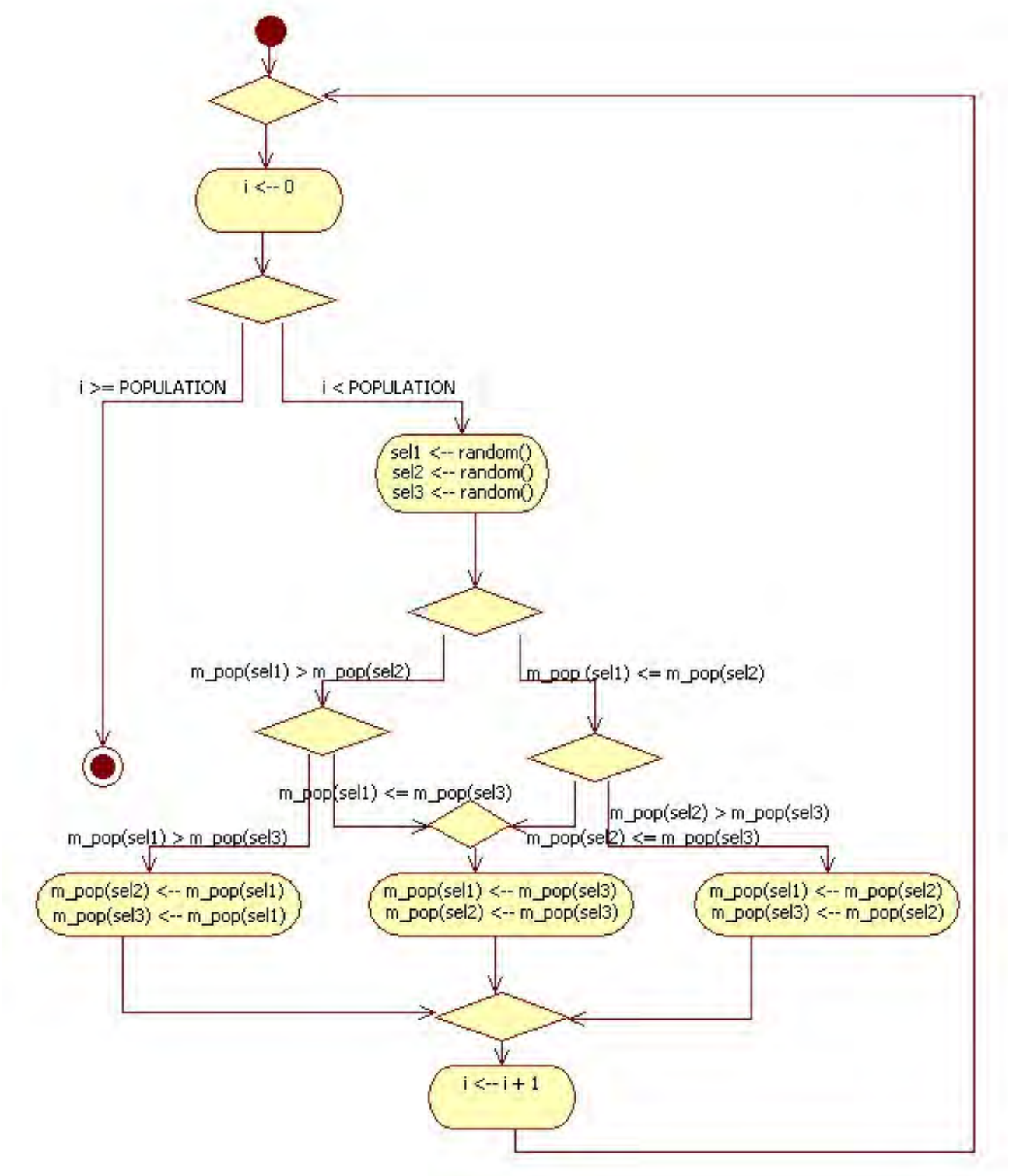

Figura 5.5: Diagrama d'activitats de la selecció per torneig d'un algorisme genètic.

Encreuament

S'ha utilitzat un encreuament dos a dos amb un punt. Com ja s'ha explicat al capítol 3 a l'apartat d'algoritmes genètics i es mostra en el diagrama de la figura 5.6, per a fer l'encreuament s'ha de recórrer tota la població i generarà un nombre aleatori per cada cromosoma; si aquest nombre es menor que la probabilitat d'encreuament definida, aleshores aquest cromosoma queda seleccionat per a l'encreuament. Quan un segon cromosoma és seleccionat, aleshores els dos cromosomes seleccionats es parteixen i intercanvien les seves parts de forma que sortiran dos individus.

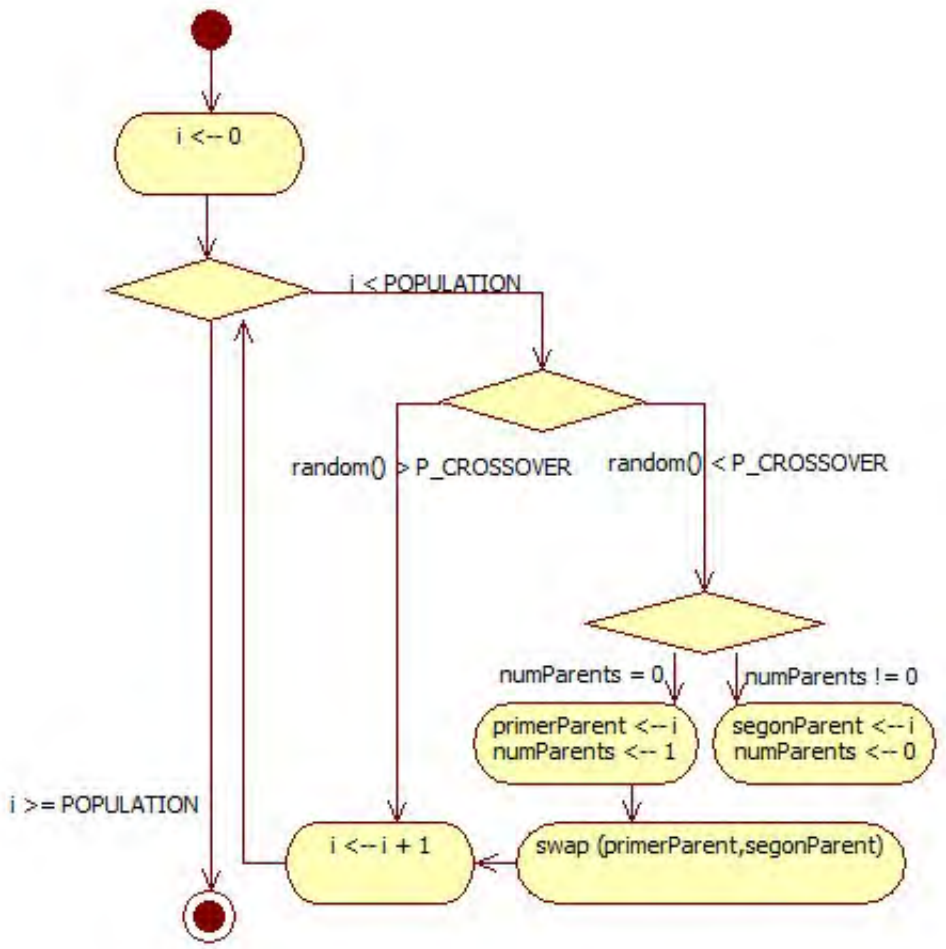

Figura 5.6: Diagrama d'activitats de l'encreuament per un punt d'un algorisme genètic.

Mutació

La mutació, com es mostra en la figura 5.7, fa que un gen pugui canviar de valor aleatòriament. En el cas d'aquest projecte, si un gen es seleccionat per a ser mutat,

aleshores el valor del gen serà incrementat o decrementat en Δ.

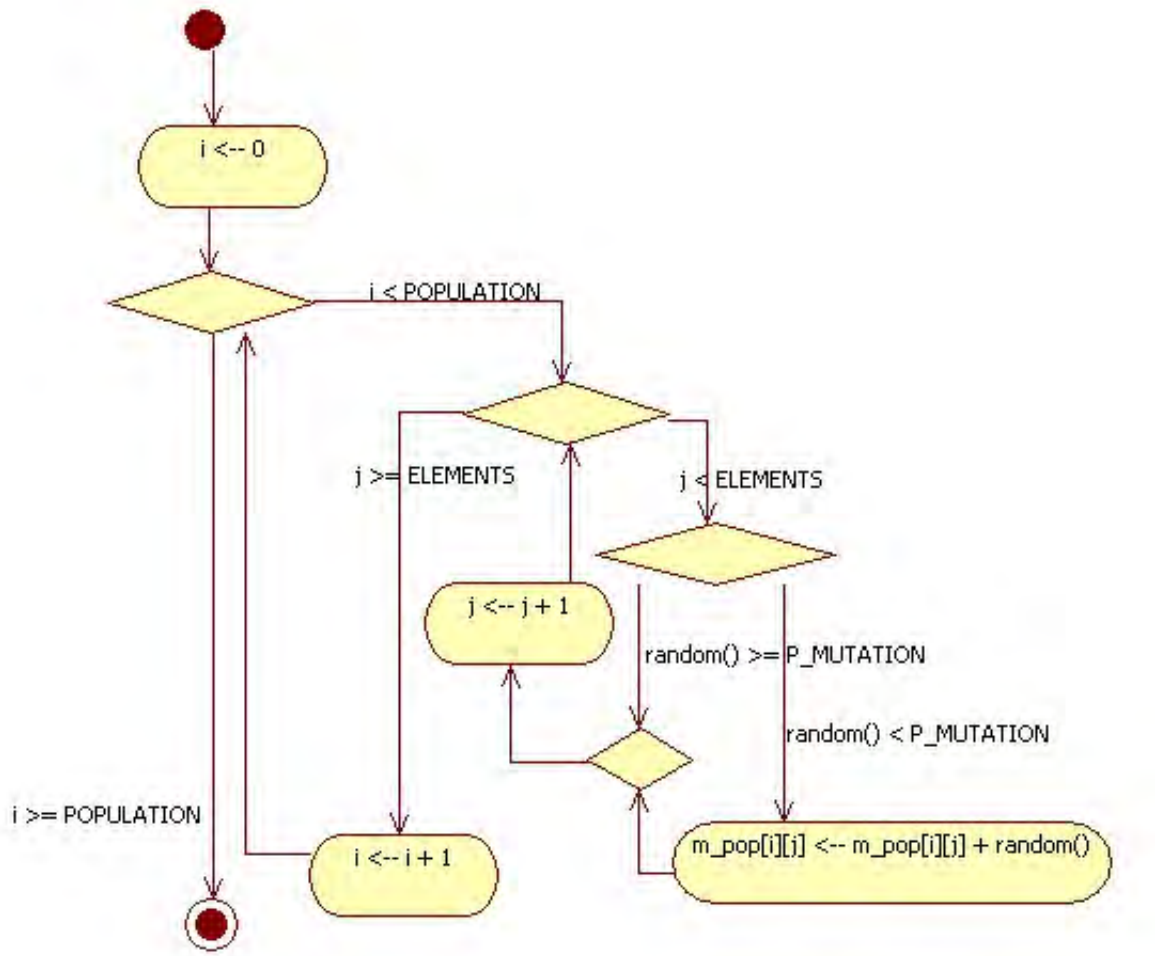

Figura 5.7: Diagrama d'activitats de la mutació en un algoritme genètic.

5.2.4 Proves de l'algoritme genètic

Com que un algoritme genètic és un algoritme de optimització de funcions s'han triat un conjunt de funcions per a avaluar el sistema desenvolupat. Les funcions triades [7][11][12] són funcions típiques de la literatura sobre optimització.

Els test que s'han realitzat a continuació són:

- Sphere Model
- Funció de Rosenbrock
- Funció de Griewngk's

Aquests tests volen desmostrar el bon funcionament de l'algoritme implementat en aquesta fase.

Sphere Model

El test *Sphere Model* és un exemple simple tridimensional de paràbola de contorn esfèric. Degut a la seva simplicitat i simetria, aquesta funció dóna un primer test que serà senzill d'analitzar. La funció és la següent:

$$f(x) = \sum_{i=1}^{3} x_i^2$$

En la taula 5.1 es pot veure la configuració de l'experiment i els resultats obtinguts en aquest problema tenint en compte que s'ha delimitat l'espai a -5.12 $\leq x_i \leq$ 5.12 i que l'objectiu del test és minimitzar la funció.

Test: Sphere Model
Dimensió: 3
Algorisme genètic
Població: 1000
Generacions: 1000
Probabilitat d'encreuament: 0.5
Probabilitat de mutació: 0.2
Resultat obtingut:$(3.2637E-6, 3.05616E-6, -1.90161E-6)$
Resultat ideal: $(0, 0, 0)$

Taula 5.1: Configuració i resultat del test Sphere Model

En la figura 5.8 es mostra l'evolució del *fitness* tenint en l'eix X les generacions i en l'eix Y el *fitness* dels individus.

