Implementació d'una xarxa neural configurable amb optimització genètica

Jordi Mas Mateo
Grau Enginyeria Informàtica
Àrea Intel·ligència Artificial

David Isern Alarcón Ph.D.
Carles Ventura Royo Ph.D.

Juny de 2016

FITXA DEL TREBALL FINAL

Títol del treball:	Implementació d'una xarxa neural configurable amb optimització genètica
Nom de l'autor:	Jordi Mas Mateo
Nom del consultor:	David Isern Alarcón Ph.D.
Nom del PRA:	Carles Ventura Royo Ph.D.
Data de lliurament:	Juny de 2016
Titulació:	Grau Enginyeria Informàtica
Àrea del Treball Final:	Intel·ligència Artificial
Idioma del treball:	Català
Paraules clau:	xarxa neural, hiperparàmetres, algorisme genètic

Resum:

En aquest treball s'implementarà una xarxa neural de tipus perceptró multicapa, els hiperparàmetres de la qual s'ajustaran per mitjà d'un algorisme genètic, que intentarà trobar una configuració propera a l'òptima per a un conjunt de dades donat. Es tracta d'una prova de concepte que fusiona dos algorismes inspirats en la natura: les xarxes neurals que imiten el funcionament de les neurones biològiques, i els algorismes genètics que imiten la selecció natural. Algorismes com aquest ja s'han proposat i estudiat abans, però no existeix cap implementació per al programari estadístic R. Així, el producte final d'aquest treball és un paquet R que es distribuirà sota llicència GPL.

Abstract:

In this work, a multilayer perceptron neural network will be implemented, hyperparameters of which will be tuned by a genetic algorithm, that will try to find a near-optimal configuration for a given dataset. This is a proof of concept that will merge two algorithms inspired by nature: neural networks that imitates biological neurons operation, and genetic algorithms that mimics the natural selection. Algorithms like that had been proposed and studied before, but does not exist any implementation of it for the R statistical software. Thus, the final product of this work is an R package that will be distributed under the GPL.

Contingut

Índex de figures

Índex de taules

1 Introducció

1.1 Context i justificació del treball

Aquest treball es situa en el context de la intel·ligència artificial, d'on s'utilitzaran tècniques de dues branques, l'aprenentatge automàtic i els algorismes evolutius. L'aprenentatge automàtic persegueix, entre d'altres, l'objectiu de construir models matemàtics amb capacitat predictiva a partir d'exemples, tècnica coneguda com *aprenentatge inductiu*. D'altra banda, els algorismes evolutius imiten l'evolució de les espècies biològiques per realitzar algun tipus d'optimització.

Un dels algorismes més destacats d'aprenentatge inductiu són les xares neurals. Aquestes es fonamenten en una unitat de càlcul inspirada en el funcionament de les neurones biològiques, descrita per primer cop al 1943 per Warren McCulloch i Walter Pitts, i perfeccionada posteriorment per Frank Rosenblatt al 1958. Aquesta unitat de càlcul rep noms com *neurona artificial* o *perceptró*. Les neurones artificials es poden interconnectar en forma de graf dirigit, amb mides i complexitats diverses, donant lloc a les *xarxes neurals*. La principal característica d'aquestes xarxes és que tenen la capacitat d'aproximar qualsevol funció matemàtica. No importa la complexitat de la funció ni les seves dimensions, sempre es podrà trobar una xarxa neural que l'aproximi amb un error fitat. George Cybenko va demostrar aquest teorema l'any 1989 [Cyb89].

Amb aquesta propietat, les xarxes neurals poden utilitzar-se per a fer qualsevol tipus de regressió estadística, amb múltiples variables explicatives i múltiples variables resposta. Si les variables resposta són categòriques, les xarxes neurals es converteixen en algorismes de classificació. La regressió i el seu cas particular, la classificació, amb totes les seves aplicacions pràctiques, són les dues utilitats fonamentals de les xarxes neurals en l'aprenentatge automàtic. Però el treball de Cybenko és un teorema d'existència, que no dona cap indicació de com trobar la XN per a un problema de regressió concret.

Al 1974 Paul Werbos va proposar l'algorisme de *retropropagació*, conegut en altres àrees des dels anys 1960, per a l'entrenament de xarxes neurals. Amb aquest avanç es va resoldre el problema d'ajustar les connexions sinàptiques de la xarxa neural, doncs la retropropagació és un algorisme d'optimització que troba la millor assignació possible per als pesos sinàptics. Però la retropropagació nomes resol una part del problema, segueix oberta la qüestió de trobar una configuració adient per a la XN, és a dir, trobar una topologia i un procés d'aprenentatge tals que la XN sigui capaç d'aprendre correcta i eficientment de les dades.

La configuració d'una XN s'ajusta per mitjà d'hiperparàmetres[1] que són dependents de cada problema o conjunt de dades a tractar. Trobar una configuració adient és una tasca manual de prova i error [Alm08, Cur11], que és possible automatitzar total o parcialment mitjançant tècniques de cerca com *grid search*, cerca estocàstica o inclús optimització [Ben12]. No hi ha regles clares per escollir aquests hiperparàmetres ni metodologies per guiar les cerques, però la seva adequada configuració determinarà la precisió i l'eficiència del model de XN resultant.

Al 1950 Alan Turing va proposar que podria imitar-se l'evolució natural per aconseguir màquines que milloressin la seva capacitat d'aprenentatge en successives generacions. El material hereditari determinaria l'estructura de la màquina, les mutacions d'aquest material els canvis generacionals, la opinió d'un experimentador seria la selecció natural [Tur50]. Aquest procediment es coneix actualment com *algorisme genètic*, i s'utilitza com a mètode d'optimització estocàstica quan altres mètodes d'optimització no són factibles, per exemple, si la mida de l'espai de cerca és molt gran i no pot definir-se una heurística que serveixi de guia. El seu inconvenient és que al ser mètodes estocàstics, els algorismes genètics no sempre troben la millor solució, sinó que s'aproximen a la solució òptima [Tor10, Gre00].

La idea d'Alan Turing pot aplicar-se a les xarxes neurals, que serien la seva *learning machine*, usant un algorisme genètic que imita l'evolució natural, per millorar la seva capacitat d'aprenentatge generació rere generació. És també un plantejament interessant perquè l'evolució de les espècies

[1] Es distingeix entre *hiperparàmetres*, els valors que determinen la topologia i les característiques d'aprenentatge d'una xarxa neural, en contraposició als *paràmetres* que són els valors que troba la xarxa neural durant el procés d'aprenentatge, per ajustar un model matemàtic a partir de les dades [Goo16, Ben12].

a conduit a l'aparició d'éssers amb capacitat d'aprenentatge, que és precisament la característica que s'espera de les xarxes neurals com a algorisme d'aprenentatge automàtic. Més encara, quan les xarxes neurals també s'inspiren en les neurones biològiques, sembla natural combinar tots dos algorismes.

Hi ha dues orientacions per combinar xarxes neurals amb algorismes genètics [Koe94]. La més antiga consisteix en usar l'algorisme genètic per trobar directament els paràmetres de la xarxa, que es coneixen com pesos sinàptics. Aquest enfoc ha quedat superat pels algorismes de retropropagació. L'altra aproximació consisteix en usar l'algorisme genètic per configurar els hiperparàmetres de la xarxa neural per a cada problema concret, i deixar que un algorisme d'entrenament trobi els paràmetres a partir de les dades. Així l'algorisme genètic s'usa com a mètode d'optimització estocàstica per trobar una configuració propera a l'òptima. Aquesta darrera és l'aproximació que se seguirà en aquest treball.

A continuació es proporcionen diverses citacions textuals, que donen una idea de la dificultat d'ajustar aquests hiperparàmetres manualment:

> "[...] one remains with a substantial number of choices to be made, which may give the impression of neural network training as an art." [Ben12]
>
> "Although neural networks are an established tool, their optimization remains a matter of active research. There is as yet no algorithm that can guarantee to create the optimal network, which is not surprising because topology optimization must satisfy multiple goals: the 'perfect' network must yield dependable predictions, avoid overtraining, provide faster convergence, minimize training time, [...]" [Cur11]
>
> "The construction of near-optimal ANN configurations involves difficulties such as the exponential number of (hyper)parameters that need to be adjusted; the need for a priori knowledge of the problem domain and ANN operation in order to define these (hyper)parameters." [Alm08]
>
> "La tria dels (hiper)paràmetres és molt important [...]. No existeixen mètodes per estimar els (hiper)paràmetres. A vegades per a construir l'arquitectura de la xarxa es fan servir algorismes genètics." [Tor13]

A més de tractar-se d'un problema interessant per si mateix, la configuració automàtica de xarxes neurals pot ser una eina útil en els casos on cal crear un model de regressió o classificació, però no es disposa de temps o recursos per a experimentar amb diferents topologies de xarxa i configuracions d'aprenentatge. No sempre es tractarà de grans volums de dades, sovint poden ser datasets de mides petites o moderades, massa complexos per ser tractats amb eines d'estadística clàssica com la regressió lineal, però que una xarxa neural configurada amb un algorisme genètic podria resoldre en un PC en un temps raonable.

Per això, i com que no s'ha trobat cap implementació de codi obert que ofereixi aquesta funcionalitat, el programari desenvolupat en aquest treball es distribuirà sota llicència GPL, i es dissenyarà per ser usat com a extensió opcional del software estadístic R.

1.2 Objectius

L'objectiu d'aquest treball es implementar una prova de concepte[2] de l'ús d'un algorisme genètic per configurar una xarxa neural segons s'ha descrit a la secció anterior. Per poder posar a prova la implementació, i per tal que aquesta sigui útil en posteriors problemes, caldrà integrar-la en un paquet estadístic que faciliti mecanismes d'entrada i sortida, un llenguatge d'scripting i funcions estadístiques.

Els objectius generals del treball són:

- Desenvolupar un algorisme genètic, que aproximi la configuració òptima per als hiperparàmetres d'una xarxa neural.

[2] De l'anglès *Proof of Concept (PoC)*. És una implementació, sovint resumida o incompleta, d'una idea, amb el propòsit de verificar que és susceptible de ser explotada d'una manera útil.

- Integrar el resultat en un software estadístic existent.
- Posar a prova la implementació amb dades reals.

Aquests objectius generals, degut a factors com la limitació temporal del treball, i que les xarxes neurals són encara tema de recerca[3], es redueixen a les següents directives específiques:

- Implementar una xarxa neural, de tipus perceptró multicapa, amb els següents hiperparàmetres configurables: nombre de capes, mida de cada capa, factor d'aprenentatge i moment.
- Implementar un algorisme de retropropagació per a entrenar la xarxa a partir d'exemples.
- Trobar una codificació adient, que permeti convertir els valors dels diversos hiperparàmetres de la xarxa neural, en cromosomes per a l'algorisme genètic.
- Implementar un algorisme genètic que, donada una generació de xarxes neurals, un conjunt de dades d'entrenament, i un temps limitat, trobi una configuració final millor que les generacions precedents.
- Es considerarà que la configuració final és millor si:
  1. La XN final aprèn més que les inicials, sense memoritzar les dades[4].
  2. La XN final aprèn el mateix que les inicials, però és de mida menor.
  3. La XN final aprèn el mateix i és de la mateixa mida que les inicials, però aprèn més ràpid, i.e. el seu entrenament és més eficient (millora opcional).
- Integrar la implementació en el software estadístic R.
- Aprofitar la integració amb R, per posar a prova el software desenvolupat amb dades reals. S'utilitzaran conjunts de dades de [Pre94, Lic13, Lec99].

Possibles millores:

- Utilitzar càlcul paral·lel on sigui factible, per aprofitar la característica multicore dels processadors actuals.
- En l'algorisme genètic, prioritzar els descendents que aprenen més ràpid, sempre que es verifiqui la propietat 1. Pot ser necessari trobar un compromís entre mida de la xarxa i temps d'aprenentatge.
- Parar l'algorisme genètic abans d'esgotar el limit de temps, si es detecta que s'ha aconseguit una configuració òptima.
- Comparar els resultats de la implementació, amb altres algorismes d'aprenentatge computacional.

1.3 Mètode seguit i planificació

Es seguirà un desenvolupament top-down iteratiu, que permetrà provar i validar els algorismes des de la primera iteració, i anar afegint característiques progressivament. La planificació és la següent:

[9 març] Inici del projecte

[2 setmanes, 9 – 23 de març] Recopilar informació sobre xarxes neurals i algorismes genètics, que serveixi de guia en la implementació.

[2 setmanes, 23 de març – 6 d'abril] Iteració 0: Preparar un marc de treball, amb:

- Estructura de llibreria per a la xarxa neural.
- Estructura de llibreria per a l'algorisme genètic.

[3] Aquest factor determina el tipus de xarxa neural que s'implementarà: el perceptró multicapa, que és una variant general i ben coneguda.

[4] Evita l'*overfitting*

- Paquet R que cridi a les llibreries anteriors.
- Marc de proves des d'R amb les dades escollides.

[6 abril] Lliurament PAC2

[2 setmanes, 6 – 20 d'abril] Iteració 1: Augmentar el marc de treball anterior amb:

- Implementació de la xarxa neural.
- Validar implementació amb el marc de proves.

[2 setmanes, 20 d'abril – 4 de maig] Iteració 2: Augmentar el marc de treball anterior amb:

- Implementació de l'algorisme genètic, treballant sobre la xarxa neural.
- Validar implementació amb el marc de proves.

[4 maig] Lliurament PAC3

[1 setmana, 4 – 11 de maig] Proves: Comparar els resultats obtinguts amb l'ús de l'algorisme genètic i sense (iteracions 1, 2). Possible millora: comparar els resultats obtinguts amb altres algorismes d'aprenentatge automàtic.

[3 setmanes, 11 – 31 de maig] Completar treball

- Completar l'elaboració de la memòria del treball.
- Elaborar la presentació virtual.

[1 juny] Lliurament final

1.4 Sumari de productes obtinguts

Juntament amb aquesta memòria, es lliurarà un fitxer comprimit amb el següent contingut:

- El codi font C++ de la implementació de la xarxa neural i de l'algorisme genètic.
- El codi font C++ i R que integra la implementació amb el software estadístic R.
- Totes les dades usades en les proves, juntament amb els scripts R que executen les proves i generen les taules de resultats i gràfics inclosos en aquesta memòria.

1.5 Descripció de les properes seccions

En aquest text es documenta el desenvolupament, la fonamentació teòrica que avala les decisions de disseny i les proves realitzades. L'ordre de la memòria coincideix amb l'ordre del desenvolupament, fet que facilitarà la lectura i el seguiment del treball realitzat.

En la propera secció es presentarà el marc de treball, on es tracten decisions com el llenguatge de programació i el programari estadístic que s'han utilitzat, així com una descripció de la seva integració. També es presenten els conjunts de dades escollits per provar els algorismes i es proporciona un tutorial que permet una primera presa de contacte amb el programari.

A continuació, a les seccions 3 i 4 es descriuen les implementacions de la xarxa neural i l'algorisme genètic. Es justifiquen les decisions de disseny i les alternatives escollides, en base a diversos llibres i treballs de recerca, i es proporciona la fonamentació teòrica necessària per comprendre el seu funcionament i ús.

Per finalitzar el treball, la secció 5 presenta les proves realitzades i els resultats obtinguts. Les proves seran variades i aniran des de proves teòriques sobre petits conjunts de dades sintètiques, fins a aplicacions pràctiques amb datasets relativament grans. La secció 6 conté les conclusions del treball, i a continuació s'adjunta un glossari de termes i les referències, que tancaran el treball.

2 Marc de treball

En aquesta secció es descriuran les eines usades en l'elaboració del treball i els motius de la seva elecció. El programari desenvolupat s'integrarà en el software estadístic R, que facilita la feina d'importació de dades i visualització de resultats. Com a llenguatge de programació s'usarà C++ seguint unes normes d'estil prefixades. Al final de la secció, es presentarà una sessió d'exemple amb una petita demo d'ús del programari desenvolupat en el treball.

El hardware usat ha estat un PC *Intel i3* amb 8GiB de RAM per al desenvolupament, i un servidor *Intel Xeon* amb 32GiB de RAM per a determinades tasques que han requerit hores de càlcul. Totes dues màquines amb sistema operatiu GNU/Linux, distribució *ArchLinux*[5]. Com a entorn de desenvolupament s'ha utilitzat Eclipse[6] versió MARS.2 (4.5.2), amb el plug-in CDT versió 8.8.1 per desenvolupament C/C++.

2.1 Llenguatge C++

Per a la implementació, tant de la xarxa neural com de l'algorisme genètic, s'ha optat pel llenguatge C++ en la seva versió més recent. El codi escrit s'ajusta estrictament a l'estàndard del llenguatge. Els motius d'aquesta elecció són:

- Portabilitat: és un llenguatge estàndard (ISO/IEC 14882:2014) amb compiladors disponibles per a múltiples plataformes. La plataforma de desenvolupament és GNU/Linux amb el compilador g++ (GCC 5.3.0), però ha de ser possible compilar el codi a qualsevol combinació sistema/compilador que suporti l'esmentat estàndard.
- Eficiència: és un llenguatge compilat a codi natiu, amb compiladors capaços de generar codi optimitzat especialment per a cada processador. Proves empíriques han mostrat que un programa C++ s'executa més ràpid i requereix menys memòria que altres llenguatges com Java o Scala. Per a un mateix programa, s'han mostrat factors de 5.8 en temps d'execució i 3.7 en ús de memòria, de Java respecte a C++ [Hun11].
- Eines del llenguatge: Per a la implementació d'aquest algorisme serà útil disposar de les eines que proporciona la STL i la orientació a objectes. Aquestes eines no estan disponibles en llenguatges més senzills com C.

El motiu determinant per a l'elecció de C++ sobre altres llenguatges interpretats o compilats a *bytecode*, és principalment l'eficiència del codi natiu en velocitat i ús de memòria. En un algorisme com el que s'implementa en aquest treball, aquesta elecció implica poder crear xarxes neurals més grans i que es poden entrenar en un temps menor, segons els resultats empírics referits.

Estil del codi font

En el codi font d'aquest treball s'ha seguit un estil inspirat en la guia d'estil de Google[7] per a C++. Es destaquen els següents punts:

- Sufixos per unitats de compilació: .hpp per capçaleres, .cpp per implementacions.
- Identificadors de tipus (class, struct, enum, typedef) s'escriuen en notació CamelCase, excepte quan complementin la llibreria estàndard o el llenguatge (per exemple `uint`).
- Identificadors de mètodes i funcions s'escriuen en notació dromedaryCase.
- Variables locals i paràmetres d'una funció: identificadors curts, minúscules_amb_subratllats.
- Variables d'una classe: identificadors descriptius. Notació amb subratllats si són privats.
- Constants: MAJUSCULES_AMB_SUBRATLLATS.

[5] `http://www.archlinux.org/`
[6] `http://www.eclipse.org/`
[7] `https://google.github.io/styleguide/cppguide.html`

- Identificadors _amb_subratllat_inicial quan es pugui ocultar variables membre d'una classe. Aquesta notació s'usa en els paràmetres dels constructors, per exemple:
`Constructor(Tipus _var): var(_var) { ... }`

Organització del codi C++

Tant la xarxa neural com l'algorisme genètic s'organitzen en diverses classes, totes dins de l'espai de noms `genann`, acrònim de *genetic artificial neural network*. Tot el contingut d'aquest espai de noms es pot usar com a llibreria, des d'un altre programa. Tal com es descriurà a continuació, s'ha mantingut separat el codi C++ que permet la integració amb R, precisament per permetre aquesta funcionalitat alternativa.

A continuació es resumeixen les classes principals:

- `genann::Mlp` implementa un perceptró multicapa.
- `genann::MlpBp` estén la classe `Mlp` per proporcionar la funcionalitat de retropropagació clàssica. Es podrien estendre altres classes a partir de `Mlp` per proporcionar altres tipus de retropropagació, com Rprop o les seves variants.
- `genann::Trainer` és una classe abstracta que serveix de base a una o més estratègies d'entrenament de xarxes neurals.
- `genann::OnlineTrainer` estén la classe `Trainer` per proporcionar la estratègia d'entrenament online. Altres estratègies d'entrenament podrien ser BatchTrainer o MiniBatchTrainer.
- `genann::GenAlg` és la classe que implementa l'algorisme genètic. Aquesta classe es concentra en l'evolució i el manteniment de la població d'organismes.
- `genann::Organism` encapsula un organisme, mantenint un cromosoma o estan codificats els gens, i implementant les diverses operacions genètiques.

2.2 Integració amb el software estadístic R

R ha esdevingut en els darrers anys una eina d'anàlisi de dades molt popular en l'àmbit acadèmic i científic. Diverses enquestes[8] el situen ja per sobre d'altres paquets estadístics coneguts, com SPSS o SAS, i amb una tendència creixent. Un dels principals motius que s'argumenten en favor d'R és que aquest és codi obert, amb una comunitat científica que l'amplia i el millora constantment per a tasques d'anàlisi que van més enllà de l'estadística. És adient l'elecció d'R per a un treball com aquest, que és una eina per a l'anàlisi de dades i que serà distribuït també sota llicència de codi obert.

A més, R està construït sobre un llenguatge de programació interpretat, amb una interfície per desenvolupar extensions amb altres llenguatges compilats, especialment dissenyada per C i C++ [Rct16]. Així, el programari desenvolupat en aquest treball serà concebut com una extensió d'R, i usat des de sessions interactives o des d'scripts R. Aquesta facilitat permetrà reutilitzar totes les funcions d'R per entrada i sortida, tractament de dades, scripting i gràfics.

Estructura del paquet

Per crear un paquet R només cal seguir una estructura de directoris prefixada i incloure-hi uns fitxers amb noms i continguts determinats. El paquet tindrà el mateix nom que el directori arrel de l'estructura, i un cop finalitzat es pot comprimir amb tot el seu contingut per facilitar la distribució. A continuació es detalla aquesta estructura:

[8] Els articles que les citen estan disponibles a:
`http://www.nature.com/news/programming-tools-adventures-with-r-1.16609`
`http://r4stats.com/articles/popularity/`

```
genann
    man
        ann.default.Rd
        gann.default.Rd
    R
        ann.R
        ann_plot.R
        ann_predict.R
        gann.R
        gann_plot.R
        ...
    src
        activators.hpp
        ga.hpp
        ga.cpp
        ga_organism.hpp
        ga_organism.cpp
        ...
        rentry.cpp
    COPYING
    DESCRIPTION
    NAMESPACE
```

El nom del paquet d'aquest treball es diu *genann*, acrònim de *genetic artificial neural network*. El directori `man` conté la documentació en forma de pàgines de manual. El directori `R` conté les funcions R i aquestes, usant la interfície `.Call`, cridaran a les funcions C++ `r_ann_train`, `r_ann_predict` i `r_gann` que són els punt d'entrada a la llibreria C++.

El codi C++ està en el directori `src` i es compilarà automàticament quan s'instal·li el paquet. D'aquesta manera es garanteix la portabilitat, i és la manera estàndard de crear paquets amb R [Rct16]. Perquè aquest procediment sigui possible, cal que el sistema operatiu tingui instal·lat un compilador de C++ i que R el conegui. Aquesta configuració pot variar entre diferents sistemes operatius; en una distribució de GNU/Linux només cal instal·lar R i GCC. El codi font d'aquest treball requereix que el compilador compleixi com a mínim l'estàndard C++11.

Per instal·lar el paquet, cal arrancar R des d'el directori que conté el paquet `genann`. Llavors es pot instal·lar:

```
install.packages("genann", repos=NULL)         # directament del directori
install.packages("genann.tar.gz", repos=NULL)  # o des d'el fitxer comprimit
```

A continuació, en les sessions R o scripts on s'hagi d'usar el paquet, es carregarà amb:

```
library("genann")
```

Interfície amb C++

La interfície entre R i C++ es construeix en dues parts. En la part externa es defineixen funcions i mètodes R que es guarden en el directori `genann/R`. Aquestes funcions poden cridar, quan sigui necessari, a funcions C o C++ amb la facilitat `.Call`. La part interna de la interfície es composa de les funcions C o C++ que són cridades per `.Call`, i que han de tenir una signatura específica com la del següent exemple:

```
extern "C" {
SEXP nom_funcio(const SEXP p_1, ... const SEXP p_n) {
    ...
    SEXP r = ...;
    return r;
}}
```

En aquest treball, seguint l'estratègia de minimitzar les dependències, les funcions C++ que són cridades amb la facilitat `.Call` s'han encapsulat en la unitat de compilació `rentry`. La resta de codi C++ està lliure de R, i únicament depèn de les llibreries estàndard de C++, facilitant així el seu possible ús integrat en altres programes.