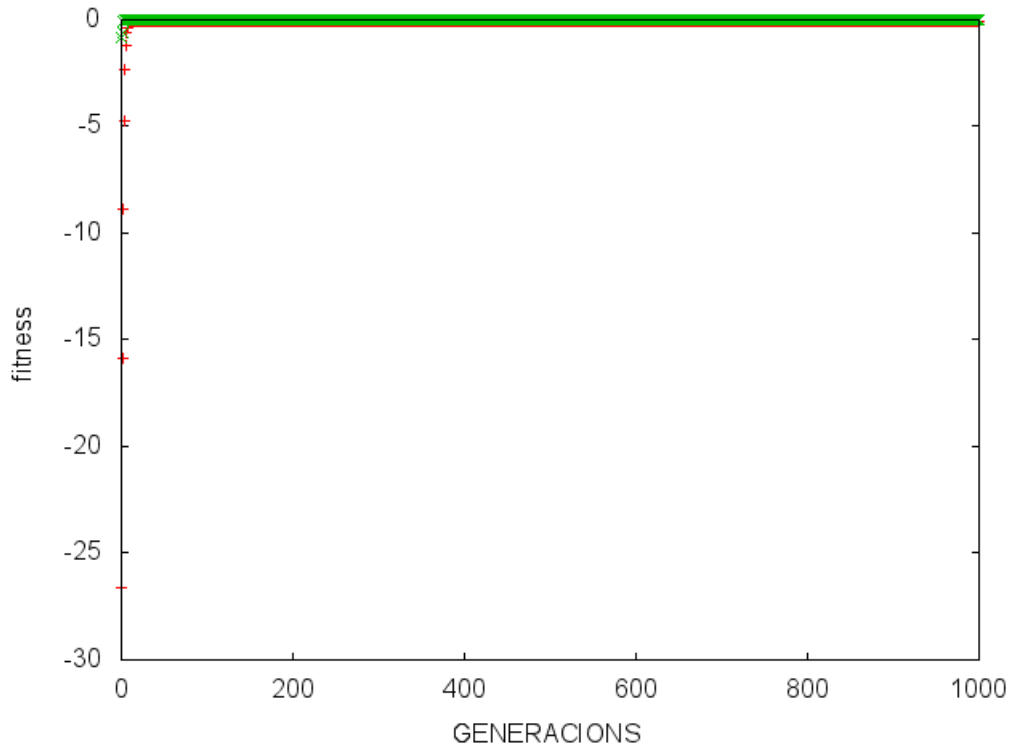

Figura 5.8: Evolucio del fitness durant l'experiment Sphere Model, en vermell es mostra la mitjana del *fitness*, en verd el màxim.

Funció de Rosenbrock

La *funció de Rosenbrock*, definida de la següent forma:

$$f(x) = 100 * (x_1^2 - x_2)^2 + (1 - x_1)^2 \text{ [22]}$$

Aquesta és una funció continua que ens presenta més dificultat que la *Sphere Model* a l'hora de minimitzar-la perquè ja presenta una forma on ens podem trobar mínims o màxims relatius.

A la taula 5.2 es pot veure la configuració de l'experiment i els resultats obtinguts en aquest problema tenint en compte que s'ha delimitat l'espai a -2.048 $\leq x_i \leq$ 2.048 i que l'objectiu del test és minimitzar la funció.

Test: Funció de Rosenbrock
Dimensió: 2
Algorisme genètic
Població: 1000
Generacions: 1000
Probabilitat d'encreuament: 0.5
Probabilitat de mutació: 0.2
Resultat obtingut:$(0.99774, 0.99548)$
Resultat ideal: $(1, 1)$

Taula 5.2: Configuració i resultat del test de la funció de Rosenbrock

En la figura 5.8 es mostra l'evolució del *fitness* tenint en l'eix X les generacions i en l'eix Y el *fitness* dels individus.

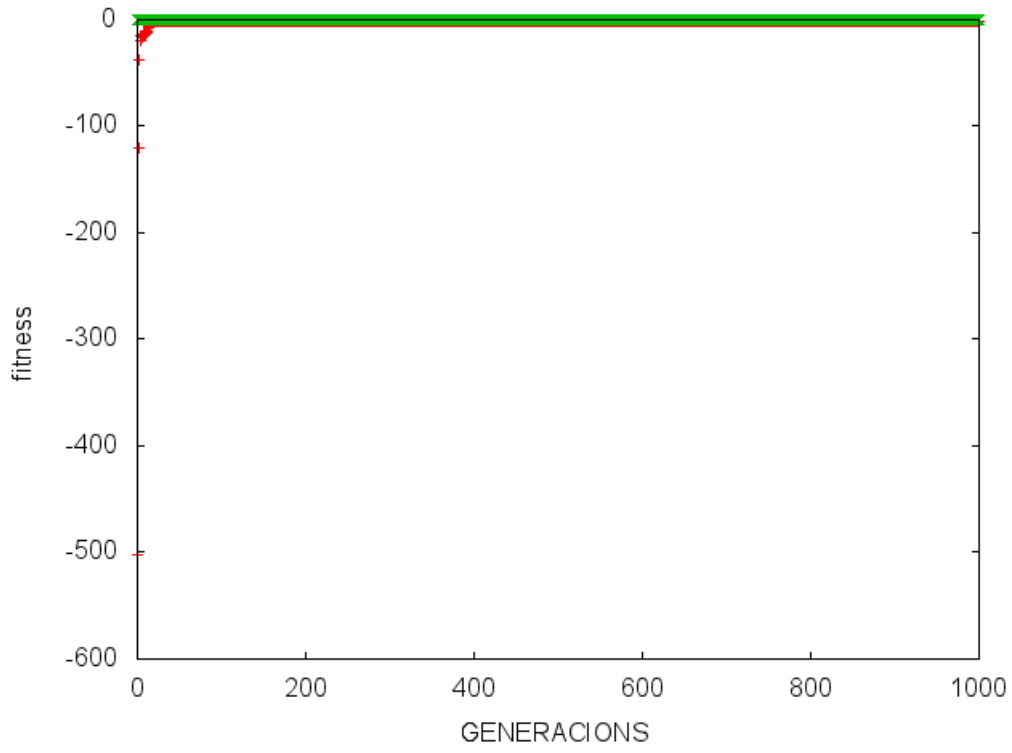

Figura 5.9: Evolucio del fitness durant l'experiment de la funció de Rosenbrock, en vermell es mostra la mitjana del *fitness*, en verd el màxim.

Funció de Griewngk's

La *funció de Griewngk*, que es defineix com:

$$f(x) = \sum_{i=1}^{10} \frac{x_i^2}{4000} - \prod_{i=1}^{10} cos(\frac{x_i}{\sqrt{i}}) + 1$$

És una funció d'alta complexitat pels algorismes genètics degut a la seva gran dimensionalitat i a que el productori fa que els gens siguin bastant independents entre ells.

A la taula 5.3 es pot veure la configuració de l'experiment i els resultats obtinguts en aquest problema tenint en compte que s'ha delimitat l'espai a -512 $\leq x_i \leq$ 512 i que l'objectiu del test és minimitzar la funció.

Test: Funció de Griewngk
Dimensió: 10
Algorisme genètic
Població: 1000
Generacions: 1000
Probabilitat d'encreuament: 0.5
Probabilitat de mutació: 0.2
Resultat obtingut: $(3.981, -20.307, 18.496, 19.272, -22.308, -1.718, -8.062, -9.176, 15.915)$
Resultat ideal: $(0, ..., 0)^T$

Taula 5.3: Configuració i resultat del test de la funció de Griewngk

No obstant en aquest problema, es pot veure que els resultats són realment dolents. Això és degut a que no s'ha tingut en compte que la probabilitat de mutació hauria de variar, ja que ens els problemes de dimensió baixa, si per exemple un individu té dos gens i la probabilitat de mutació és de 0.2, de mitjana mutarà un gen cada tres cromosomes, no obstant, si es té la mateixa probabilitat de mutació però el cromosoma té 10 individus, de mitjana mutaran dos gen per cromosoma, que farà que cada generació tots els elements mutin. Per solucionar això s'ha decidit que la mutació serà inversament proporcional a la dimensionalitat del problema, seguint la següent relació:

$$P\_MUTACIO = \frac{1}{ELEMENTS * 2.5}$$

Si es repeteix l'experiment, utilitzant la nova configuració, es pot veure que els resultats obtinguts són molt millors respecte els anteriors que en l'últim experiment com es pot veure en la taula 5.4.

Test: Funció de Griewngk
Dimensió: 10
Algorisme genètic
Població: 1000
Generacions: 1000
Probabilitat d'encreuament: 0.5
Probabilitat de mutació: 0.04
Resultat obtingut: $(0.001, -0.001, 0.001, 0.002, -1.189E-4, 0.001, 0.002, -0.002, 0.01, 0.005)$
Resultat ideal: $(0, ..., 0)^T$

Taula 5.4: Configuració i resultat del test de la funció de Griewngk

En la figura 5.10 es mostra l'evolució del *fitness* tenint en l'eix X les generacions i en l'eix Y el *fitness* dels individus en l'experiment en que s'ha corregit l'error de la probabilitat de mutació.

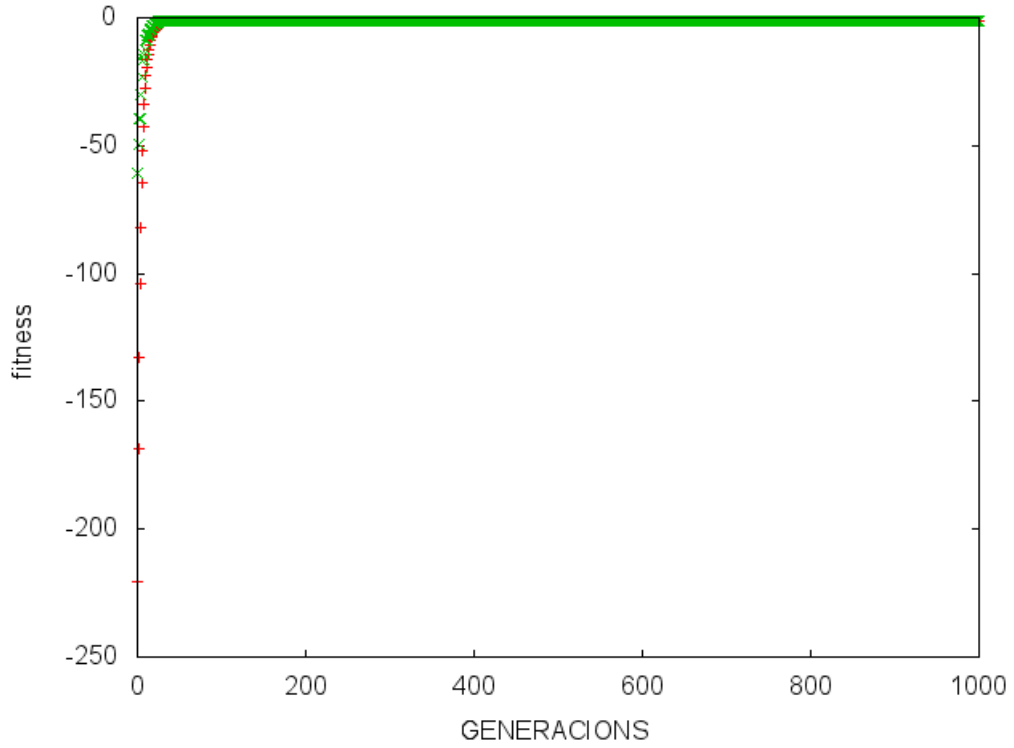

Figura 5.10: Evolucio del fitness durant l'experiment de la funció de Griewngk, en vermell es mostra la mitjana del *fitness*, en verd el màxim.

5.3 Fase 2: Implementació de la xarxa neuronal evolutiva

5.3.1 Recollida de requeriments i anàlisi de la xarxa neuronal

Aquesta segona fase consisteix en dissenyar i implementar la xarxa neuronal que se n'encarregarà posteriorment de la conducció del cotxe en el simulador. Per a poder-ho dur a terme s'han definit les següents especificacions que marcaran aquesta fase:

- La xarxa ha de ser completament configurable i escalable en quant a nombre de capes i de neurones per capa.
- La funció d'activació ha de poder donar una sortida binària com analògica.
- L'entrenament de la xarxa es durà a terme a partir de l'algoritme genètic implementat en la fase anterior.
- La xarxa ha de ser capaç de aproximar qualsevol funció.

Aquests requisits són els que es seguiran durant aquesta fase per a poder dissenyar i implementar la xarxa d'una forma metodològica.

5.3.2 Disseny de la xarxa neuronal

La figura 5.11 mostra el diagrama de classes de la xarxa realitzat seguint les especificacions marcades en el punt anterior.