Nombres aleatoris

R disposa del seu propi generador de nombres aleatoris, però des d'el codi C++ s'utilitza el generador de la llibreria estàndard. És habitual i útil usar la funció `set.seed` en alguns scripts R, quan es vol repetir la seqüència de números aleatoris en diferents execucions. Això permet per exemple tornar a crear els mateixos models de xarxa neural, o repetir una evolució de l'algorisme genètic.

Perquè la implementació disposi d'aquesta funcionalitat, en totes les crides a la facilitat `.Call` d'R, es passa com a darrer paràmetre un enter aleatori generat a R, que s'usarà com a llavor del generador de números aleatoris de la llibreria estàndard de C++. D'aquesta manera es propaga l'efecte del `set.seed` de R, cap al codi escrit en C++.

Documentació

La documentació de les funcions R es guarda en el directori `man`, un fitxer per cada pàgina de documentació en format propi similar a LaTeX. Quan es carrega el paquet a R, es llegeixen automàticament tots els fitxers d'aquest directori i s'integra la documentació. Per accedir a les pàgines de documentació des d'R, executar:

```
?ann   # documentació de la xarxa neural
?gann  # documentació de l'algorisme genètic
```

Usabilitat

L'usuari esperat d'aquest programari té un perfil avançat, amb coneixements de programació i amb inclinació científica, que probablement ja coneixerà el software R. Per això s'ha prioritzat usar les notacions i pràctiques habituals d'R, que també segueixen altres paquets d'aprenentatge computacional i que són ben coneguts per la comunitat d'usuaris d'R. En concret:

- S'han creat comandes directament executables des d'R, de manera que es poden usar en una sessió interactiva o en un procés automàtic en un script.
- Les funcions que generen un model, el retornen en forma d'objecte R, que posteriorment pot: serialitzar-se amb save(), guardar-se en la sessió actual, usar-se en mètodes R que han estat sobrecarregats, com print(), summary(), plot() o predict().
- S'ha usat la tècnica habitual a R d'assignar valors per defecte a tots els paràmetres possibles d'una funció, deixant que l'usuari configuri només aquells que consideri necessari, i permetent que la crida a les funcions sigui el més curta possible.
- S'ha proporcionat una breu documentació de les funcions, en el propi sistema d'ajuda d'R.

2.3 Conjunts de dades

En aquest treball no s'està realitzant cap estudi sobre un conjunt de dades en particular. Al contrari, el que interessa és estudiar un algorisme i s'usen els conjunts de dades com a eina de prova. En conseqüència, s'han escollit conjunts de dades coneguts i no es realitza una anàlisi rigorosa de les dades, sinó que es comparen els resultats obtinguts amb els d'altres treballs que usen les mateixes dades.

Problema	Entrades				Sortides		Exemples	Entropia
	b	c	m	total	b	c		
building	8	6	0	14	—	3	4208	—
flare	22	2	0	24	—	3	1066	—
card	40	6	5	51	2	—	690	0.99
heart	18	6	11	35	2	—	920	0.99
thyroid	9	6	6	21	3	—	7200	0.45

Taula 1: Característiques dels datasets de PROBEN1 escollits. Les columnes b, c indiquen valors binaris o continus, la columna m correspon a entrades afegides per indicar que l'exemple conté valors mancants. Com es pot deduir per les sortides, els dos primers són regressions i els tres últims classificacions.

Dades sintètiques

Les dades sintètiques es generen directament a R a partir de funcions lògiques i matemàtiques. En el tutorial de la propera secció 2.4 es crea un d'aquests conjunts i s'afegeix soroll aleatòri. Les dades sintètiques dels conjunts per proves es construeixen de forma anàloga en els scripts R que es lliuren en el producte.

En concret, s'han generat:

- Les funcions lògiques AND i XOR, per verificar que l'AG és capaç de trobar una xarxa neural que resol el problema clàssic XOR.
- Una funció contínua $y = \sin(x)$ que segons el teorema de Cybenko s'ha de poder aproximar amb una única capa oculta.
- Una funció discontinua $y = 1/x$, que requereix més d'una capa oculta per a la seva aproximació.
- Una funció soroll, per a la qual l'algorisme genètic no ha de ser capaç d'evolucionar cap xarxa neural que l'aproximi.
- Una funció trigonomètrica addicional $y = x\sin(5x) + \varepsilon$, $x \in [0, \pi]$, amb soroll i amb entrades i sortides fora dels rangs recomanats per a xarxes neurals, per posar a prova l'AG amb xarxes saturades.

PROBEN1

En el treball [Pre94], Lutz Prechelt va recopilar diversos datasets de dominis diferents del repositori UCI [Lic13], que inclouen problemes de classificació i d'aproximació o regressió, amb l'objectiu de que fossin usats en altres treballs sobre xarxes neurals com a eina de benchmark. Per això els va preparar, normalitzant totes les dades en un format adient per a xarxes neurals i va publicar el treball amb el nom de PROBEN1. Aquesta publicació inclou a més les instruccions per usar els datasets i fer les comparacions correctament.

De cada un dels problemes de PROBEN1 es proporcionen 3 permutacions, cada una de les quals es parteix en 3 conjunts: entrenament, validació i proves. Així, les tres permutacions permeten avaluar diferents dades d'entrenament provinents del mateix dataset, formant una validació creuada. Per exemple, del problema *building*, es creen tres datasets diferents *building1*, *building2* i *building3*.

Dels 15 problemes disponibles a PROBEN1, en aquest treball se n'ha seleccionat 5, 2 problemes de regressió i 3 de classificació. A continuació es descriuen breument i a la taula 1 es resumeixen les seves característiques.

- **Building:** Predicció del consum d'energia en un edifici. Les entrades corresponen a factors ambientals, i les sortides pretenen predir els consums elèctric, d'aigua calenta i freda d'un

edifici. Les dades provenen del concurs *The Great Energy Predictor Shootout – the first building data analysis and prediction problem*, de 1993 per l'*ASHRAE meeting*, a Denver, Colorado.

- **Flare:** Predicció d'erupcions solars. Amb aquestes dades s'intenta predir el nombre d'erupcions solars petites, mitjanes i grans, que es produiran en les properes 24 hores en una regió de la superfície solar. Les entrades corresponen a l'activitat històrica de la regió. Les dades provenen del problema *solar flare* d'UCI.
- **Card:** Prediccions sobre l'aprovació o denegació d'una targeta de crèdit. Cada exemple és una sol·licitud real, les entrades són diverses variables del sol·licitant, i les sortides corresponen a l'aprovació o la denegació de la targeta. Aquest dataset està creat a partir de dades el problema *Credit screening* d'UCI.
- **Heart:** Dades mèdiques amb les que es pretén construir un classificador capaç de predir una malaltia cardíaca. Cada exemple correspon a un pacient, les entrades corresponen a diverses mesures i hàbits de vida. Els exemples procedeixen de quatre datasets i corresponen al problema *heart disease* d'UCI. L'ús d'aquestes dades en qualsevol treball requereix citar el nom de les institucions i persones que les varen recopilar:
  1. Hungarian Institute of Cardiology, Budapest; Andras Janosi, M.D.
  2. University Hospital, Zurich, Switzerland; William Steinbrunn, M.D.
  3. University Hospital, Basel, Switzerland; Matthias Pfisterer, M.D.
  4. V.A. Medical Center, Long Beach & Cleveland Clinic Foundation; Robert Detrano, M.D., Ph.D.
- **Thyroid:** Dades per a la diagnosi d'hiper o hipotiroidisme. Les entrades corresponen a l'examen d'un pacient, i les tres sortides corresponen a la classificació d'hipertiroidisme, hipotiroidisme o funció normal. Les dades provenen del problema *thyroid disease* d'UCI.

MNIST

El conjunt de dades MNIST ha estat preparat per Yann LeCun (Courant Institute, NYU), Corinna Cortes (Google Labs) i Christopher J.C. Burges (Microsoft Research), a partir d'una base de dades anterior i més gran de dígits escrits a mà del NIST (National Institute of Standards and Technology). A MNIST els dígits han estat centrats i redimensionats, de manera que siguin tractables per algorismes d'aprenentatge automàtic. El dataset consta de 60k exempes d'entrenament, i 10k exemples per validació.

Existeixen nombrosos treballs realitzats amb aquestes dades, els resultats de molts dels quals es presenten a [Lec99] i permeten fer comparacions de rendiment entre algorismes. Al llibre [Rus10] també es parla d'aquest dataset i es proporciona una taula amb comparatives dels models obtinguts amb diferents algorismes, mesurant diverses mètriques com qualitat, temps d'execució o mida del model.

2.4 Sessió d'exemple

A continuació es presenta un breu exemple en forma de tutorial, de com usar la implementació. Cal començar iniciant R en mode interactiu. Un cop a R, cal instal·lar i carregar el paquet *genann*. Si no s'ha executat R en el directori amb el paquet, pot usar-se la ruta completa a `install.packages`:

```
install.packages("genann.tar.gz", repos=NULL)
library("genann")
```

Com que l'objectiu és veure com funciona el paquet `genann`, es generarà un conjunt de dades sintètiques molt simple, de manera que els algorismes s'executin ràpid:

```
set.seed(2016)  # intentar produir el mateix resultat, pot dependre
                # del generador de nombres aleatoris del sistema

# generar un dataset sintètic amb entrada propera a 0 i sortida en [0, 1]
data <- data.frame(x=seq(-2.5, 2.5, by=0.02))
data$y <- exp(-data$x^2) + rnorm(length(data$x), sd=0.03)

# desordenar les files aleatoriament
data <- data[sample(nrow(data)),]

# partir en subconjunts d'entrenament i prova
data_train <- data[1:175,]      # 70% aprenentatge AG+XN
data_test  <- data[176:251,]    # 30% validació del model de XN

# partir subconjunt d'entrenament en 2 per l'AG
data_gt <- data_train[1:87,]    # 50% per entrenament de XN en AG
data_gv <- data_train[88:175,]  # 50% per validació de XN en AG
```

Ara es vol usar el conjunt `data_train` per trobar un model basat en una xarxa neural que permeti fer prediccions. El conjunt `data_test` s'usarà per comprovar que el model prediu dades correctament, per tant no es pot usar durant l'entrenament de la XN. Per trobar una XN adequada, cal respondre a les següents preguntes:

- Quantes capes ha de tenir la XN?
- Quantes neurones ha de tenir cada capa?
- Quin valor ha de tenir el factor d'aprenentatge?
- Afegir un terme de momentum accelerarà l'aprenentatge? Si es així, quin valor ha de tenir?
- Fins a quin punt es pot minimitzar l'error de la XN sense produir overfitting?

Respondre aquestes preguntes requereix experimentar manualment o automàticament amb diferents configuracions [Ben12, secció 3.3]. En aquest treball s'ha optat per l'experimentació automàtica guiada per un algorisme genètic. Per executar-lo:

```
evol <- gann(data_gt$x, data_gt$y, data_gv$x, data_gv$y, lact="sigmoid")
```

Amb aquest problema, l'algorisme genètic tarda uns minuts en executar-se, però un cop finalitzat respon totes les preguntes. L'AG finalitza amb una sortida com la següent, on es mostren per files els organismes de la població final, amb la configuració de cada un d'ells i la puntuació (o *fitness*) que ha obtingut desglossada en els seus tres termes:

```
FINAL POPULATION

 fitness      mse      fnp  np      fne  ne | ann config
0.002412 0.002130 0.000231 231 0.000050  50 | tmse=0.001953, lr=0.878370, mo=0.502525, hls=c(16, 11)
0.003150 0.002761 0.000345 345 0.000044  44 | tmse=0.002274, lr=0.816571, mo=0.533410, hls=c(10, 27)
0.003500 0.003191 0.000249 249 0.000059  60 | tmse=0.002075, lr=0.816571, mo=0.663294, hls=c(10, 19)
0.003750 0.003415 0.000229 229 0.000106 106 | tmse=0.001968, lr=0.818769, mo=0.659266, hls=c(6, 27)
0.003941 0.003671 0.000231 231 0.000039  39 | tmse=0.001953, lr=0.877394, mo=0.631434, hls=c(16, 11)
0.004533 0.004077 0.000389 389 0.000067  67 | tmse=0.001953, lr=0.885817, mo=0.502510, hls=c(12, 26)
...
```

La columna `mse` és l'error quadràtic mig obtingut amb el conjunt de validació, la columna `np` és la mida de la XN, i.e. el nombre de paràmetres (pesos sinàptics i biaixos), la columna `ne` el nombre d'epochs que han estat necessaris en l'entrenament, les columnes `fnp` i `fne` són tots dos valors multiplicats per factors configurables. La columna `fitness` és la suma `mse + fnp + fne`.