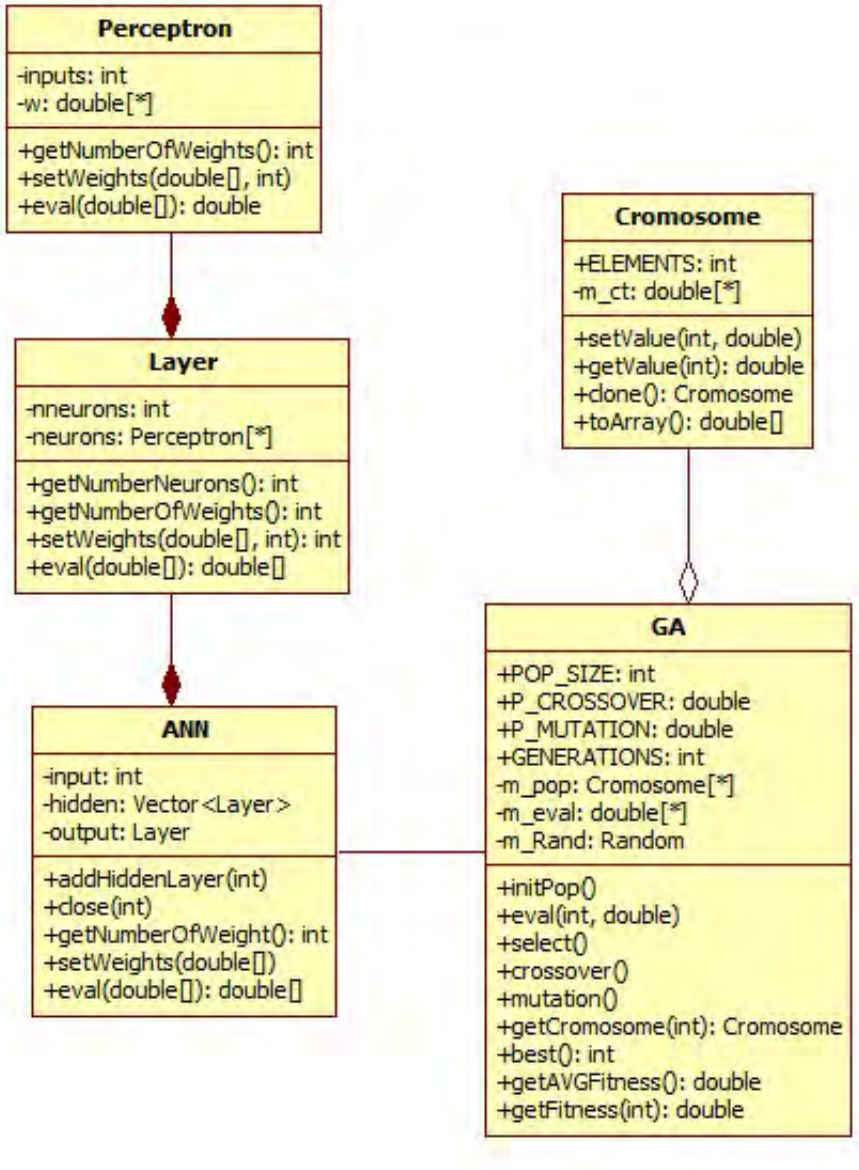

Figura 5.11: Diagrama de classes de la xarxa neuronal.

La classe **Perceptron** conté la informació d'una única neurona. Aquesta té com atributs el nombre d'entrades que té la neurona i un array de tots els pesos de la neurona. Els seus mètodes són els següents:

- **getNumberOfWeight():** Aquest mètode retorna el nombre de pesos total que té la neurona.
- **setWeights(double[],int):** Funció que rep un array amb els pesos i un enter que indica a partir de quina posició estan els pesos que corresponen a la neurona. Retorna un enter que indica quin seria l'índex on hauria d'accedir

la següent neurona per trobar els seus pesos en cas de ser una xarxa amb més d'una neurona.

- **eval(double[]):** A partir d'un array d'entrada que són els valors coneguts de la funció, retorna el valor resultant de la funció de la neurona. Aquest valor pot ser binari si la neurona té la funció esgraó o continu si té una funció lineal.

La classe **Layer** és qui agruparà totes les neurones d'una mateixa capa i se n'encarregarà de processar-les totes alhora. De fet aquesta capa no és més que una capa de connexió entre les neurones i la xarxa neuronal en sí, ja que aquesta conté un nombre que és el nombre de neurones d'aquesta capa i un array de neurones que són les que la formen. Els mètodes són els següents:

- **getNumberNeurons():** Retorna el nombre de neurones que hi ha en aquesta capa.
- **getNumberOfWeights():** Retorna el nombre de pesos totals que hi ha en aquesta capa.
- **setWeights(double[], int):** Aquest mètode crida el mètode *setWeights()* de cada neurona. Passant l'array amb tots els pesos que rep a cada neurona i dient-li a la primera neurona de la capa a partir de quina posició té els seus pesos. Finalment retorna un enter que indica a quina posició començaran els pesos de la següent capa.
- **eval(double[]):** Funció que donades totes les entrades de totes les neurones de la capa, retorna un array que conté les sortides de les neurones de la mateixa capa.

Finalment, la classe **ANN** és la que manega totes les capes formant així la xarxa neuronal. Aquesta classe conté el nombre d'entrades que tindrà la xarxa, un vector amb totes les capes ocultes i finalment la capa de sortida, de forma que si es volgués fer una xarxa de només una capa, caldria crear una capa sense capes ocultes. Els mètodes d'aquesta classe són:

- **addHiddenLayer(int):** Aquest mètode crea una capa oculta amb el nombre indicat per paràmetres de neurones.
- **close(int):** Generà la capa de sortida amb un nombre de neurones igual a l'indicat per paràmetres. Aquest mètode s'ha de cridar sempre per al correcte funcionament de la xarxa.
- **getNumberOfWeights():** Retorna el nombre total de pesos que té la xarxa.
- **setWeights(double[]):** Donat un array que conté tots els pesos de la neurona els assigna a totes les neurones de cada capa.
- **eval(double[]):** Donat un array d'entrades, la xarxa les processa i retorna un array amb les sortides.

5.3.3 Implementació de la xarxa neuronal

Aquest punt explicarà la implementació de l'algoritme a partir del disseny realitzat. Tal com es mostra a la figura 5.12, a l'inici de tot s'ha de crear i configurar la xarxa especificat el nombre d'entrades que tindrà, el nombre de capes i el nombre de neurones per capa. Un cop creada, l'algoritme genètic creat en la fase 1, ens proporcionarà conjunts amb els pesos de cada neurona que la xarxa avaluarà. En aquest cas la xarxa serà supervisada en tot moment de forma que l'avaluació consistirà en comparar els resultats de la xarxa amb els ideals. Finalment el conjunt de pesos per a les neurones serà aquell que ens hagi donat un resultat més aproximat a les solucions reals.

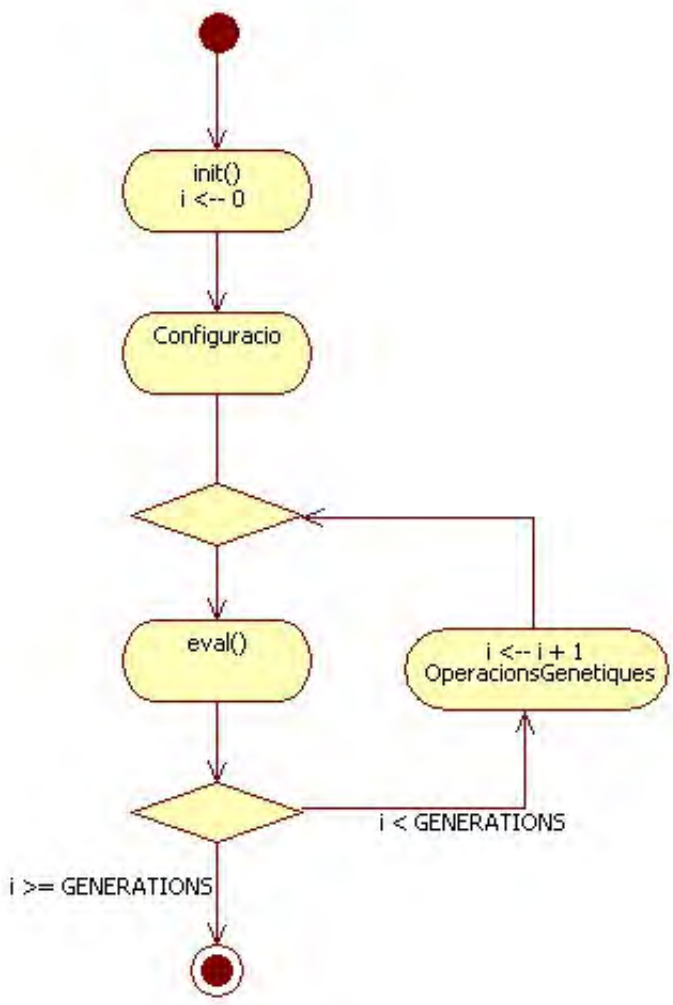

Figura 5.12: Diagrama d'activitats d'una xarxa neuronal.

Com s'ha vist, l'estructura general de l'algorisme és com l'algoritme genètic, afegint la configuració de la xarxa que es pot veure a la figura 5.13 i definint l'avaluació

com el resultat de la xarxa neuronal, cosa que es mostra a la figura 5.14.

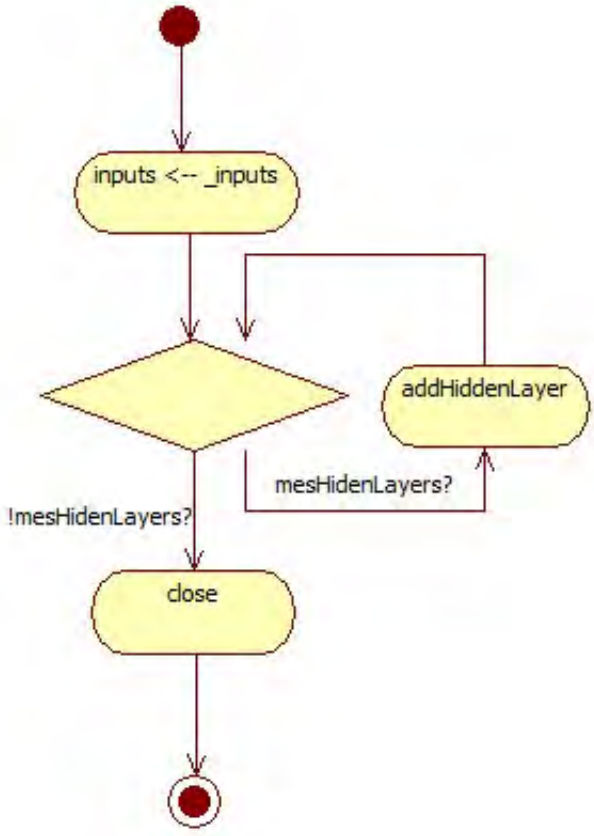

Figura 5.13: Diagrama d'activitats de la configuració de la xarxa neuronal.

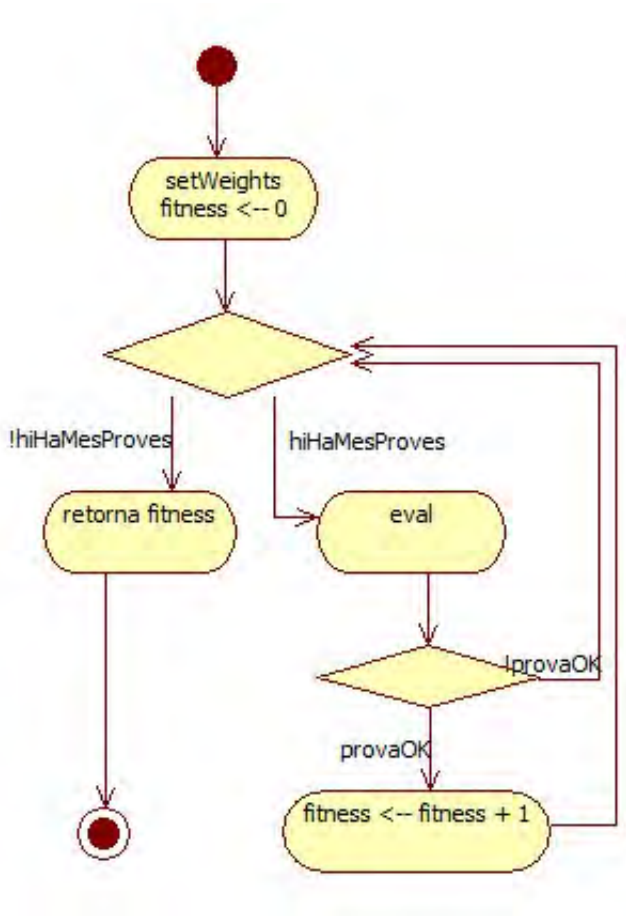

Figura 5.14: Diagrama d'activitats de l'avaluació d'un set de neurones de la xarxa neuronal.

5.3.4 Proves de la xarxa neuronal

En aquesta fase de testeig s'han agafat uns quants problemes de l'UCI repositori [8] per a provar el rendiment de la xarxa. Els test que s'han triat són tant de classificació com de regressió perquè en la següent fase del projecte la funcionalitat de canviar de marxa s'assembla més a un problema de classificació, mentre que la part de gir i d'acceleració és una regressió.

Iris Data Set

El problema de les flors *Iris* és un problema de classificació. A la taula 5.5 es mostren les característiques del problema.

Nom: Iris Data Set
Tipus d'atributs: Real
Nombre d'atributs: 150
Nombre d'instàncies: 4
Falten valors? No

Taula 5.5: Descripció de l'Iris Data Set

La xarxa utilitzada en aquesta prova, com es pot veure a la figura 5.15, està formada per 4 neurones d'entrada, una capa oculta de dues neurones cadascuna i finalment la capa de sortida amb dues sortides. Es considerarà que els codis de les sortides correspondran amb una classe de plantes de la següent forma:

- **Iris-setosa** $\rightarrow 1, 1$
- **Iris-versicolor** $\rightarrow 0, 1$
- **Iris-virginica** $\rightarrow 0, 0$

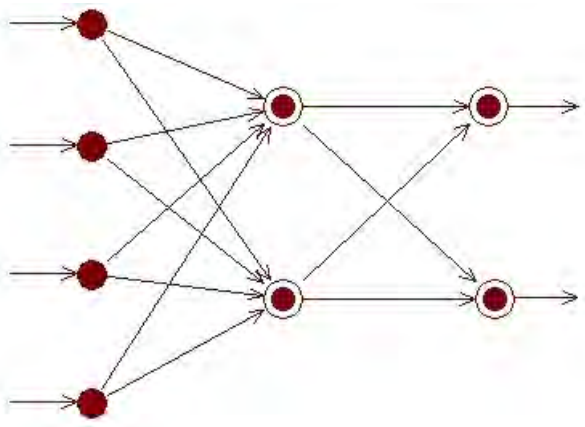

Figura 5.15: Model de la xarxa utilitzada en aquest experiment.

La taula 5.6 mostra la configuració de l'experiment i els resultats obtinguts. Una de les característiques d'aquest dataset és que una de les classes es linealment separable a les altres dues, de fet els resultats mostren que la aquesta classe és la *Iris-setosa*, ja que no hi ha hagut cap errada a l'hora de classificar-les, en canvi les altres dues han sigut les que han fet baixar la eficàcia del sistema (la descripció del problema ja indica que hi ha dos dades que fallaran, per tant els resultats són els esperats). A la figura 5.16 es mostra com el sistema ha evolucionat la xarxa de forma que s'ha aconseguit que classifiques correctament gran part de les dades.

Test: Iris Data Set
Algorisme genètic
Població: 100
Generacions: 1000
Probabilitat d'encreuament: 0.5
Probabilitat de mutació: 0.0333
Xarxa neuronal
Entrades: 4
Capes ocultes: 1
Neurones capa oculta: 2
Sortides: 2
Resultats (mitjana de 10 execucions)
Percentatge d'encerts : 98.4%
Nombre d'encerts: 148
Nombre total: 150

Taula 5.6: Configuració i resultat del problema de l'Iris Data Set

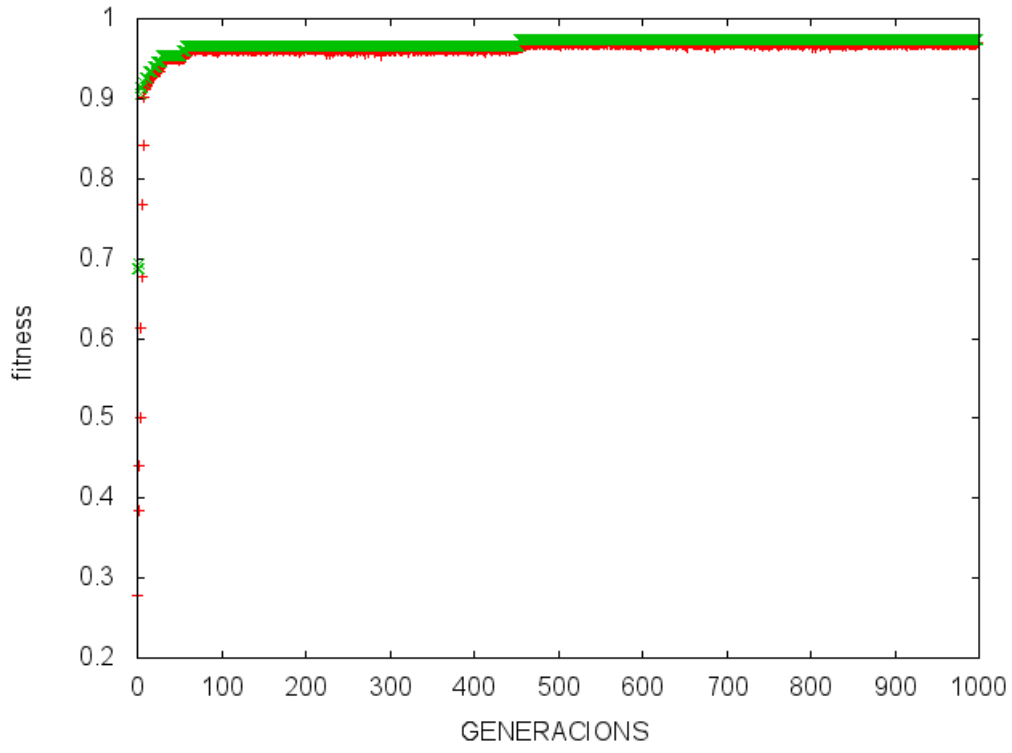

Figura 5.16: Evolució del nombre d'encerts de la xarxa durant totes les generacions,
.

Wine Quality Data Set

El problema de la qualitat del vi, ofereix dos dataset amb dues variants del vi portuguès *Vinho Verde*: el blanc i el negre. Aquests datasets poden ser analitzats ja sigui com un problema de classificació o com un problema de regressió. La taula 5.7 mostra les característiques dels datasets.

Nom: Wine Quality Data Set
Tipus d'atributs: Real
Nombre d'atributs: 6497
Nombre d'instàncies: 12
Falten valors? No

Taula 5.7: Descripció del Wine Quality Data Set

Tractant aquest problema com un problema de classificació, s'ha modelat una xarxa que té 12 entrades, una capa oculta amb una neurona i una capa de sortida també amb una neurona (una és suficient, ja que només hi ha dos classes de vi). La figura 5.17 mostra un esquema de la xarxa modelada per a aquest experiment.

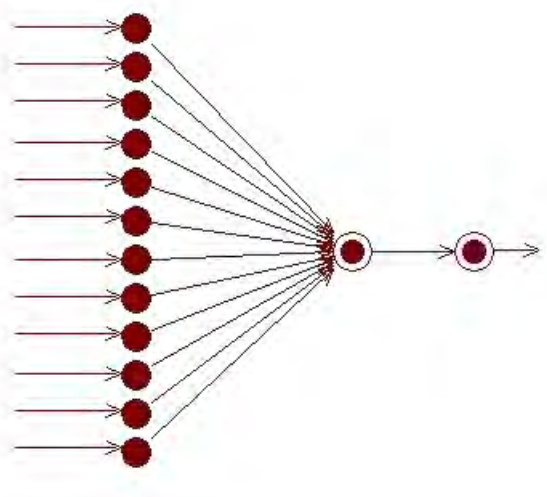

Figura 5.17: Model de la xarxa utilitzada en aquest experiment.

A continuació es mostra en la taula 5.14 la configuració que s'ha utilitzat en aquest experiment i els resultats que s'han obtingut. Comparat amb l'anterior la classificació dels vins dóna molt millor resultat, ja que en deu execucions en totes ha aconseguit un 100% d'encerts. La figura 5.18 mostra l'evolució dels encerts durant un conjunt de 100 generacions.

Test: Wine Quality Data Set
Algorisme genètic
Població: 100
Generacions: 100
Probabilitat d'encreuament: 0.5
Probabilitat de mutació: 0.08
Xarxa neuronal
Entrades: 12
Capes ocultes: 1
Neurones capa oculta: 1
Sortides: 1
Resultats (mitjana de 10 execucions)
Percentatge d'encerts : 100%
Nombre d'encerts: 6497
Nombre total: 6497

Taula 5.8: Configuració i resultat del problema del Wine Quality Data Set

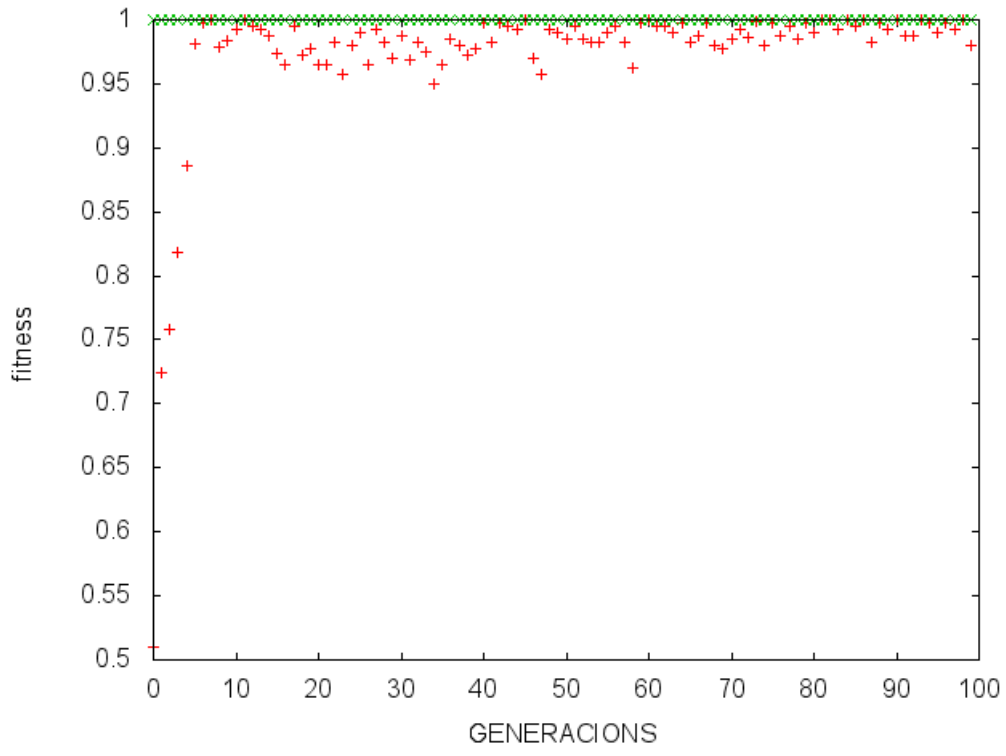

Figura 5.18: Evolució del nombre d'encerts de la xarxa durant totes les generacions, en vermell es mostra la mitjana del *fitness*, en verd el màxim.

5.4 Fase 3: Integració de la xarxa neuronal evolutiva al simulador TORCS

5.4.1 Recollida de requeriments i anàlisi del controlador

Un cop ja ens funciona el nucli d'aprenentatge del s'integrarà en el controlador del simulador TORCS que ve donat per la competició del GECCO [15]. En aquesta fase s'haurà d'adaptar el client de forma que la conducció sigui evolutiva tal i com s'especifica als següents requisits.

- El sistema haurà de ser flexible al paràmetres dels algoritmes ja implementats, és a dir, s'haurà de fer coincidir el nombre d'intents que tindrà el cotxe per a entrenar amb les generacions i el nombre d'individus de cada població.

- S'haurà de trobar una solució que sigui capaç de conduir d'una forma coherent, competitiva, minimitzant els danys del cotxe i el temps que s'està fora del circuit.

5.4.2 Disseny del controlador

El següent diagrama de classes mostrat a la figura 5.19 mostra la integració dels components creats en les dues primeres fases amb el controlador base.

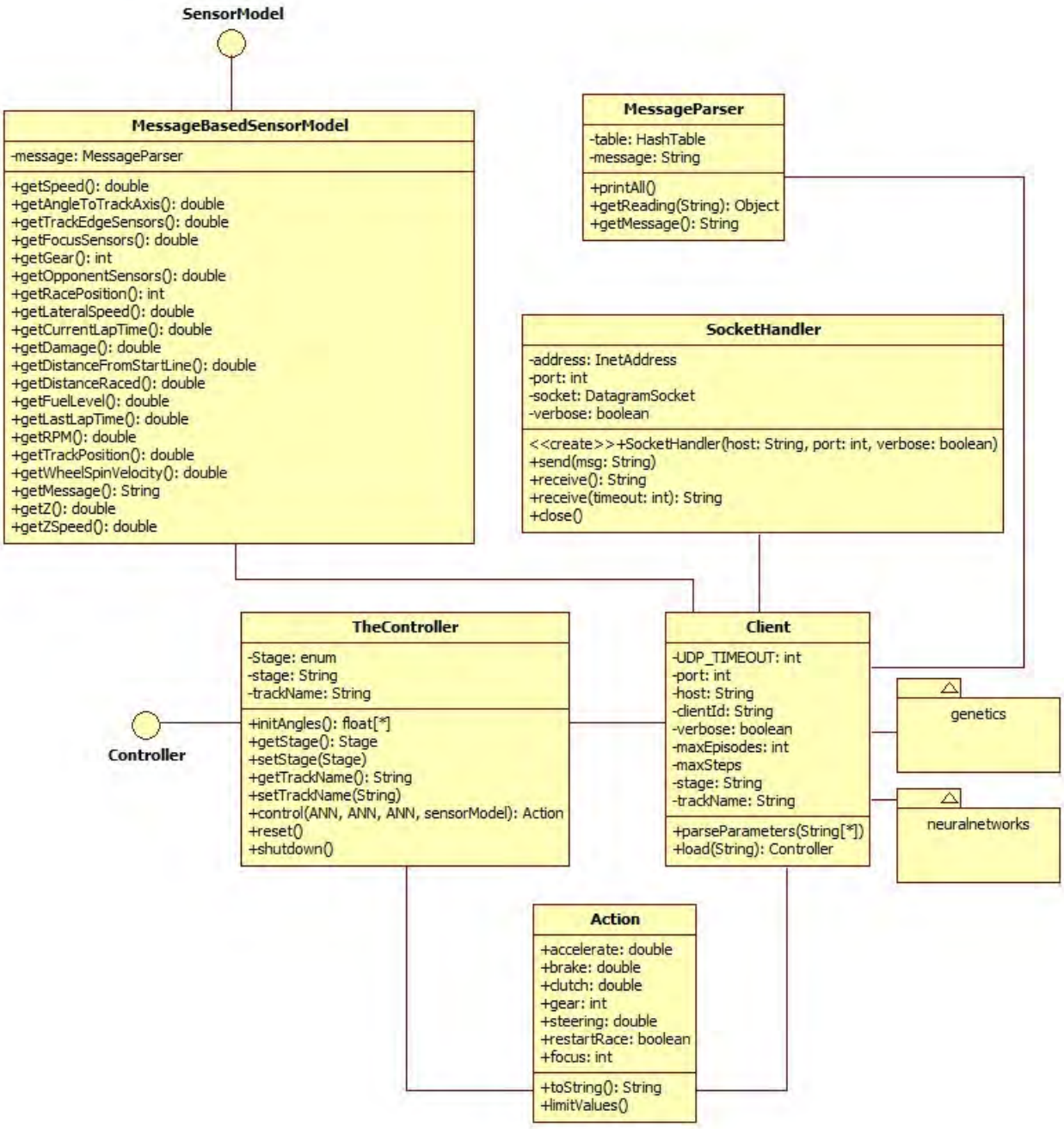

Figura 5.19: Diagrama de classes del controlador amb la xarxa neuronal evolutiva.