Pot escollir-se un dels millors organismes en base a preferències diverses. Encara que en aquest exemple tots els organismes són semblants, segons el dataset el fitness serà una funció multimodal, i pot tenir diversos òptims similars. En conseqüència, diferents execucions de l'AG o inclús una sola, poden retornar configuracions de XN molt diferents amb valors de fitness aproximadament iguals. En aquests casos cal escollir un organisme, no és correcte fer una mitja dels hiperparàmetres dels millors. Veure la secció 4.4 per més informació d'aquest tema.

La columna titulada `ann config` dona la configuració corresponent a la XN, i pot copiar-se tal qual en la crida a la funció `ann()`, que entrena una XN i retorna el model. Alternativament, es pot usar el resultat de l'evolució de l'AG guardat en l'objecte `evol` per recuperar els hiperparàmetres del millor organisme, tal com es fa a continuació:

```
model <- ann(data_train$x, data_train$y, lact="sigmoid",
             tmse=evol$besttmse, lr=evol$bestlr, mo=evol$bestmo, hls=evol$besthls)
```

Finalment, amb el model creat, es poden fer prediccions i comprovar que, en efecte, l'AG ha retornat una configuració de XN vàlida per aquest conjunt de dades:

```
fitted <- predict(model, newdata=data_test$x)  # realitzar prediccions
errors <- data_test$y - fitted                 # residus
print(sum(errors^2) / nrow(errors))            # error quadràtic mig
```

Veiem que l'EQM amb el conjunt de test és 0.0025, molt proper al que indica l'algorisme genètic, que ha treballat amb un conjunt de validació diferent del de test. Amb la següent instrucció es poden veure algunes prediccions:

```
print(head(data.frame(x=data_test$x, y=data_test$y, y_estimat=fitted)))
```

```
      x          y  y_estimat
1  0.52 0.75504427 0.75141193
2  1.64 0.07194686 0.05002992
3 -0.92 0.41788123 0.48948736
4  2.14 0.00031114 0.02124688
5  1.00 0.36336024 0.32373553
6  1.72 0.07264442 0.04107427
```

Finalment, es poden obtenir els gràfics que es mostren a les figures 1, 2 i 3:

```
plot(evol)                        # evolució de l'AG
plot(model)                       # visualització de la XN
plot(data_train$x, data_train$y)  # dades d'entrenament
plot(data_test$x, fitted)         # dades ajustades
plot.ts(model$errors, log="y")    # aprenentage de la XN
```

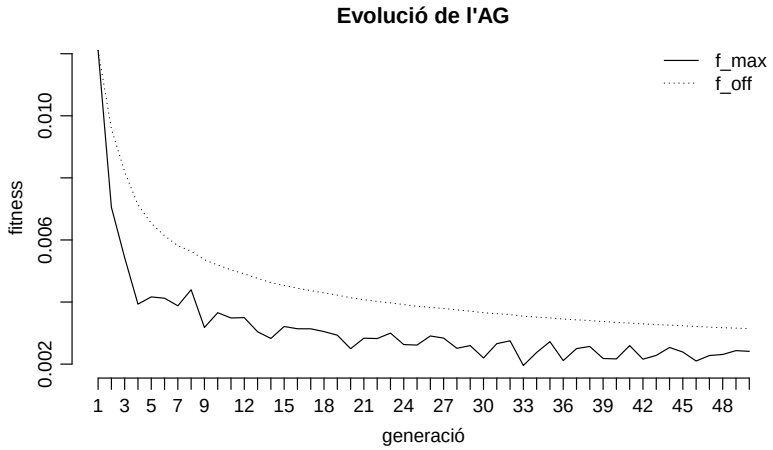

Figura 1: Evolució de l'AG del tutorial. A la secció 4.4 es descriuen les mètriques $f_{\max}$, f_{off} i el criteri de parada usat en l'AG.

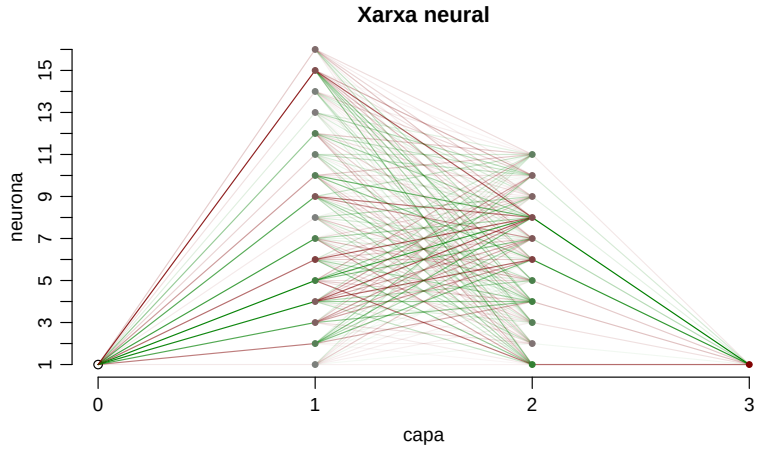

Figura 2: Representació gràfica de la xarxa neural del tutorial. Les línies representen els pesos sinàptics i els punts les neurones amb el seu biaix. El color vermell són números negatius i el verd positius, la saturació de color indica la magnitud.

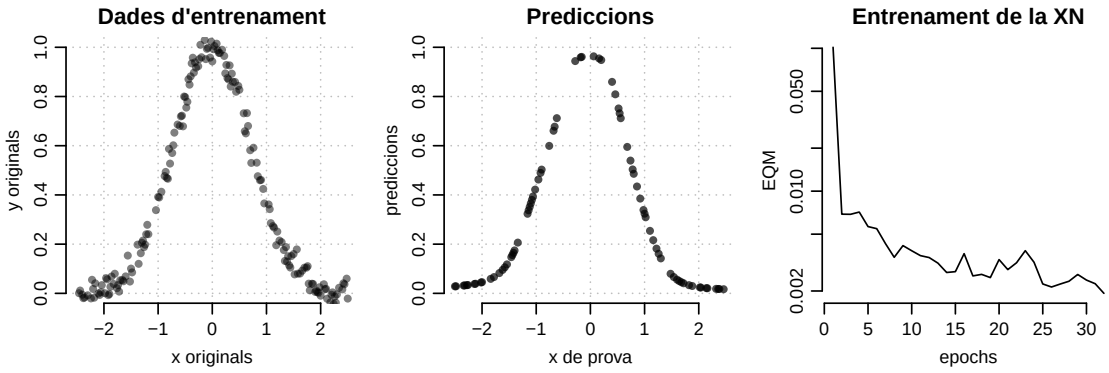

Figura 3: Els dos primers gràfics són les dades originals i les prediccions. El tercer mostra com canvia l'EQM de la XN durant el seu entrenament.

3 Implementació de la xarxa neural

En aquesta secció es tractarà la implementació de la xarxa neural i la seva fonamentació teòrica. S'inicia la secció amb una breu introducció als models basats en xarxa neural, que serviran per conèixer les limitacions i l'ús de la implementació. A continuació es detalla el funcionament de la xarxa neural i es justifiquen les decisions de disseny. Finalment es realitzen diverses proves per comprovar que la implementació té el rendiment esperat.

3.1 Models amb xarxes neurals

Es disposa d'un *conjunt de dades d'entrenament*, cada element del qual es diu *instància* o *exemple*[9]. Una instància consta de diverses *variables explicatives* o *entrades*, i diverses *variables resposta* o *sortides*. No poden haver-hi dades mancants, si existeixen s'han de tractar prèviament. Es vol aconseguir un model matemàtic que sigui capaç d'interpolar[10] valors de variables resposta, a partir de valors de variables explicatives que no necessàriament seran els mateixos que els existents en els exemples. És a dir, es vol un model matemàtic que *aprengui* i *generalitzi* el conjunt de dades.

Aquest procediment es diu *aprenentatge supervisat* perquè es disposa d'informació sobre la resposta que ha de donar el model, per a cada entrada existent en els exemples. Com que interessa aconseguir el millor model possible, el procés d'aprenentatge és en realitat una optimització, i un problema plantejat així es diu d'*aprenentatge inductiu* [Tor13]. Formalitzar aquest aprenentatge és equivalent al formalisme que s'utilitza en regressió estadística. De fet el problema és equivalent amb la diferència que es canvia el mètode de mínims quadrats per la xarxa neural.

Cada instància del conjunt d'entrenament és un parell (x, y), amb $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, i $y = (y_1, \ldots, y_m) \in \mathbb{R}^m$. El vector x és la part explicativa i el vector y la resposta. Es suposa que les variables resposta s'obtenen de les explicatives per un fenomen que s'explica amb una funció $y = f(x|W) + \epsilon$, on W és un conjunt de paràmetres que a priori són desconeguts, i el valor ϵ apareix perquè no es pot descartar que no existeixi una component aleatòria en les dades. Si $\epsilon = 0$ llavors el fenomen és determinista i el model que s'obtindrà serà exacte. Si $\epsilon > 0$ llavors el fenomen és estocàstic, i la precisió del model dependrà de la component aleatòria present en les dades.

Així, l'objectiu de l'aprenentatge inductiu és trobar el millor model possible donant valor als paràmetres W. Es disposa de diversos algorismes per assolir aquest objectiu. Les xarxes neurals en son un, i en les properes seccions es descriurà com aproximen la funció f, i quin procediment segueixen per obtenir el valor dels paràmetres W.

Igual que els models estadístics de regressió, un cop construït el model $f(x|W)$ a partir de les dades d'entrenament, cal validar que efectivament serveix com a model predictiu. A continuació s'avancen les possibilitats existents:

- L'error del model és baix tant amb dades d'entrenament com amb dades noves. Per tant, el model és correcte. Si a més l'error és 0 o molt proper, podria tractar-se d'un fenomen determinista que el model explica amb exactitud.
- L'error del model és baix amb les dades d'entrenament, però alt amb dades noves. En aquest cas el model ha memoritzat els exemples i no té capacitat per interpolar dades desconegudes. Es diu que hi ha *error de generalització* o *overfitting*.
- L'error és alt amb les dades d'entrenament. Significa que el model no s'ha ajustat, potser perquè la xarxa neural no té la configuració adient o el temps necessari d'aprenentatge, o perquè en efecte no existeix cap fenomen que expliqui les y com a funció de les x.

[9] S'usaran indistintament els noms "exemple", "instància" o "individu" per denotar una *fila*; i els noms "atribut" o "variable" per denotar una *columna* de la taula de dades.

[10] Existeixen dos tipus diferents de generalització: *interpolació* i *extrapolació*. La interpolació s'aplica entre exemples propers. En altres casos es tracta d'extrapolació, per exemple fora del rang del conjunt de dades, o dins de forats grans entre instàncies. En xarxes neurals, la extrapolació no és fiable, en canvi la interpolació serà fiable si les dades d'entrenament són una mostra representativa del fenomen a modelar [Sar02, FAQ3: How is generalization possible?].

Cal notar que com les xarxes neurals són aproximadors universals, al contrari que els mínims quadrats, amb prou temps i una configuració adient, sempre es podrà trobar el model $y = f(x|W)$. A menys que x i y siguin independents.

Avaluació de models basats en xarxa neural

L'avaluació de models és una àrea extensa i no és l'objectiu d'aquest treball tractar-la, no obstant, cal enumerar les tècniques i mètriques que s'usaran. És habitual partir el conjunt de dades original en dos, un per a l'entrenament de la xarxa neural i l'altre per validar el model que s'anomenarà conjunt de test, normalment usant 2/3 de les dades per entrenament i 1/3 per test [Mol10]. Durant l'entrenament de la XN, no pot usar-se cap exemple del conjunt de test.

Per quantificar la qualitat del model, tant durant l'entrenament com en la validació, s'utilitza una mesura de l'error que comet, normalment l'error quadràtic mig o EQM, que a l'igual que a regressió estadística, es basa en la distància entre els valors esperats i les prediccions realitzades pel model. L'EQM calculat amb el conjunt d'entrenament es diu *error d'entrenament*, mentre que l'EQM calculat amb el conjunt de test es diu *error de generalització*. Un error d'entrenament baix indica que el model ha après correctament les dades, un error de generalització baix indica que el model té capacitat d'interpolar dades desconegudes.