La classe principal d'aquesta fase és la classe **Client** que és la que se n'encarrega de connectar-se al servidor i llençar les execucions. A més del *main()* que és qui ho controla tot ja que és qui controla la evolució de l'algoritme genètic, té dos mètode:

- **load(String):** Carrega el controlador que es passa com a paràmetre al programa.

- **parseParameters(String[*]):** Aquesta funció rep tots els paràmetres passats al programa i configura l'execució segons aquests.

Per a connectar-se al servidor s'utilitza la classe **SocketHandler** que és qui conte l'adreça IP, el port i gestiona el *socket* cap al servidor. Els mètodes que conté són:

- **send(String):** Envia una cadena al servidor.
- **receive():** Rep una cadena del servidor.
- **receive(int):** Rep una cadena del servidor però si es sobrepassa el temps indicat aleshores retorna *null*.
- **close():** Tanca la connexió.

La informació que es rep queda emmagatzemada a la classe **MessageBasedSensorModel** que és una implementació de la interfície **SensorModel**. Aquesta classe conté mètodes per a retornar tota la informació indicada al manual [15]. Per a poder parsejar el missatge fa servir la classe **MessageParser** que és qui se n'encarrega de tractar el missatge rebut per la xarxa a un format que separi totes les dades.

En canvi per a enviar el missatge s'encapsulen les accions en la classe **Action**. Aquesta simplement és un contenidor de les dades que se li envien al servidor.

Finalment, la classe **TheController** que implementa l'interfície **Controller** és la que se n'encarrega de la conducció. El mètode més important d'aquesta classe és el mètode *control(ANN, ANN, ANN, sensorModel)* que rep les tres xarxes configurades i avalua cada situació del cotxe indicant que ha de fer i retornant la acció que toca en aquell mateix instant.

5.4.3 Implementació del controlador

En aquesta fase la part d'implementació és menys complexa que les dues fases anteriors ja que com s'ha comentat es tracta d'integrar i adaptar el nucli d'aprenentatge al controlador. El que s'ha fet ha sigut que el sistema realitzi tants intents de conducció com generacions per cromosomes i hagi, d'aquesta forma s'avaluen totes les possibilitats. A més, s'ha hagut de definir què serien les entrades de la xarxa i què les sortides.

De forma que l'aprenentatge sigui més escalonat i senzill, s'ha separat la conducció en tres parts que seran tres xarxes independents entre sí.

- **Xarxa de velocitat:** Aquesta xarxa se n'encarrega de controlar la velocitat a la que va el cotxe. Donades dues entrades, que són la velocitat actual del

cotxe i el resultat de la funció *avgTrackEdge*, aquestes passen per una capa intermitja i finalment hi ha una sortida que és el valor amb el que es pitjaran els pedals. Si aquest valor es positiu és el valor de l'accelerador, si és negatiu és el valor del fre. La figura 5.20 mostra un esquema de la xarxa realitzada.

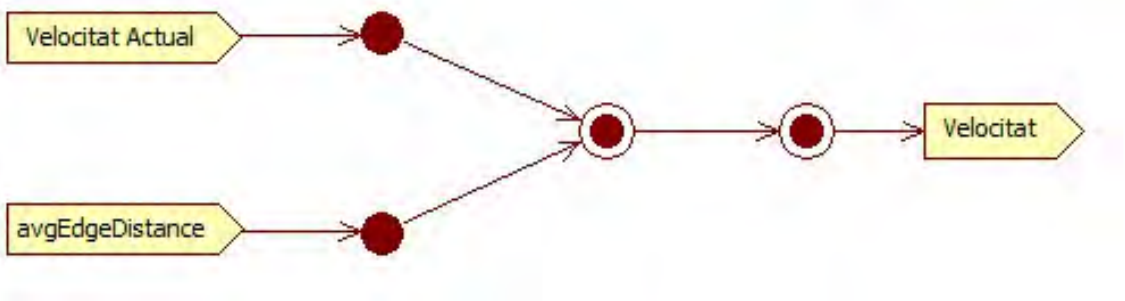

Figura 5.20: Esquema de la xarxa de control de la velocitat

- **Xarxa de gir:** És la encarregada de controlar com agafarà les corbes el cotxe. A la xarxa hi entra l'angle respecte l'eix de la carretera, i després d'una capa intermitja en surt directament el valor que se li donarà al volant del cotxe. L'esquema de la xarxa de gir es mostra a la figura 5.21.

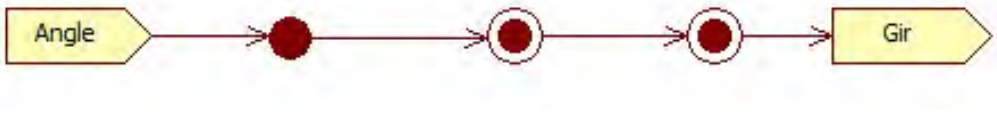

Figura 5.21: Esquema de la xarxa de control de gir

- **Xarxa de canvi de marxes:** Se n'encarrega de canviar de marxes segons les revolucions per minut del cotxe. És a dir, donada l'entrada de les revolucions, i sense capes intermitges, la xarxa té dues sortides que indiquen si s'augmenta de marxa, es decrementa de marxa o no es canvia. Una petita vista d'aquesta xarxa es mostra a la figura 5.22.

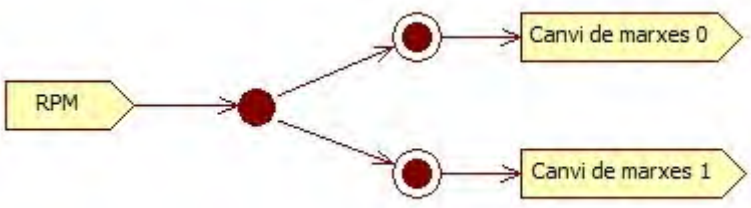

Figura 5.22: Esquema de la xarxa de control del canvi de marxes

Com es pot veure, totes les xarxes minimitzen el nombre de neurones sempre que es pot, això les degut a que una xarxa amb una capa oculta és capaç de resoldre la gran majoria de problemes. [5]

5.4.4 Proves del controlador

Com s'acaba de comentar en el punt anterior, la xarxa implementada es divideix en tres subxarxes. D'aquesta manera les proves que es realitzaran s'aplicaran a cada xarxa per separat i finalment, un cop s'hagin obtingut els resultats es farà una prova amb el sistema global.

Xarxa de control de gir

La primera prova que s'ha realitzat ha sigut la del control de gir i pretén que fent que el cotxe es mogui a una velocitat constant i que el llindar de revolucions en el que el cotxe canvia de marxes estigui ja definit s'entreni la xarxa neuronal.

En aquest cas s'ha forçat que el cotxe circuli a una velocitat de 100 kmh i les marxes es controlen de forma que quan el nombre de revolucions per minut sigui inferior a 2000 RPM i la marxa engranada superior a 2ª aleshores es reduirà la marxa, en canvi si les revolucions són superiors a 7000 RPM i la marxa és inferior a 6ª aleshores s'augmentarà de marxa.

La configuració i el resultat de l'experiment es poden veure a la taula 5.9 i la figura 5.23 mostra l'evolució de la distància recorreguda en cada generació.

Test: Control de gir
Algorisme genètic
Població: 100
Generacions: 35
Probabilitat d'encreuament: 0.5
Probabilitat de mutació: 0.1
Xarxa neuronal
Entrades: 1
Capes ocultes: 1
Neurones capa oculta: 1
Sortides: 1
Fitness: 5437

Taula 5.9: Configuració i resultat de l'entrenament del gir

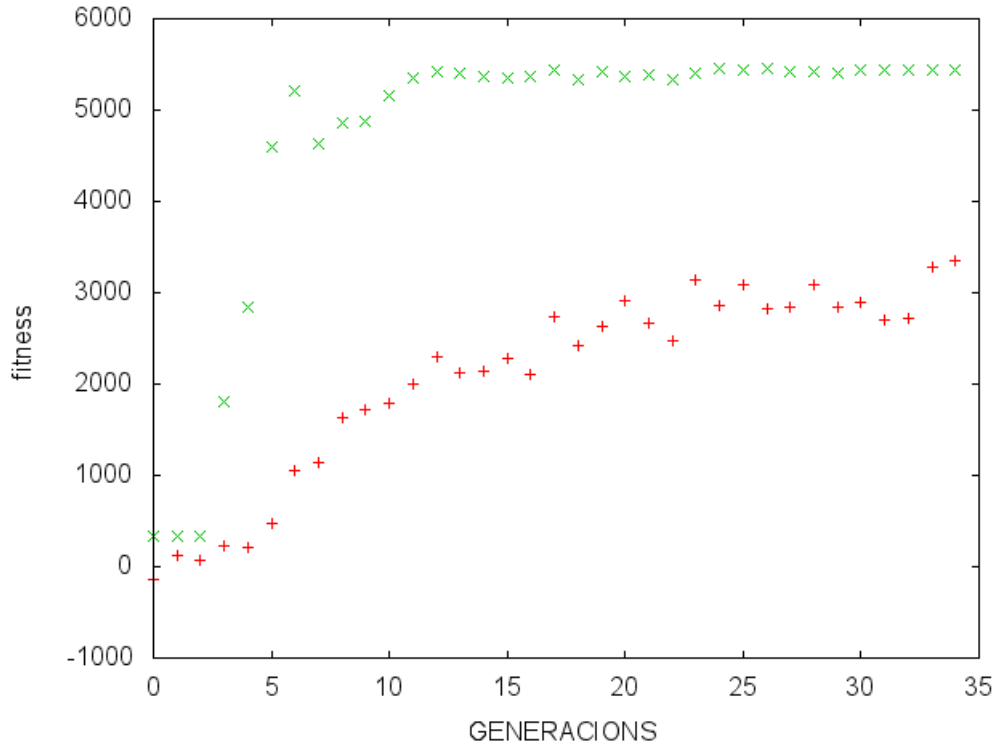

Figura 5.23: Distància recorreguda per cada generació en el test de gir, en vermell es mostra la mitjana del *fitness*, en verd el màxim.

Xarxa de control de velocitat

Aquesta segona prova realitzada vol aconseguir que amb els resultats de l'entrenament de gir ara el cotxe aprengui a córrer variant la seva velocitat anant el més ràpid possible en cada moment. No obstant, el control de revolucions serà el mateix que en la prova anterior.

La taula 5.10 mostra la configuració de l'experiment i el millor resultat obtingut. A més la taula 5.24 mostra com la distància de cada cursa en una mateixa generació augmenta durant aquestes.

Test: Control de velocitat
Algorisme genètic
Població: 100
Generacions: 35
Probabilitat d'encreuament: 0.5
Probabilitat de mutació: 0.08
Xarxa neuronal
Entrades: 2
Capes ocultes: 1
Neurones capa oculta: 1
Sortides: 1
Fitness: 6662

Taula 5.10: Configuració i resultat de l'entrenament de la velocitat

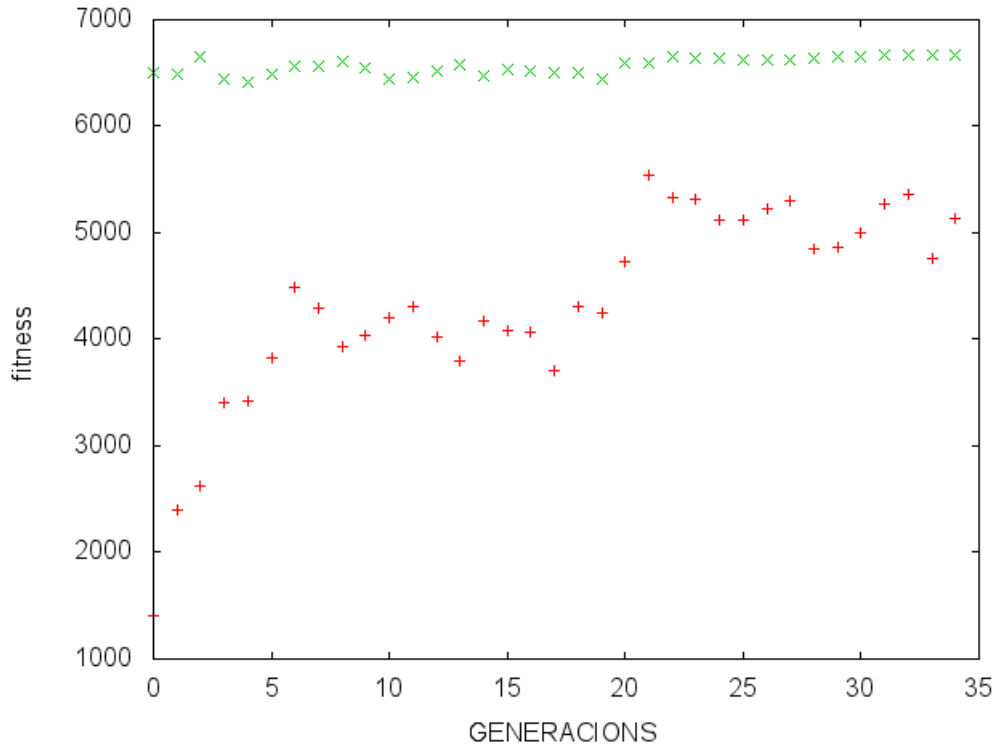

Figura 5.24: Distància recorreguda per cada generació en el test de velocitat, en vermell es mostra la mitjana del *fitness*, en verd el màxim.