Quan es té un error d'entrenament baix i un error de generalització alt, es diu que s'ha produït *overfitting*, perquè el model ha apres els exemples d'entrenament però no té capacitat d'interpolar dades desconegudes. L'objectiu és minimitzar tots dos errors, però és important destacar que a partir d'algun moment de l'entrenament que no es coneix a priori, l'error d'entrenament i l'error de generalització es converteixen en objectius contradictoris [Roj96], i millorar l'error d'entrenament empitjorarà l'error de generalització produint overfitting.

Normalització de dades per a xarxes neurals

Les xarxes neurals limiten els rangs de les variables explicatives i resposta. Cal usar estandarització per a les variables explicatives, i normalització per a les variables resposta. La normalització de les respostes dependrà de la funció d'activació que s'utilitzi en l'última capa, i serà [0, 1] per la sigmoide o [-1, 1] per la tangent hiperbòlica, sempre evitant que molts valors estiguin en els extrems de l'interval per evitar problemes de saturació.

La *saturació* es produeix quan molts valors d'una variable resposta estan a l'extrem del rang de la funció d'activació. En aquests casos, existeix la possibilitat de que els pesos de la XN creixin excessivament produint la saturació de la XN, degut als límits dels números en coma flotant. Cal evitar aquest problema especialment en problemes de classificació, canviant els valors de les variables resposta per sortides com {0.1, 0.9}, o {-0.9, 0.9} segons la funció d'activació.

A continuació s'enumeren els mètodes d'estandaritazció o normalització més habituals. Siguin D el domini d'una variable, v el seu valor, i μ_D, σ_D, $\min_D$, $\max_D$ la mitja, desviació típica, mínim i màxim dels valors. Es defineixen les següents normalitzacions:

- L'*estandarització* centra i redueix els valors de manera que la mitja sigui 0 i la desviació típica 1:
$$v' = \frac{v - \mu_D}{\sigma_D}$$
- La *normalització per la diferència* o *ranging* normalitza els valors de manera que quedin fitats en l'interval $[0, 1]$, transformant $v \in [\min_D, \max_D]$ en $v' \in [0, 1]$:
$$v' = \frac{v - \min_D}{\max_D - \min_D}$$

- La *normalització pel màxim* és similar a l'anterior, però conserva el mínim en el valor 0, és a dir, transforma $v \in [0, \max_D]$ en $v' \in [0, 1]$.

$$v' = \frac{v}{\max_D}$$

- La *normalització [-1,1]* transforma l'interval $[\min_D, \max_D]$ en $[-1, 1]$:

$$v' = \frac{v - (\max_D + \min_D)/2}{(\max_D - \min_D)/2}$$

Les variables categòriques es poden convertir a binàries segons siguin nominals o ordinals. Per a les nominals es creen tantes noves variables binàries com categories. Per a les ordinals, si es tenen n categories, llavors es creen $n - 1$ noves variables binàries utilitzant codificació additiva.

valor original	cod. nominal				cod. additiva		
A	A_1	A_2	A_3	A_4	A_2	A_3	A_4
a_1	1	0	0	0	0	0	0
a_2	0	1	0	0	1	0	0
a_3	0	0	1	0	1	1	0
a_4	0	0	0	1	1	1	1

Taula 2: Conversió de variables categòriques a binàries, usant codificació nominal o additiva per a variables ordinals.

Problemes de classificació

Quan les variables resposta són categòriques, la regressió s'anomena *classificació*, ja que es pot interpretar com que a cada x li correspon una categoria o classe. La xarxa neural no tracta dades categòriques, i per això cal transformar les variables resposta en numèriques, normalment binàries, mitjançant codificació nominal o additiva segons cada cas, tal com s'ha indicat abans.

En els problemes de classificació, a més de l'EQM, per mesurar l'error s'usen mètriques addicionals calculades a partir d'una matriu de confusió. En la secció 5.4 s'usaran algunes d'aquestes mètriques en un conegut problema de classificació de dígits anomenat MNIST. Cal destacar que aquestes mètriques tenen sentit en la interpretació del model. Per a l'entrenament de la xarxa neural es pot seguir usant l'EQM com a mesura d'error, independentment de que el problema sigui de regressió o de classificació.

3.2 Neurona artificial

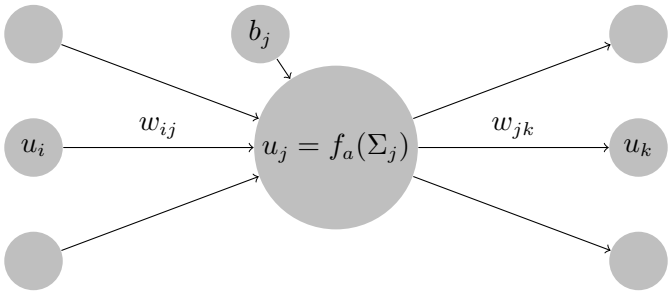

Figura 4: Esquema del funcionament de la neurona u_j en un perceptró multicapa. Σ_j denota la suma ponderada dels valors de les neurones de la capa precedent u_i, pels pesos de cada connexió sinàptica entrant w_{ij}, més el biaix b_j.

Ja s'ha comentat en les seccions precedents que les xarxes neurals es fonamenten en una unitat de càlcul que s'anomena *neurona artificial* o *perceptró*. Aquesta unitat es forma d'unes entrades i genera una sortida. En les XN una neurona no actua sola, sinó connectada amb altres, i així les entrades a una neurona es corresponen amb les sortides d'unes altres. La figura 4 mostra un esquema del funcionament.

L'entrada de la neurona u_j es calcula com

$$\Sigma_j = \left(\sum_{\forall i} u_i w_{ij} \right) + b_j$$

Es tracta d'una funció lineal sobre les entrades u_i, on el biaix és l'ordenada a l'origen[11]. Els valors w_{ij} s'anomenen *pesos sinàptics* i al estar multiplicant a les entrades, permeten regular cada connexió. Un pes $w_{ij} = 0$ implica que no hi ha connexió entre les neurones u_i i u_j, valors grans impliquen més vinculació que valors petits. És en els pesos sinàptics on es guarda l'aprenentatge, i són aquests valors (juntament amb els valors de biaix), els que formen els paràmetres W que cal trobar per construir el model.

Sobre l'entrada s'aplica una funció d'activació f_a, el resultat de la qual és la sortida de la neurona

$$u_j = f_a(\Sigma_j)$$

Funcions d'activació

Com a funcions d'activació, s'utilitzen funcions derivables amb forma logística o *sigmoides* com les següents:

$$\mathrm{sgm}(x) = \frac{1}{1 + e^{-x}}$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

La derivada de totes dues es pot expressar en termes de la funció original

$$\mathrm{sgm}'(x) = \mathrm{sgm}(x)(1 - \mathrm{sgm}(x))$$

$$\tanh'(x) = 1 - \tanh(x)^2$$

fet que permet dues optimitzacions:

- Per una banda no s'utilitza l'expressió de la derivada sinó l'anterior, que és més ràpida de calcular.

[11] Punt de tall amb l'eix y en les funcions lineals d'una variable.

- Per altra banda, no cal guardar el valor de l'entrada neta Σ_j de cada neurona, sinó únicament la seva sortida $f_a(\Sigma_j)$, en la qual ja s'ha aplicat la funció d'activació.

Les funcions d'activació com $\tanh$ que són simètriques respecte de l'origen són preferibles per a les capes ocultes [Roj96, Lec98], perquè produeixen sortides que en promig estaran centrades en zero, al contrari que la funció sgm, que generarà sortides sempre positives i per tant, la seva mitja serà sempre positiva.

Per a la capa de sortida es pot fer servir qualsevol de les dues anteriors, segons la sortida esperada de la XN. També s'acostuma a usar la funció identitat $f_a(x) = x$, la derivada de la qual és 1, i que és útil quan les sortides de la XN poden prendre qualsevol valor $y_k \in \mathbb{R}$, encara que en la pràctica, i degut a les limitacions dels números en coma flotant, es produirà saturació en la XN si les entrades o les sortides s'allunyen massa de l'interval $[-1, 1]$. També pot ser útil afegir un terme lineal petit per evitar la saturació de la XN, quan les sortides poden prendre valors en els extrems de l'interval del recorregut de la funció, aquest terme s'anomena *twisting term* [Lec98]. A continuació es llisten les funcions d'activació que s'han implementat i les seves propietats:

Activació	$f_a(x)$	$f'_a(x)$	Sortida	Entrada
Lineal	x	1	$\mathbb{R}$	$\mathbb{R}$
Sigmoide	$\mathrm{sgm}(x)$	$\mathrm{sgm}(x)(1-\mathrm{sgm}(x))$	$(0,1)$	$\mathbb{R}$
Tanh	$\tanh(x)$	$1-\tanh(x)^2$	$(-1,1)$	$\mathbb{R}$
Sigmoide+tt	$\mathrm{sgm}(x)+\alpha x$	$\mathrm{sgm}(x)(1-\mathrm{sgm}(x))+\alpha$	$[0,1]$	$\mathbb{R}$
Tanh+tt	$\tanh(x)+\alpha x$	$1-\tanh(x)^2+\alpha$	$[-1,1]$	$\mathbb{R}$

Taula 3: Funcions d'activació disponibles per a la capa de sortida amb les seves propietats. La sortida s'obté del recorregut de la funció, i l'entrada es correspon amb el domini. El twisting term α es un real petit. El conjunt $\mathbb{R}$ indica el recorregut teòric, en la pràctica, i degut a les limitacions de la coma flotant, no convé allunyar-se de l'interval $[-1, 1]$.

3.3 Perceptró multicapa

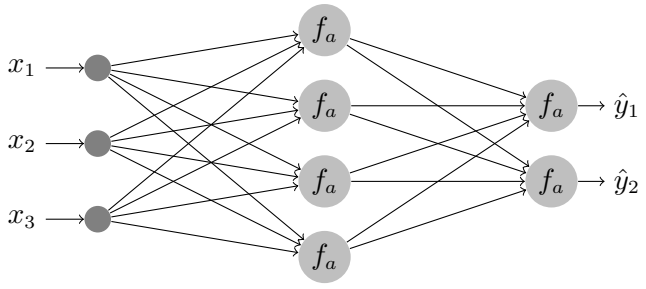

Figura 5: Esquema d'una xarxa neural de tipus perceptró multicapa, amb tres entrades, una capa oculta de 4 neurones, i una capa de sortida de 3 neurones.

Un perceptró multicapa[12] es forma de neurones o *unitats* que es denoten per u, i s'organitzen en capes, formant un graf dirigit acíclic com el que mostra la figura 5. En aquest treball, es denotará per u_i a les neurones de la capa precedent, u_j les neurones de la capa actual, i u_k les neurones de la capa següent. Els subindexos i, j, k s'utilitzaran així per denotar capes relatives, també per a altres components com per exemple el biaix de la capa actual b_j. La capa de sortida sempre utilitzarà el subindex k. Aquesta mateixa notació s'estén al codi C++.

Les capes reben diversos noms segons la seva posició: la capa d'entrada no és una capa en sí, ja que no té neurones. La *capa de sortida* són les neurones finals, les capes precedents a la sortida es

[12] En anglès *multilayer perceptron (MLP)*. Quan es formen de múltiples capes ocultes també reben noms com *deep neural networks* i el seu aprenentatge es diu *deep learning*.

diuen *capes ocultes*. Per denotar la topologia d'un perceptró multicapa s'utilitza la notació I-H-...-H-O, on I és el nombre d'entrades, les H són el nombre de neurones de les capes ocultes, i O és el nombre de neurones de la capa de sortida. Per exemple, 5-16-10-2 és una xarxa de 3 capes, amb 5 entrades, dues capes ocultes de 16 i 10 neurones, i una capa de sortida de 2 neurones. Aquesta nomenclatura evita confusions [Pre94] i es seguirà al llarg d'aquest treball.

Com que cada capa està completament connectada amb la seva capa precedent, es tindrà una matriu de pesos W_{ij} prèvia a cada capa. En el codi font, aquesta matriu té tantes files com neurones la capa a la que pertany, i tantes columnes com entrades (o equivalentment, com neurones de la capa precedent).

Per avaluar una entrada $x = (x_1, \ldots, x_n)$, s'apliquen les equacions de cada neurona, des de la primera capa fins a l'última. Els valors de les neurones de l'última capa es correspondran amb les sortides $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_m)$. Es tracta d'un procés iteratiu que s'anomena *propagació* o *feedforward*:

$$\hat{y}_k = u_k = f_a(\sum_{\forall j} u_j w_{jk} + b_k), \quad u_j = f_a(\sum_{\forall i} u_i w_{ij} + b_j), \quad u_i = \ldots$$

que es pot expressar en línia com una superposició finita de funcions d'activació:

$$\hat{y}_k = f_a(\sum_{\forall j} f_a(\sum_{\forall i}(\ldots) w_{ij} + b_j) w_{jk} + b_k)$$

És d'aquest model matemàtic, quan les funcions d'activació f_a son sigmoides i la xarxa té almenys una capa oculta i una de sortida, que George Cybenko en va demostrar la capacitat d'aproximador universal [Cyb89]. Si la xarxa té una capa de sortida i una capa oculta, el model podrà aproximar qualsevol funció continua. Si té més d'una capa oculta, podrà aproximar també funcions discontínues. Afegir capes permet incrementar la complexitat de la funció que aproxima.

Aquest és un teorema d'existència, i.e. ens diu que la funció existeix, però no dona cap indicació de com trobar-la. L'algorisme de retropropagació o *backpropagation* resol el problema de trobar els paràmetres: pesos sinàptics w_{ij} i biaix b_j per a totes les capes de la xarxa, i es tractarà a continuació. No obstant, encara quedarà pendent el problema de trobar la topologia de la xarxa, a més d'altres hiperparàmetres que aniran apareixent.

3.4 Retropropagació

La retropropagació o *backpropagation* permet ajustar progressivament els valors dels pesos sinàptics i els valors de biaix de la xarxa neural, propagant enrere l'error comès per la xarxa neural. Per minimitzar l'error, s'usa el descens per gradient, que és un procés d'optimització.

El primer pas és realitzar una propagació o feedforwad amb una entrada $(x_1, \ldots, x_n)$, que és la part explicativa d'una instància del conjunt de dades. A continuació es comparen les sortides de la xarxa $(\hat{y}_1, \ldots, \hat{y}_m)$, amb el valor esperat $(y_1, \ldots, y_m)$ que és la part resposta de la mateixa instància del conjunt de dades. Per a una component k del vector sortida es calcula el seu error com

$$e_k = y_k - \hat{y}_k$$

Aquest error és el que ha comes la neurona u_k de la capa de sortida, i s'utilitza per calcular el valor

$$\delta_k = f'_a(\Sigma_k) \cdot e_k$$

que cal propagar cap a les capes precedents, per poder determinar en quina mesura ha contribuït cada neurona de la xarxa en l'error de la sortida. En una neurona u_j d'una capa oculta, cal ponderar els valors δ_k provinents de la capa següent, segons els pesos que els connecten. Quedant el δ_j d'una neurona oculta com

$$\delta_j = f'_a(\Sigma_j) \sum_{\forall k} w_{jk} \cdot \delta_k$$

Aquest proces es realitza començant per la última capa i iterant fins a la primera. Quan cada neurona ja te el seu valor δ calculat, s'actualitzen els pesos de tota la xarxa afegint els increments

$$\Delta w_{ij} = \lambda \cdot \delta_j \cdot u_i$$

on λ es diu *factor d'aprenentatge* i permet regular la velocitat d'aprenentatge. Es procedeix igual amb el biaix de cada neurona, tenint en compte que el biaix sempre té entrada 1

$$\Delta b_j = \lambda \cdot \delta_j$$

A [Rus10, Bis06] es donen demostracions d'aquest procediment aplicant la regla de la cadena de derivació a les capes ocultes. A [Roj96] es proporciona una demostració alternativa utilitzant procediments gràfics.

L'algorisme de retropropagació té una complexitat computacional lineal $O(W)$, respecte del nombre de paràmetres de la xarxa, que són tots els pesos sinàptics de les connexions més el biax de cada neurona [Bis06, pàg 246].

Aprenentatge amb moment

Durant l'actualització dels pesos sinàptics, es pot afegir un terme anomenat *moment* o *momentum*, que es denota per μ en aquest treball. [Qia98] demostra que el moment actua com la massa a física, creant una inèrcia en el descens per gradient, que pot evitar que l'algorisme s'estanqui en mínims locals i pot accelerar l'aprenentatge. El valor $\mu = 0$ equivaldria a un objecte sense massa, i en l'aprenentatge de la xarxa neural equival al mètode de retropropagació clàssic, ja que anul·la el terme del moment.

Utilitzant moment, l'actualització dels pesos sinàptics i biaix queda com

$$\begin{aligned} \Delta w_{ij} &= \lambda \cdot \delta_j \cdot u_i + \mu \cdot \Delta w'_{ij} \\ \Delta b_j &= \lambda \cdot \delta_j + \mu \cdot \Delta b'_j \end{aligned}$$

on $\Delta w'_{ij}$, $\Delta b'_j$ és l'increment que es va aplicar en la retropropagació anterior, i que serà inicialment zero. Emmagatzemar aquest increment entre execucions de la retropropagació, obliga a afegir una nova matriu per cada capa, amb les mateixes dimensions que la matriu de pesos, i un vector amb tantes components com neurones. En el codi font, aquesta matriu es diu `weights_sto` i el vector `bias_sto`.

Inicialització dels pesos sinàptics

El principal requeriment per a la inicialització dels pesos és que eviti la simetria[13] entre neurones d'una mateixa capa [Ben12]. La solució més estesa és inicialitzar-los amb valors aleatoris en un interval centrat $[-r, r]$.

A [Lec98] s'argumenta que els pesos inicials han de tenir valors tals que la funció $\tanh$ s'activi en la seva regió lineal. Si els pesos són massa grans la funció d'activació es saturarà i l'aprenentatge serà lent, amb valors molt petits els gradients seran molt petits i l'aprenentatge serà també lent. Escollint valors intermitjos la xarxa aprendrà primer la part lineal deixant les complexitats no lineals per després.

El valor dels pesos ha de ser inversament proporcional a l'arrel quadrada del nombre de neurones de la capa precedent o *fan-in*[14]. Combinant-ho amb l'aleatorietat, els pesos inicials han d'estar

[13] Si neurones d'una mateixa capa comparteixen un mateix pes d'entrada i de sortida, llavors aquestes neurones són *simètriques* i els seus pesos sempre s'actualitzaran igual, i per tant tindran la mateixa capacitat d'aprenentatge que una sola neurona.

[14] Fan-in és el nombre d'entrades d'una neurona.

centrats en 0 i amb desviació típica $1/\sqrt{m}$, amb m el fan-in de la neurona. La variància d'una variable aleatòria uniforme U amb paràmetres a, b és

$$V(U) = \frac{(b-a)^2}{12}$$

com que ens interessen valors centrats en 0, es pot fer $d = b - a$, llavors igualant

$$\sqrt{\frac{d^2}{12}} = \frac{1}{\sqrt{m}} \Rightarrow |d| = \frac{2\sqrt{3}}{\sqrt{m}}$$

així, interessen els pesos inicials aleatoris en l'interval $\left[-\frac{\sqrt{3}}{\sqrt{m}}, \frac{\sqrt{3}}{\sqrt{m}}\right]$.

3.5 Estratègies d'entrenament

L'entrenament de la xarxa neural consisteix en presentar tots els exemples del conjunt d'entrenament, i per a cada un executar una fase de propagació, i immediatament una de retropropagació per corregir els pesos en funció de l'error comès. Quan s'ha processat tot el conjunt d'entrenament, es diu que s'ha superat una *època* o *epoch*.

Aquest proces es repetirà fins que es compleixi un criteri de parada, que dependrà de l'estratègia d'entrenament seguida. Normalment serà que es verifiqui una de dues condicions: que l'error de la època és menor que un error objectiu prefixat, o que s'ha superat un limit màxim d'èpoques. També és habitual parar abans de que es verifiqui cap de les dues condicions anteriors, si es detecta que s'incrementa l'error de generalització. Aquest criteri de parada es diu *early-stopping*, i per poder aplicar-lo es necessita un conjunt de dades addicional durant l'entrenament, anomenat *conjunt de validació*, independent del conjunt de test, que s'utilitzarà finalment per validar el model, i que no ha de tenir cap implicació durant l'entrenament. Aquest conjunt addicional implica partir les dades originals en tres conjunts: entrenament, validació i test, per tant, aquesta estratègia està condicionada al nombre d'exemples disponibles en el conjunt de dades.

Càlcul de l'error

Segons [Pre94, Secció 2.6], es poden usar diverses mesures per a l'error[15] de la xarxa neural. La més comú és l'error quadràtic, que es calcula per a cada instància com $e = \sum_{k=1}^{m}(y_k - \hat{y}_k)^2$, on $\hat{y}_k$ són els valors de les neurones de la capa de sortida, i y_k els valors esperats per a cada una[16]. Per donar una mesura independent del nombre de neurones de la sortida, es pot calcular l'error quadràtic mig de la sortida com e/m.

La mesura anterior és per a un únic exemple. Per a mesurar l'error d'un conjunt de dades, és habitual calcular la mitja de l'error de cada exemple, que serà l'error quadràtic mig del conjunt de dades, i és independent de la mida del conjunt de dades i del nombre de sortides de la xarxa:

$$\text{EQM} = \frac{1}{Nm}\sum_{i=1}^{N}\sum_{k=1}^{m}(y_k - \hat{y}_k)^2$$

on N és el cardinal del conjunt d'instàncies, i m el nombre de sortides de la xarxa. No obstant, l'EQM segueix sent dependent dels valors de sortida del conjunt de dades, i serà necessari normalitzar per poder comparar l'EQM entre conjunts diferents.

[15] Aquestes mesures també reben noms com *funció d'error*, *funció objectiu*, *funció de cost* o *funció de pèrdua*.

[16] Seguint les equacions habituals de retropropagació amb descens per gradient, aquest error s'ha de multiplicar per 1/2, ja que així les derivades són més simples, no obstant no és una pràctica estàndard ja que distorsiona la mesura d'error, i és preferible multiplicar per 2 la derivada, o dividir de 2 el factor d'aprenentatge [Pre94, pàg. 13].

Entrenament on-line vs batch

L'entrenament on-line és el que implementa la classe `OnlineTrainer`, i consisteix en presentar un exemple a la xarxa neural, calcular l'error, aplicar la retropropagació i immediatament actualitzar els pesos sinàptics de la xarxa. A continuació es procedeix amb el següent exemple, fins completar una epoch. L'avantatge d'aquesta estratègia és que la xarxa neural aprèn des del primer exemple.

Una alternativa és el *batch training*, que consisteix en presentar un exemple a la XN, calcular l'error, aplicar la retropropagació i acumular els increments Δw_{ij} en una estructura separada de la xarxa sense actualitzar els pesos sinàptics. Quan ha finalitzat la epoch, es procedeix a actualitzar els pesos de la XN amb els valors acumulats. Una variant d'aquesta estratègia és *minibatch training*, que procedeix de la mateixa manera però actualitza els pesos de la XN diverses vegades per epoch, amb l'objectiu de trobar un compromís entre *online* i *batch*.

Treballs com [Wil03] argumenten que les estratègies d'entrenament *batch* i *minibatch* són ineficients en la majoria dels casos, i que no hi ha cap raó per usar-les. Altres autors [Roj96, pàg. 170] recomanen usar entrenament on-line per a conjunts de dades amb molts exemples, argumentant que és molt costós calcular la direcció exacta del gradient quan una epoch conté milers d'exemples.

Criteri de parada

El criteri de parada és la condició que determina quan cal parar l'entrenament de la xarxa neural. Els més simples són executar l'entrenament durant un número fix d'epochs, o parar quan una mesura d'error de la xarxa, normalment l'EQM, sigui menor que un valor donat. Aquest error dependrà de les dades i del soroll que contenen, dades molt estocàstiques convergiran a errors alts, dades deterministes convergiran a errors baixos. Per contra, si es pretén que dades estocàstiques convergeixin a un error molt baix i la XN té capacitat suficient, possiblement es produirà overfitting.

És probable que l'error de les dades no sigui un valor conegut, i en aquest cas és difícil escollir el valor de parada adient:

- Una sobreestimació de l'error de parada produirà un model poc acurat, desaprofitant el potencial de la xarxa neural.
- Una subestimació de l'error de parada produirà que la xarxa aprengui més del compte, incrementant el risc d'overfitting.

Quan s'usa el criteri de l'error, es diu que la XN *convergeix* si para per haver assolit l'error objectiu, i que no convergeix si oscil·la en valors superiors a l'error de parada sense arribar-hi, i en aquests cas es pararà l'entrenament per esgotar un limit d'epochs de seguretat.