Entrenament de canvi de marxes

Aquest experiment utilitza els resultats dels dos experiments anteriors per configurar les altres xarxes. La següent taula 5.11 mostra la configuració i el resultat de

l'experiment, mentre que la figura 5.25 mostra la millora que ens dóna l'algoritme cada generació.

Test: Control del canvi de marxes
Algorisme genètic
Població: 100
Generacions: 35
Probabilitat d'encreuament: 0.5
Probabilitat de mutació: 0.1
Xarxa neuronal
Entrades: 1
Sense capes ocultes
Sortides: 2
Fitness: 6689

Taula 5.11: Configuració i resultat de l'entrenament del canvi de marxes

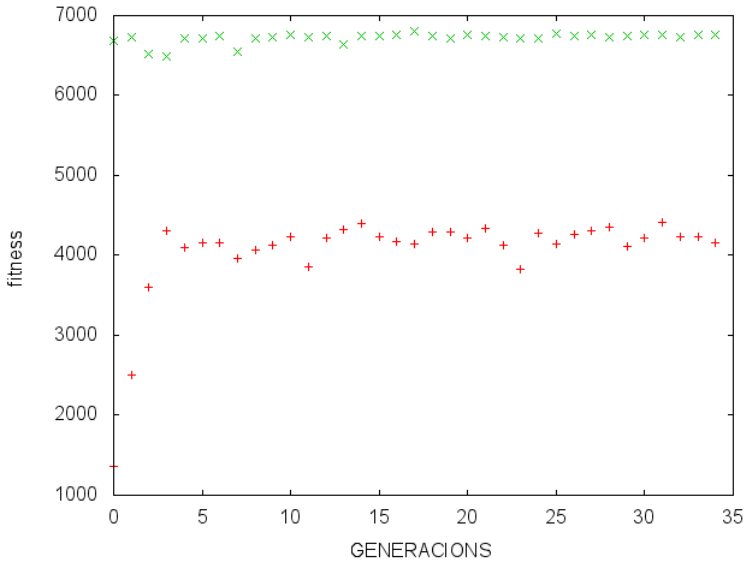

Figura 5.25: Distància recorreguda per cada generació en el test de canvi de marxes, en vermell es mostra la mitjana del *fitness*, en verd el màxim.

Entrenament global

Finalment en aquesta fase s'ha fet un test per a evolucionar totes les tres xarxes de cop, per a acabar d'ajustar els pesos de la xarxa i així arreglar possibles desajustaments degut al fet dels paràmetres que s'havien ajustat a mà.

En aquest test la població de l'algoritme genètic no és aleatòria, en aquest cas s'ha partit de la millor combinació de pesos dels tres experiments anteriors i per

a afegir variabilitat alguns gens han mutat. D'aquesta forma ja es parteix d'un corredor capacitat per a conduir i s'intenta anar millorant el resultat.

Un altre cop es torna a mostra la configuració i el resultat de l'experiment en la taula 5.12 i la figura 5.26 mostra l'evolució del *fitness*.

Test: Control del canvi de marxes
Algorisme genètic
Població: 100
Generacions: 35
Probabilitat d'encreuament: 0.5
Probabilitat de mutació: 0.0333
Xarxa neuronal
Xarxa de control de gir
Xarxa de control de velocitat
Xarxa de control de canvi de marxes
Fitness: 6741

Taula 5.12: Configuració i resultat de l'entrenament global

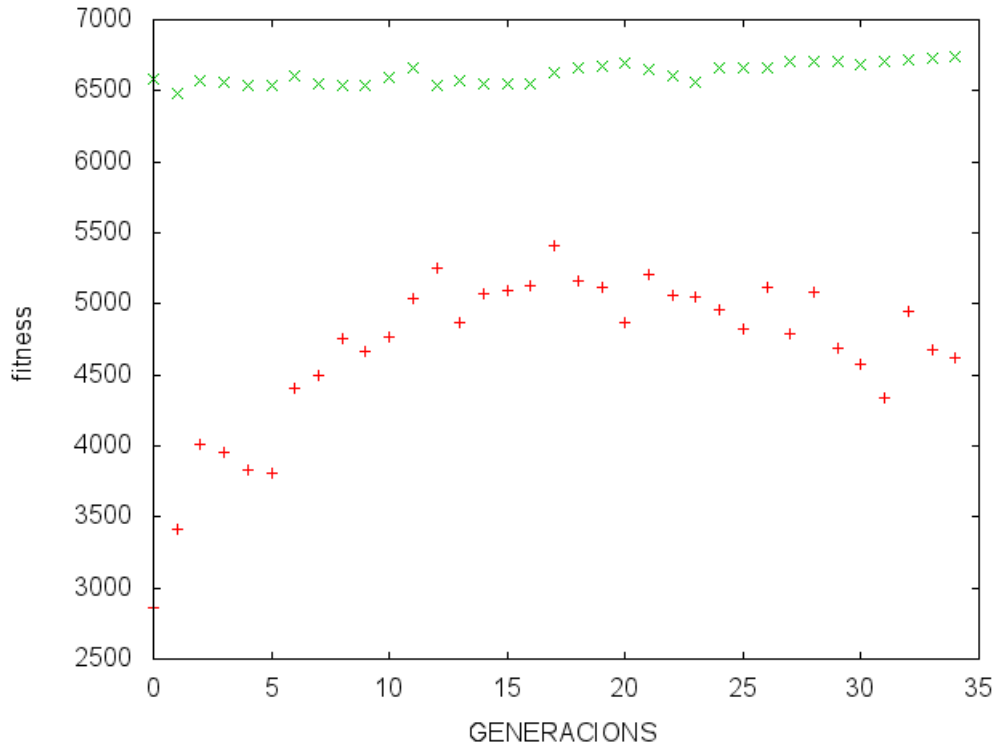

Figura 5.26: Distància recorreguda per cada generació en l'entrenament global, en vermell es mostra la mitjana del *fitness*, en verd el màxim..

5.5 Fase 4: Implementació de la xarxa neuronal amb backpropagation

5.5.1 Recollida de requeriments i anàlisi del backpropagation

La quarta fase consisteix en implementar dins la xarxa creada a la segona fase l'algoritme d'aprenentatge *backpropagation* i en aquest apartat s'han recollit els requeriments per a poder fer-ho.

- Les constants de l'algoritme han de ser fàcilment configurables.

- La implementació del *backpropagation* es durà a terme sobre la xarxa implementada en la fase 2, poden triar sempre el mètode d'entrenament entre l'implementat en aquesta fase o un algoritme genètic.

- L'algorisme ha de ser capaç de solucionar casi qualsevol problema.

5.5.2 Disseny del backpropagation

La figura 5.27 mostra el diagrama de classes de la xarxa amb *backpropagation* realitzat seguint els requeriments marcats en el punt anterior.

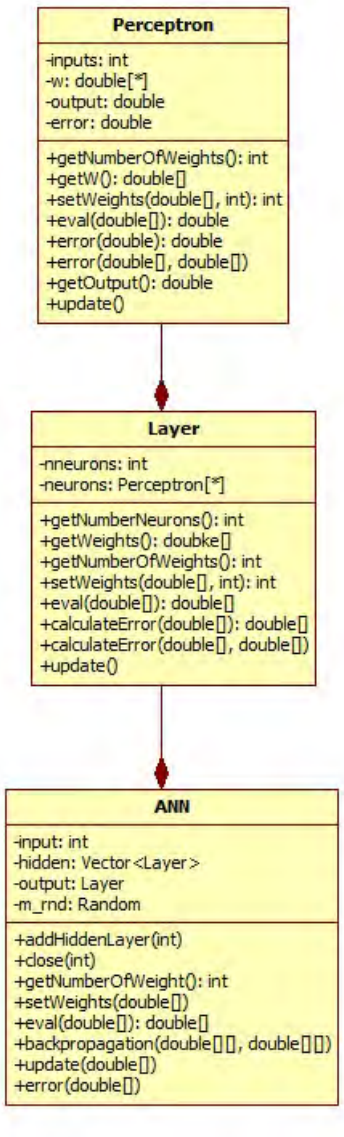

Figura 5.27: Diagrama de classes de la xarxa neuronal amb backpropagation.

En aquest cas les classes són les mateixes que a la fase 2, encara que s'hi han posat alguns mètodes nous. A **Perceptron** s'han definit els següents mètodes:

- **getW():** Retorna un array de nombres que contenen els valors dels pesos.
- **error(double):** Donada la sortida que hauria de treure, es calcula l'error que s'ha obtingut.
- **error(double[], double[]):** Donats els pesos de les neurones de la capa anterior i donat l'error obtingut també a la capa anterior, es calcula l'error a les neurones d'aquesta capa.
- **getOutput():** Retorna la sortida generada en l'últim *eval()*.

- **update():** Actualitza els pesos de ls neurona.

A la classe **Layer** se li han afegit els següents mètodes:

- **calculateError(double):** En una capa de sortida calcula l'error per cada neurona.
- **calculateError(double[], double[]):** A les capes intermitjes calcula l'error per cada neurona.
- **update():** Actualitza tots els pesos de totes les neurones de la capa.

Finalment, a la classe **ANN** se li han definit alguns mètodes:

- **backpropagation(double[][], double[][]):** Recalcula i actualitza els pesos a partir de l'error.
- **update(double[]):** Actualitza tots els pesos de totes les neurones de la xarxa.
- **error(double[]):** Es calcula l'error per cada unitat de la xarxa neuronal.

5.5.3 Implementació del backpropagation

Aquest punt explica la implementació de la xarxa neuronal amb l'algoritme de *back-propagation*. El següent diagrama, que es pot veure a la figura 5.28, mostra el seu funcionament, tenint en compte que la creació de la xarxa i la definició de la seva topologia és igual que a la fase 2. El que varia ara és el fet de que donat un *dataset* la pròpia xarxa es recalibra tenint en compte el seu error.

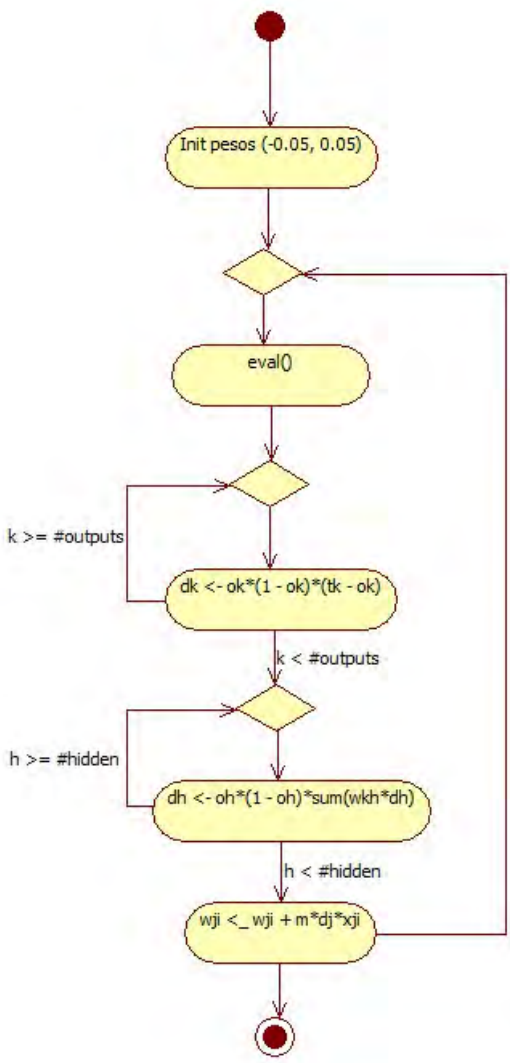

Figura 5.28: Diagrama d'activitats de la xarxa neuronal amb backpropagation.

5.5.4 Proves del backpropagation

Aquesta fase es provarà el funcionament de l'algoritme implementat amb els mateixos problemes de l'UCI repositori [8] que s'han utilitzat a la fase 2. Els tests tornen a ser el de les flors *Iris*i el dels vins.

Iris Data Set

En aquest cas, la xarxa utilitzada torna a ser la mateixa que en la fase 2 i la taula 5.13 mostra la configuració de l'experiment amb els seus resultats

Test: Iris Data Set
Xarxa neuronal
Entrades: 4
Capes ocultes: 1
Neurones capa oculta: 2
Sortides: 2
Backpropagation
Learning rate: 0.05
Iteracions: 1000
Resultats (mitjana de 10 execucions)
Percentatge d'encerts : 98.4%
Nombre d'encerts: 148
Nombre total: 150

Taula 5.13: Configuració i resultat del problema de l'Iris Data Set

Es pot veure que es tornen a obtenir els mateixos resultats que en la fase 2, d'aquesta forma es verifica que els resultats que s'obtenen amb aquest mètode són els mateixos.

Wine Quality Data Set

En aquest cas es torna, també, a utilitzar la xarxa de la fase 2. A continuació es mostra la taula **??** que conté la configuració que s'ha utilitzat a l'experiment i els resultats obtinguts. També es pot observar que es tornen a obtenir els mateixos resultats que a la fase 2.

Test: Wine Quality Data Set
Xarxa neuronal
Entrades: 12
Capes ocultes: 1
Neurones capa oculta: 1
Sortides: 1
Backpropagation
Learning rate: 0.05
Iteracions: 1000
Resultats (mitjana de 10 execucions)
Percentatge d'encerts : 100%
Nombre d'encerts: 6497
Nombre total: 6497

Taula 5.14: Configuració i resultat del problema del Wine Quality Data Set

5.6 Fase 5: Integració de la xarxa neuronal de backpropagation al simulador

5.6.1 Recollida de requeriments i anàlisi del controlador final

L'última fase d'aquest projecte consisteix en adaptar l'algoritme de *backpropagation* al simulador TORCS per a que sigui capaç d'entrenar i actualitzar els valors de la xarxa a l'entrenament en un entorn no supervisat.