Un criteri de parada alternatiu que pretén evitar l'overfitting és *early-stopping*. Amb aquest criteri, la XN s'entrena amb dos datasets diferents, d'entrenament i de validació. El primer s'usa per entrenar la XN, i amb l'altre es calcula l'error de la XN a cada epoch. L'error calculat amb el conjunt de validació es diu *error de generalització*. En un entrenament típic, l'error de generalització es reduirà inicialment, fins arribar un punt en que comenci a creixer lentament. En aquest punt la XN ha assolit el seu menor error i es pot parar l'entrenament. Es construeix el model amb els valors dels pesos sinàptics on es va aconseguir el menor error de generalització.

L'*early-stopping* presenta tres problemes fonamentals. En primer lloc és complicat decidir quan l'error de generalització ha assolit el mínim, ja que aquest error pot oscil·lar durant l'entrenament. Un altre inconvenient és que cal sobreentrenar la XN per trobar aquest mínim. El pitjor inconvenient és que necessita un segon conjunt de dades per la validació, que no es podrà usar per entrenament ni tampoc per validar el model final. És a dir, cal partir les dades originals en tres subconjunts per a construir el model de XN. Això pot ser un inconvenient greu en datasets petits.

La classe `OnlineTrainer` usa el criteri de parada per error i per màxim d'epochs. El motiu és que a l'algorisme genètic ja s'implementa un mecanisme equivalent a l'*early-stopping*.

3.6 Detalls de la implementació

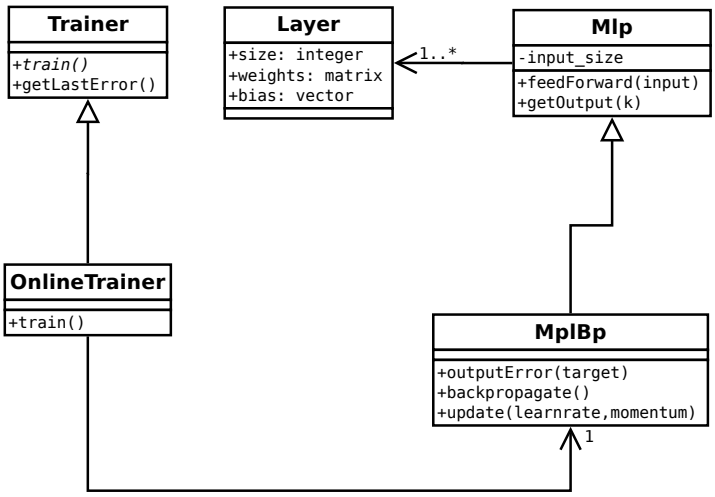

Figura 6: Esquema UML amb els principals conceptes de la xarxa neural.

La figura 6 mostra les relacions entre les principals components de la implementació de la xarxa neural. Una classe abstracta Mlp conté una o més capes i proporciona operacions com feedForward(input) per calcular sortides a partir d'una entrada, i getOutput(k) per recuperar la sortida k. La classe Layer és un simple contenidor de dades relatives a una capa. La classe MlpBp deriva de Mlp i proporciona la funcionalitat de retropropagació amb el mètode backpropagate(), igualment proporciona mètodes per calcular l'error de la última capa a partir d'una entrada objectiu *target*, i per actualitzar tots els pesos de la XN després del backpropagate. La classe abstracta Trainer proporciona diverses eines per recuperar informació d'un entrenament, i la classe que s'en deriva OnlineTrainer implementa l'entrenament online en el mètode train(). D'aquesta manera, és possible afegir noves estratègies d'entrenament amb modificacions mínimes del codi.

Ús de la xarxa neural des de C++

La classe genann::MlpBp encapsula un perceptró multicapa amb aprenentatge per retropropagació. Els paràmetres del seu únic constructor són:

- `uint isize`: Mida de l'entrada de la XN.
- `const UIntVec& lsizes`: Vector d'enters amb la mida de cada capa oculta.
- `Activator* lastAct`: Funció d'activació per a les neurones de l'última capa.

Per entrenar un objecte MlpBp, s'usa un objecte de tipus Trainer, com OnlineTrainer, i els paràmetres del seu constructor són:

- `MlpBp& nn`: La XN a entrenar.
- `double lr`: El factor d'aprenentatge λ.
- `double mo`: El momentum μ.
- `double tmse`: L'EQM usat com a criteri de parada.
- `uint maxep`: El nombre màxim d'epochs a executar.

El següent fragment de codi, extret de la unitat de compilació rentry.cpp, serveix d'exemple de com usar-los conjuntament:

```
UIntVec lsizes = getLayerSizes(rhlsizes, output_size);
Activator* lact = getActivator(SIGMOID);
```

```
MlpBp mlpBp(input_size, lsizes, lact);

vector<double*> inputs = ...
vector<double*> outputs = ...

Trainer* trainer;
trainer = new OnlineTrainer(mlpBp, learnfactor, momentum, targetmse, maxep);
trainer->train(inputs, outputs);
```

Un cop entrenada la XN, es poden realitzar prediccions passant una entrada al mètode feedForward i recollint les sortides amb `getOutput`:

```
mlpBp.feedForward(input);
for (uint k=0; k<output_size; ++k) cout << mlpBp.getOutput(k) << endl;
```

Ús de la xarxa neural des d'R

La sintaxi per executar la XN des l'R és:

```
ann(x, y, hls=NULL, lr=0.01, mo=0, tmse=0.001,
    maxep=5000, lact="linear", printevery=100)
```

on `x` i `y` són dataframes amb les variables d'entrada i les de sortida respectivament, que òbviament han de tenir el mateix número de files. El paràmetre `hls` és un vector d'enters que determina la topologia de les capes ocultes. Els següents paràmetres determinen el factor d'aprenentatge `lr`, el moment `mo`, l'EQM objectiu `tmse`, el màxim número d'epochs `maxep`, la funció d'activació de la capa de sortida `lact` que pot ser "linear", "sigmoid", "tanh", "sigmoidtt" o "tanhtt", i s'ha d'elegir en funció dels recorreguts de les variables resposta, veure la taula 3. Finalment, el paràmetre `printevery` indica cada quantes epochs s'imprimirà un missatge amb l'EQM. Es pot obtenir una descripció detallada usant l'ajuda d'R amb `?ann`.

Normalment es disposarà d'un únic dataframe amb tots els exemples d'entrenament. Per separar les variables explicatives `x` de les resposta `y`, només cal seleccionar les columnes corresponents amb l'operador [] d'R, per exemple

```
model <- ann(data[1:10], data[11:13], hls=c(16,16), lact="sigmoid")
```

entrenarà una xarxa neural de topologia 10-16-16-3, perquè les 10 primeres columnes del dataframe `data` són les variables explicatives i per tant les entrades de la XN, les 3 últimes són les variables resposta, i la topologia de les capes ocultes és 16-16. Utilitzarà la funció d'activació $\tanh$ en les neurones ocultes i la funció sigmoide en les neurones de la capa de sortida.

L'objecte `model` retornat conté informació de l'entrenament i els paràmetres de la XN (pesos sinàptics i biaixos), de manera que es poden realitzar prediccions amb la funció `precit()`. El tutorial de la secció 2.4 conté un exemple del seu ús.

Notació i identificadors

La taula 4 resumeix la notació que s'utilitza en aquest treball i els identificadors corresponents en el codi font. Els identificadors abreviats s'utilitzen en variables locals. El codi font utilitza aquests identificadors excepte quan cal evitar un conflicte o per millorar la claredat del codi. En aquest casos s'usaran variants o noms similars.

Concepte	Notació	Identificador
index per capes:	subindex l	`l`
index neurones capa precedent:	subindex i	`i`
index neurones capa actual:	subindex j	`j`
index neurones capa següent:	subindex k	`k`
biaix de la neurona j:	b_j	`bias[j]`
increment de biaix anterior:	$\Delta b'_j$	`bias_sto[j]`
increment de biaix:	Δb_j	`inc`
pes sinàptic:	w_{ij}	`weights[j][i]` (matriu trasposada)
increment de pes anterior:	$\Delta w'_{ij}$	`weights_sto[j][i]`
increment de pes:	Δw_{ij}	`inc`
entrada de la neurona j:	Σ_j	`prevoutput[j]`
sortida de la neurona j:	u_j	`output[j]`
factor d'aprenentatge:	λ	`learnrate`, `lr`
moment o *momentum*:	μ	`momentum`, `mo`
delta de la neurona j:	δ_j	`delta[j]`
error de la sortida k:	e_k	`error`, `err`
error d'una instància	e	`sse` (sum of squared errors)
EQM d'una instància	e/m	—
EQM d'una època	EQM	`error`

Taula 4: Notació i identificadors per la implementació de la XN. Els identificadors coincidents es distingeixen clarament en el codi font segons la funció on s'usen.

3.7 Proves amb la xarxa neural

Dades sintètiques

Les proves de la xarxa neural comencen amb regressions sobre conjunts de dades sintètiques sense soroll, l'objectiu és comprovar que la XN funciona correctament i pot aproximar funcions. Les dades sintètiques són útils perquè es coneixen els valors que han de tenir les prediccions, si el conjunt serà o no ajustable i la dificultat d'obtenir el model. Són per tant un mètode fiable per posar a proba la implementació.

Per cada regressió, s'ha generat dades sintètiques en l'interval [-1, 1] seguint una funció. A continuació s'han extret aleatòriament 2/3 d'instàncies per a l'entrenament deixant l'altre 1/3 per la prova. S'ha entrenat una xarxa neural de topologia 1-16-16-1, amb $\lambda = 0.1$, $\mu = 0.1$, un EQM objectiu de 0.001 i un màxim de 5k iteracions. Finalment s'han realitzat prediccions amb el conjunt de prova.

Les figures 7, 8 i 9 mostren les gràfiques de les regressions. Les tres figures tenen la mateixa estructura. Els tres gràfics superiors mostren les dades. A l'esquerra les dades originals, al centre el conjunt d'entrenament, a la dreta les dades de prova i les seves prediccions. Els tres gràfics inferiors mostren els errors i residus. En el primer l'evolució de l'EQM durant l'entrenament de la xarxa (l'escala vertical és logarísmica), al centre un histograma que mostra la distribució dels residus de la predicció sobre el conjunt de prova, a la dreta els residus de cada predicció. En els gràfics, es pot observar que els models obtinguts tenen la capacitat d'interpolació esperada, els residus són tots molt propers a zero indicant una bona aproximació, excepte en la funció soroll.

La taula 5 mostra un resum dels resultats, on es pot comprovar que la funció soroll no ha convergit a l'error objectiu, com era d'esperar. En canvi, els models construïts per a les funcions trigonomètrica i discontinua, han assolit l'error objectiu d'entrenament, que indica que han après les dades, i a més han obtingut valors molt baixos per a l'EQM calculat en amb el conjunt de test, indicant que han aconseguit capacitat de generalització.

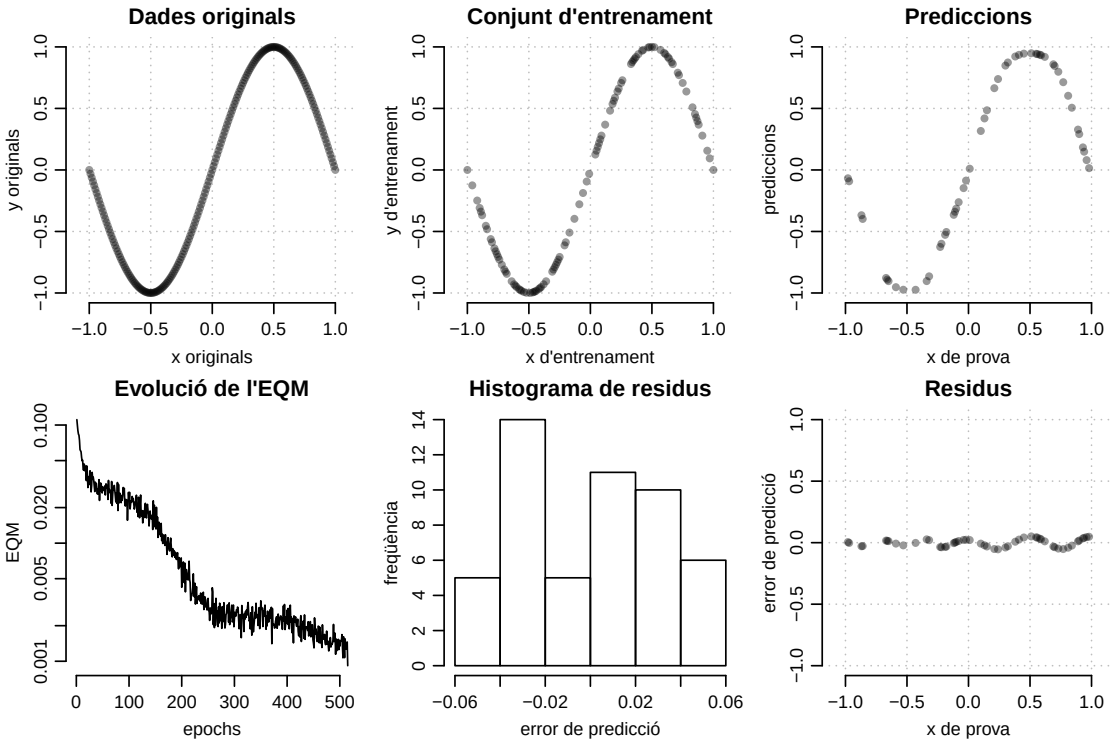

Figura 7: Regressió amb dades sintètiques per la funció $y = sin(\pi x)$. La predicció és correcte, els residus són molt propers a zero.