- L'algoritme d'entrenament haurà de ser capaç a partir d'un error donat millorar la seva conducció tenint en compte que el valor de l'error serà totalment arbitrari.

- La solució trobada haurà de conduir d'una forma coherent, competitiva i minimitzant les accions que penalitzin el seu resultat.

5.6.2 Disseny del controlador final

El diagrama de classes que es troba a la figura 5.29 mostra el diagrama de classes de l'ultima fase. Com es pot veure la única diferencia és que no utilitza el paquet de genètics.

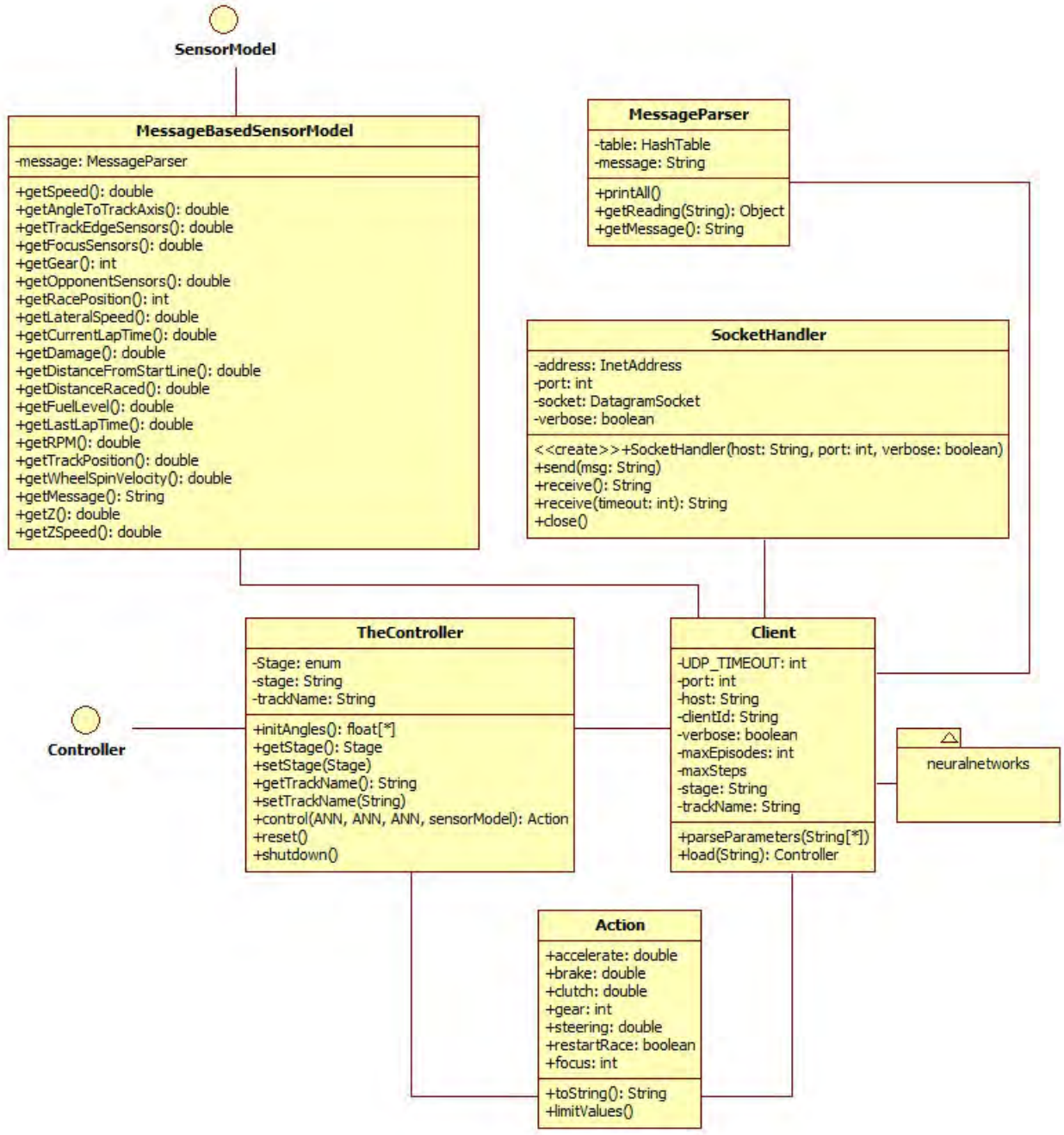

Figura 5.29: Diagrama de classes del controlador amb la xarxa neuronal amb l'algoritme de backpropagation.

5.6.3 Implementació del controlador final

Aquest punt és igual que el descrit a 'Fase 3: Integració de la xarxa neuronal evolutiva al simulador TORCS' a l'apartat 'Implementació del controlador' ja que les

xarxes que s'utilitzen, són les mateixes. La única diferència és que aquesta vegada s'entrena utilitzant l'algoritme de *backpropagation* enlloc d'utilitzar l'algoritme genètic.

5.6.4 Proves del controlador final

Finalment, amb les tres xarxes que s'han utilitzat a la fase 3, s'han realitzat les proves per a veure que un cotxe millora la seva qualitat de conducció utilitzant l'adaptació del *backpropagation* implementat.

En el test la configuració dels pesos inicial correspon amb la millor sortida de la fase 3, de forma que es partirà amb un cotxe que ja es capaç de córrer. La taula 5.15 i la figura 5.30 mostren l'evolució del textitfitness a l'experiment.

Test: Aprenentatge final
Adaptació de backpropagation
Learning rate: 0.05
Iteracions: 100
Xarxa neuronal
Xarxa de control de gir
Xarxa de control de velocitat
Xarxa de control de canvi de marxes
Fitness: 6892

Taula 5.15: Configuració i resultat de l'entrenament final

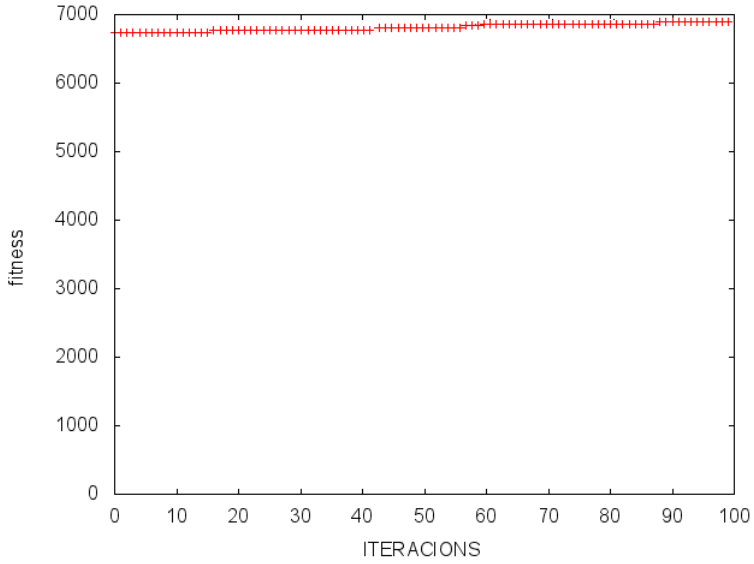

Figura 5.30: Progressió de la distància recorreguda per cada iteració en l'entrenament final

Capítol 6

Resultats i comparativa

6.1 Valoració dels resultats

Al capítol anterior s'ha vist els resultats que s'han obtingut a cada fase i si es miren les fases 3 i 5 (les que implementen controladors) es pot veure que l'aprenentatge per algoritme genètic i l'aprenentatge per *backpropagation* han respost de forma diferent.

Hem de tenir en compte que la variació d'algoritme de *backpropagation* és inviable en un entorn en que el cotxe ha de començar a aprendre de zero, però si el cotxe ja té una base aquest aprèn més ràpidament que l'algoritme genètic.

A la fase 3 s'ha explicat que a l'aprenentatge primer s'ha entrenat cada xarxa per separat i finalment s'han entrenat les tres xarxes alhora de forma que es refinava la conducció i es corregien possibles errors que hi hagués d'abans.

En canvi a la fase 5 s'ha explicat que donada una xarxa ja configurava aquesta es configurava donat una variació de l'algoritme de *backpropagation* on cada iteració de conducció tenia un petit error de conducció definit aleatòriament i s'anava reconfigurant.

El que s'ha vist és que els resultats obtinguts amb l'algoritme genètic i el nou sistema arribaven a uns resultats similars. No obstant, l'algoritme genètic ho feia en 35 generacions de 100 individus cadascuna, mentre que la variació del *backpropagation* ho feia en més o menys 100 iteracions.

Aleshores ha sorgit la pregunta, i no ens podem estalviar fer la segona fase d'entrenament amb l'algoritme genètic i per al refinament utilitzar la nostra versió? D'aquesta manera un cop entrenades les xarxes, per a fer el refinament només caldria passar unes quantes iteracions del *backpropagation* i s'obtindrien els mateixos resultats amb molt menys temps.

Per a demostrar-ho s'ha plantejat el cas on donades tres xarxes entrenades per

separat, s'han refinat els seus pesos utilitzat, per una banda, l'algoritme genètic i per l'altra la versió implementada del *backpropagation*. Les taules 6.1 i 6.2 mostren les configuracions dels dos experiments a comparar i la figura 6.1 mostrà la comparativa de resultats obtinguts en 35 generacions de l'algoritme genètic i 35 iteracions del l'altre alternativa. Cal tenir en compte que cada generació de l'algoritme genètic tardarà 100 vegades més que una iteració.

Test: Comparativa: Entrenament per GA
Algorisme genètic
Població: 100
Generacions: 35
Probabilitat d'encreuament: 0.5
Probabilitat de mutació: 0.0333
Xarxa neuronal
Xarxa de control de gir
Xarxa de control de velocitat
Xarxa de control de canvi de marxes
Fitness: 6197

Taula 6.1: Configuració i resultat del test 'Comparativa: Entrenament per GA'

Test: Comparativa: Entrenament per Backpropagation
Adaptació de backpropagation
Learning rate: 0.05
Iteracions: 35
Xarxa neuronal
Xarxa de control de gir
Xarxa de control de velocitat
Xarxa de control de canvi de marxes
Fitness: 6132

Taula 6.2: Configuració i resultat del test 'Comparativa: Entrenament per Backpropagation'

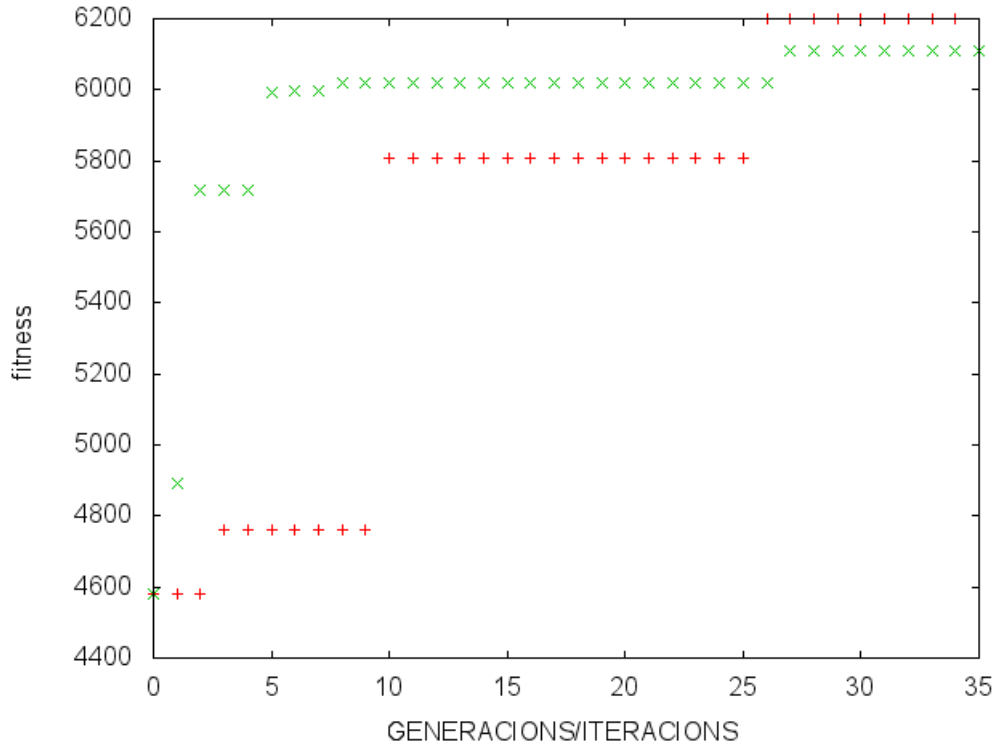

Figura 6.1: Comparativa entre la evolució per entrenament genètic (en verd), com per la variant de *backpropagation*(en vermell)

6.2 Comparativa amb altres sistemes

Com s'ha comentat, aquest problema ve donat per una competició de pilots intel·ligents de cotxes en un simulador, per això ara es compararà el sistema realitzat amb altres d'altres participants. Aquests participants s'enfronten amb els seus algoritmes que són els següents:

- **NEAT [23] (NeuroEvolution of Augmenting Topologies):** És una xarxa neuronal combinada amb un algoritme genètic que a més de codificar en els cromosomes el valor dels pesos també encapsula la topologia de la xarxa.
- **CAT [13] (Cultural Algorithm Toolkit):** És una variant dels algoritmes genètics que amplien l'espai de cerca amb un espai paral·lel.
- *Algoritme evolutiu:* Evoluciona les constants d'unes formules que indiquen com conduir.
- *Rule GA [20]:*És un algoritme genètic que troba regles de conducció a partir de la situació en la que es troba.

Per a comparar els resultats s'han triat els circuits que es va utilitzar a la competició del WCCI al 2008, ja que és l'edició del qual es tenen els resultats i els codis font dels participants. La prova consisteix en fer córrer els cotxes durant 10000 unitats de temps del joc, i l'objectiu és recórrer el màxim de distància possible en aquest temps.

Els circuits són els següents:

- **Ruudskogen (figura 6.2)**: Un circuit que presenta la màxima dificultat en dues corbes de 180º encadenades.

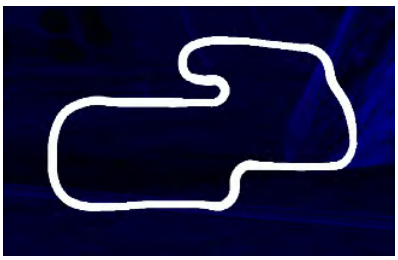

Figura 6.2: Perfil del circuit Ruudskogen

- **Street 1 (figura 6.3)**: Aquest circuit, és caracteritza per tenir corbes molt rectes de 90º i una corba de 180º molt tancada.

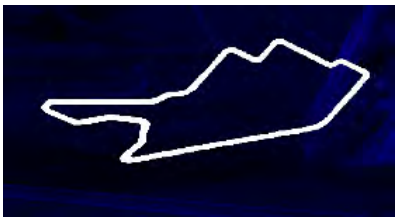

Figura 6.3: Perfil del circuit Street 1

- **D-SpeedWay (figura 6.4)**: Oval amb un gir que dóna la volta a tot el circuit i una recta on es va a fons.

Figura 6.4: Perfil del circuit D-SpeedWay

Com que s'han agafat els circuits i controladors de la competició, s'ha decidit recrear les mateixes circumstàncies. S'utilitzarà un controlador entrenat i només es tindrà una oportunitat per circuit de forma que el resultat que s'obtingui serà el que es mostrarà a la taula 6.3 que mostra els resultats obtinguts amb el sistema implementat als tres circuits. Podem veure que com més abruptes són les corbes pitjor resultat obtenim, ja que en l'oval (D-SpeedWay) obtenim la segona millor puntuació respecte els altres participants, en canvi en el circuit Street 1, obtenim la quarta puntuació allunant-nos del primer.

Controlador	**Ruddskogen**	**Street-1**	**D-Speedway**
Rule GA	3786.91	2984.75	-317.32
CAT	6716.70	3692.94	14406.90
NEAT	5934.00	6477.77	12523.30
Hacked Controller	4134.19	5502.81	12664.50
Evolutiu	3443.45	2998.50	10648.16
HyPER	5476.92	3625.41	13206.53

Taula 6.3: Resultats dels controladors als circuits de la competició. Tots els resultats excepte l'últim s'han extret de[14], els resultats de HyPER són els obtinguts en aquest projecte

En resum, com més obertes i definides són les corbes millor és capaç de conduir el sistema realitzat, no obstant quan el sistema es troba una corba de cop aleshores té problemes. Això pot ser degut a que no hem fet entrenar el cotxe de nou, sinó que tenim una configuració i aquesta és la que provem. Tenint en compte això es creu que els resultats que obtenim són bons, ja que no hi ha cap circuit on es tingui un resultat extremadament dolent respecte els altres participants.

Capítol 7

Conclusions

7.1 Conclusions metodològiques

Al capítol 1, s'han definit un seguit d'objectius. En aquest punt s'avaluaran explicant si s'han assolit o no.

- Assolir uns coneixements elevats sobre tècniques d'intel·ligència artificial com les xarxes neuronal i els algoritmes genètics.

Per la natura d'aquest projecte aquest objectiu s'ha hagut de complir obligatòriament, ja que sense una forta base teòrica el projecte era inviable. Cal dir que quan hem començat aquest projecte, l'única base que es tenia sobre les xarxes neuronals era la apresa a classe, que en aquest context era insuficient. Per això s'han hagut de consultar molts literatura sobre el tema per a veure realment com funcionen i per comprendre realment perquè funcionen. Els algoritmes genètics ja els coneixíem, no obstant, revisant més a fons documents sobre ells hem pogut ampliar el nostre coneixement sobre ells i aclarir idees que potser fins ara no havíem assimilat del tot.

Aquest objectiu dóna peu al següent, que era:

- Dissenyar, implementar i verificar el funcionament correcte d'un algoritme que sigui capaç de competir en una cursa de cotxes.

Aquest objectiu feia referència al punt més fort d'aquest projecte: el sistema intel·ligent. Després de tot els temps dedicat i l'esforç que ha requerit podem dir, també, que s'ha complert l'objectiu. S'ha aconseguit un sistema capaç de córrer d'una forma competitiva en casi qualsevol circuit, i a pesar que potser no seria un candidat a participar a la competició, potser amb un parell de retocs pot arribar molt lluny.

Cal comentar també que la metodologia utilitzada descrita al capítol 2 ha sigut un gran encert. Ja que la desfragmentació del projecte en diverses fases independents ha fet que realment quan començàvem una fase nova ja sabíem que la fase anterior funcionava perfectament. Això ha anat molt bé ja que la majoria de fases eren un component que formava part de la següent.

7.2 Conclusions científiques

Aquest projecte pretenia aprofundir-nos en el món de les xarxes neuronals i aplicar-les a un problema el més real possible que en aquest cas era una competició de cotxes que conduïen sols.

Després d'haver realitzar el projecte hem après que les xarxes neuronals són un molt bo aproximador de funcions i classificador, ja que com em pogut veure en les proves de la xarxa on classificàvem flors i vins i en el sistema integrat on aproximava les funcions de conducció, els resultats que s'han obtingut han sigut bons.

Els avantatges que li hem trobat al sistema proposat han estat:

- La facilitat amb la que es pot entrenar una xarxa neuronal és molt gran, ja que de les dues formes en les que s'ha fet, el *backpropagation* es un algoritme que té suficient documentació per a poder entendre el seu funcionament correctament i els algoritmes genètics donen l'opció a entrenar-la aleatòriament convergint cap a un resultat òptim.

- El sistema necessita poques dades per a poder prendre una decisió correcta. Cal veure com de tots els sensors explicats en el capítol 4, només se n'han necessitat una petita part. A més afegint que gràcies a l'algoritme genètic tampoc necessitem molta informació sobre l'espai de cerca (podíem estar a un circuit desconegut) el cotxe s'adaptava molt bé a casi qualsevol situació.

- La flexibilitat de tots els paràmetres de configuració de la xarxa neuronal i de l'algoritme genètic és enorme. De fet, gràcies a la implementació que s'ha fet, a l'inici del programa s'han d'indicar tots els paràmetres abans de començar a córrer, d'aquesta manera es pot llençar l'experiment amb qualsevol configuració de forma que si, per exemple, es volia un experiment d'una duració més curta només calia indicar-ho al paràmetre indicat. Fins i tot, altres paràmetres com el percentatge d'encreuament, de mutació o el ritme d'aprenentatge de la xarxa neuronal permeten determinar un equilibri entre una cerca determinista i una cerca aleatòria.

- A pesar que s'ha integrat al client, el nucli intel·ligent es podria exportar a qualsevol altre projecte on el problema fos diferent ja que, com s'ha dit, la informació de l'espai de cerca requerida és molt petita i, per tant, seria perfectament adaptable.

No obstant, el nostre sistema presenta algun inconvenient:

- La velocitat d'aprenentatge és molt baixa, ja que s'ha d'utilitzar l'algoritme genètic per a entrenar la xarxa neuronal i això implica un elevat nombre d'elements en una població alhora que un gran nombre de generacions. Tenint en compte que l'avaluació del problema té un cost temporal molt elevat, aleshores fa que un entrenament complert pugui durar setmanes.
- La dimensionalitat del problema és molt gran. Això no seria cap problema, com s'ha vist al capítol 5, si no perquè com s'acaba de dir l'avaluació és molt lenta i això fa que amb el temps disponible s'hagi hagut de reduir el temps dels entrenaments i per tant no obtenir els millors resultats que es podrien obtenir.

Resumint, tenim un sistema flexible, eficaç i adaptable a altres problemes però la eficiència és el seu punt feble. No obstant, això ve donat en gran part per culpa del simulador i no del sistema pròpi i que en altres problemes, a pesar que l'eficiència encara no fos ideal, podria tardar bastant menys temps. Per això s'ha trobat en les xarxes neuronals una eina molt útil per encarar problemes de classificació o de regressió.

Capítol 8

Línies de futur

Aquest projecte deixa moltes portes obertes per a la seva continuïtat, ja que el mateix problema es pot intentar encarar d'una altra forma o la tècnica desenvolupada es pot utilitzar per a intentar resoldre altres problemes.

Un primer camí a seguir és solucionar el problema utilitzant altres tècniques d'aprenentatge per a posteriorment realitzar una comparativa i veure quina de totes és la millor. En aquest punt es podria realitzar un sistema híbrid que utilitzes la millor tècnica i s'intentessin solucionar els seus punts dèbils amb una tècnica auxiliar. Un exemple seria que l'entrenament fos supervisat de forma que el cotxe enlloc d'aprendre corrents extragués les regles d'un cotxe que funciona bé i després per acabar de refinar el seu funcionament s'utilitzés l'adaptació del *backpropagation* realitzat en aquest projecte per a millorar el seu funcionament respecte el mestre.

De fet, un objectiu personal d'aquest projecte és continuar amb l'evolució del sistema per a que tingui un nivell suficientment competitiu per a poder-lo enviar als congressos que fan les competicions i que tingués opcions de quedar en les primeres posicions o, fins i tot, optar a la victòria.

Una altra línia de futur que es desvia del camí de les anteriors és el fet d'implementar el sistema per a un hardware, de forma que algun tipus de vehicle real fos capaç de córrer per un circuit. En un principi s'ha pensat el l'LSMaker, de forma que amb un sistema una mica més reduït que el realitzat en aquest projecte aprengués a donar voltes per un circuit. No obstant, a més llarg termini i comptant amb molt més pressupost seria interessant aconseguir implantar el sistema a un cotxe de veritat que fos capaç de conduir per una carretera o per un circuit. El problema d'aquest últim punt és que amb el sistema implementat en aquest projecte, en les primeres fases d'entrenament s'hauria de realitzar sobre un simulador més professional, i un cop ja corregués aleshores sí que es podria posar el sistema a un cotxe.

Bibliografia

[1] Bernadó, E., Solé, X., and Costa, E. *dpo*^{2} project-based learning: Software development methodologies seminar i, 2011.

[2] Boehm, B. W. A spiral model of software development and enhancement. *IEEE Computer 21*, 5 (1988), 61–72.

[3] Costa, E. ega: Un controlador genètic pel simulador torcs, 2009.

[4] Costa, E. Machine learning, 2010.

[5] Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS) 2*, 4 (Dec. 1989), 303–314–314.

[6] Deb, K., and Kumar, A. Real-coded genetic algorithms with simulated binary crossover: Studies on multimodal and multiobjective problems. In *Complex Systems* (1995), pp. 431–454.

[7] DeJong, K. A. An analysis of the behaviour of a class of genetic adaptive system, 1975.

[8] Frank, A., and Asuncion, A. UCI machine learning repository, 2010.

[9] Goldberg, D. E. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley, 1989.

[10] Gonzalo, L. *Inteligencia humana e inteligencia artificial*. Libros Mc Series. Palabra, 1987.

[11] Gordonn, V. S., and Whitley, D. Serial and parallel genetic algorithms as function optimizers. In *Proceedings of the Fifth International Conference on Genetic Algorithms* (1993), Morgan Kaufmann, pp. 177–183.

[12] Hinterding, R., Gielewski, H., and Peachey, T. C. The nature of mutation in genetic algorithms. In *In Proceedings of the Sixth International Conference on Genetic Algorithms, edited by L.J. Eshelman* (1995), Morgan Kaufmann, pp. 65–72.

[13] Kinaird-Heether, L. Racing controller trained with cultural algorithms, 2008.

[14] Lanzi, P., Togelius, J., and Loiacono, D. Car racing @ wcci 2008 $http://cig.ws.dei.polimi.it/?page\_id = 5$.

[15] Lanzi, P. L., Loiacono, D., and Togelius, J. *Simulated Car Racing Championship Competition Software Manual*, 2011.

[16] Michalewicz, Z. Statistical and scientific databases.

[17] Michalewicz, Z. *Genetic algorithms + data structures = evolution programs*, 3rd ed. Springer-Verlag, London, UK, 1996.

[18] Mitchell, T. M. *Machine Learning*. McGraw-Hill, New York, 1997.

[19] Orriols, A., and Sancho, A. Introduction to machine learning: Lecture 7 artificial neural networks, 2010.

[20] Pérez, D. Rule ga controller, 2008.

[21] Rosenblatt, F. The perceptron - a perceiving and recognizing automaton.

[22] Rosenbrock, H. H. An automatic method for finding the greatest or least value of a function. In *The Computer Journal 3* (1960), pp. 175–184.

[23] Simmerson, M. Neat4j, 2008.

[24] Sywerda, G. Uniform crossover in genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms* (San Francisco, CA, USA, 1989), Morgan Kaufmann Publishers Inc., pp. 2–9.

[25] Toscano, G. Computación evolutiva - mutación: Mutación para representación real.

[26] Widrow, B., and Hoff, M. E. Adaptative switching circuits.

[27] Widrow, B., and Lehr, M. A. 30 years of adaptive neural networks, 1990.