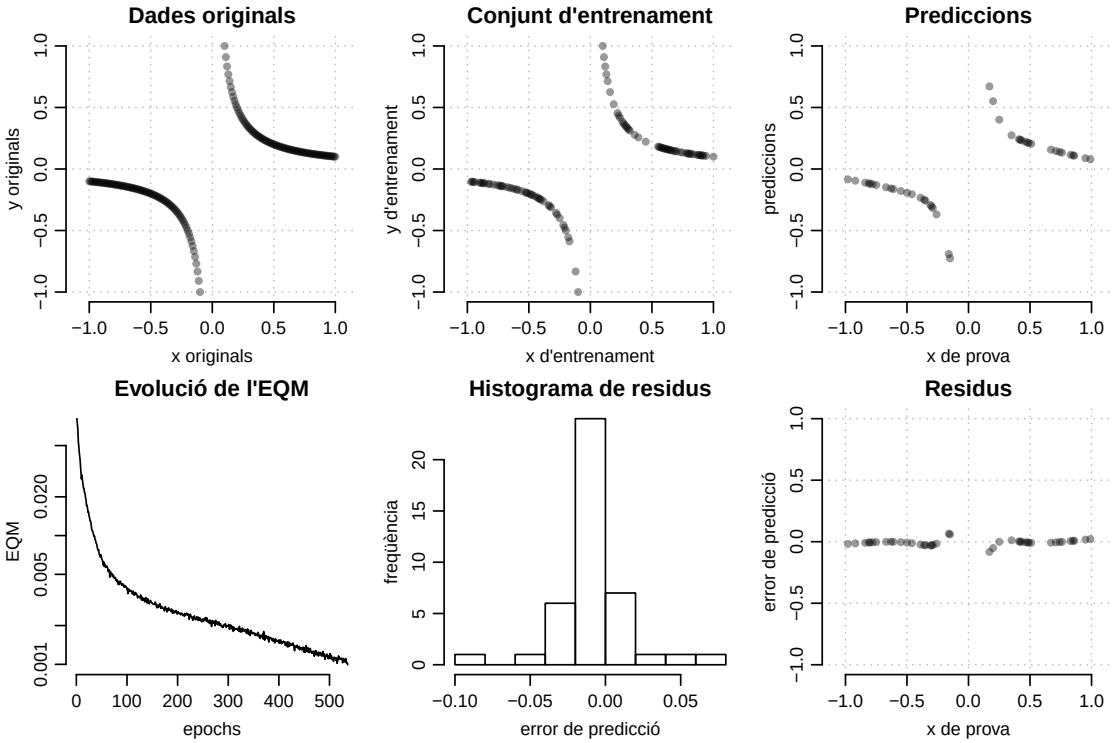

Figura 8: Regressió amb dades sintètiques per la funció $y = 1/x$. Al ser una funció discontinua, per aproximar-la correctament calen dues capes ocultes com a mínim. L'aproximació és correcta, els residus són molt propers a zero.

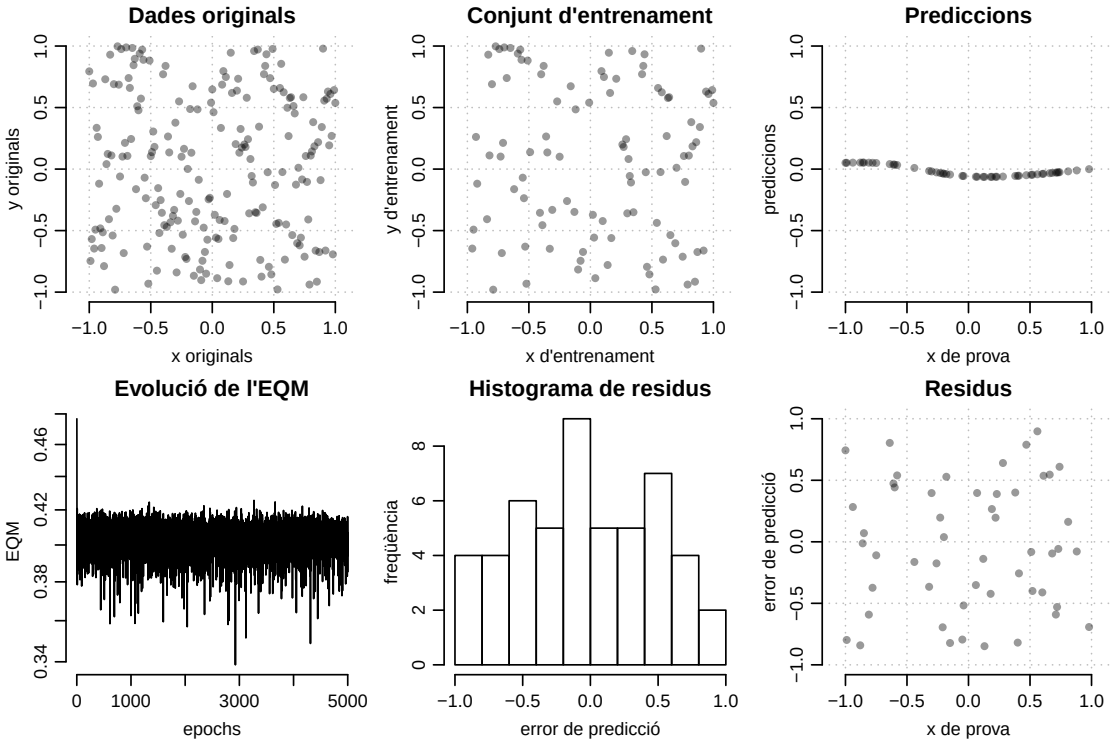

Figura 9: Regressió amb dades sintètiques per a una funció soroll. Es pot comprovar que la xarxa neural no convergeix i no es pot ajustar cap model.

	EQM entrenament	EQM test	Convergeix	Epochs	Temps
sin	0.071	0.093	si	515	0.22
discontinua	0.088	0.058	si	536	0.18
soroll	38.144	25.903	no	5000	1.59

Taula 5: Resultats dels conjunts de dades sintètiques. Els EQM es mostren multiplicats per 100, i el temps d'execució en segons.

Dades de PROBEN1

A continuació es prova la xarxa neural amb dades reals obtingudes dels conjunts PROBEN1. S'han creat 10 models per a cada un dels datasets building, flare, card, heart i thyroid. S'han realitzat 1000 epoch en cada entrenament, amb $\lambda = 0.1$, $\mu = 0.5$ i una xarxa amb topologies 14-8-8-3 per a building1–3, 24-8-8-3 per a flare1–3, 51-8-8-2 per a card1–3, 35-8-8-2 per a heart1–3, i 21-8-8-3 per a thyroid1–3.

Per a cada dataset es calcula la mitja $\bar{x}$ i la desviació típica s de l'EQM dels 10 models creats, per al conjunt d'entrenament i per al conjunt de prova. Els valors obtinguts es mostren a la taula 6 i poden comparar-se amb les taules 10 i 11 de [Pre94]. Es comprova que els valors son similars, tot i que la xarxa usada al citat article té una topologia diferent, utilitza Rprop com a algorisme d'aprenentatge i *early-stopping* com a criteri de parada. Els problemes de classificació card i heart mostren un clar overfitting, que quedarà resolt quan l'algorisme genètic calculi l'EQM de parada.

	EQM entr		EQM test	
	$\bar{x}$	s	$\bar{x}$	s
building1	0.081	0.003	0.691	0.054
building2	0.232	0.012	0.259	0.011
building3	0.234	0.006	0.263	0.005
flare1	0.183	0.004	0.756	0.059
flare2	0.226	0.011	0.471	0.070
flare3	0.238	0.007	0.479	0.060
card1	2.375	0.690	16.129	1.349
card2	2.177	0.647	22.262	2.034
card3	3.085	1.083	17.352	2.082
heart1	3.943	1.075	19.477	1.008
heart2	3.939	1.331	21.334	1.814
heart3	4.861	3.491	22.592	1.510
thyroid1	2.374	0.366	2.991	0.279
thyroid2	2.373	0.359	2.694	0.279
thyroid3	3.560	1.835	3.644	1.830

Taula 6: Resultats en conjunts de dades de PROBEN1. Els valors s'han multiplicat per 100 per facilitar la comparació.

4 Implementació de l'algorisme genètic

En aquesta secció es descriurà amb detall l'algorisme genètic implementat. L'objectiu de l'AG és, partint d'un conjunt de dades donat, trobar una configuració de xarxa neural que sigui capaç de construir un model predictiu per al problema que descriuen les dades. Així, el primer objectiu és obtenir el model, però interessa també que la XN resultant sigui el més propera a la xarxa òptima que sigui possible.

Quina és la XN òptima és dependent de les necessitats de cada problema particular i no es pot decidir a priori. En unes aplicacions convindrà que sigui d'una mida el més petita possible, per exemple si aquesta s'ha d'implementar en hardware sota certes restriccions. En altres ocasions es pot requerir la màxima precisió sense importar la mida de la XN. També pot ser necessari trobar una XN que pugui aprendre en un mínim d'epochs, podent sacrificar precisió. Per admetre aquesta versatilitat es definirà una funció fitness que permet regular la relació entre les tres característiques: precisió, mida i velocitat d'aprenentatge. No s'ha oblidar però, que en ocasions hi haurà una única configuració no dominada amb capacitat predictiva i l'AG no tindrà alternatives, i en dades amb molt de soroll, pot ser que ni tan sols sigui possible trobar un model predictiu.

En aquest treball s'ha seguit la versió més clàssica de l'algorisme genètic, amb potser una única aportació que permet regular la pressió de selecció amb un paràmetre configurable i una variable aleatòria exponencial. Existeixen múltiples variacions i versions de l'algorisme original, encara que ni tan sols en la versió original es disposa d'una fonamentació matemàtica sòlida que expliqui el seus resultats. Per això, en aquesta primera versió no té sentit provar variants i s'ha optat per la versió més coneguda i estudiada.

A pesar de la poca formalització matemàtica, els algorismes genètics són mecanismes robustos de cerca, que fonamenten el seu funcionament en imitar la genètica i la selecció natural, i que han donat bons resultats en diverses àrees i aplicacions des de la seva invenció fa més de 50 anys. El seu ús s'acostuma a justificar a partir d'estudis empírics, i són d'aplicació quan l'espai de cerca és molt gran i no es disposa de cap heurística que pugui guiar la cerca cap a una solució. Una de les claus del seu èxit, és que cal assumir molt poc respecte de l'espai de cerca, aquest pot tenir qualsevol forma, mida o complexitat, l'únic requeriment de l'AG és que sigui possible expressar els seus valors en forma de cadena de bits. Així, l'AG és adient per a la tasca de recórrer l'espai d'hiperparàmetres d'una xarxa neural, intentant aproximar una configuració que sigui propera a l'òptima. Degut a que l'AG és una cerca estocàstica, no es pot garantir que l'algorisme trobi un òptim [Gre00].

4.1 Fonamentació i conceptes

Com s'ha comentat a la introducció, ja en treballs anteriors a la informàtica actual es parlava d'usar tècniques que imitessin la selecció natural i l'evolució en processos d'optimització. S'ha posat com a exemple el treball de 1950 "Computing Machinery and Intelligence" d'Alan Turing, perquè en ell es parla a més, d'imitar la selecció natural per millorar una màquina amb capacitat d'aprenentatge.

No obstant, els algorismes genètics com a tals, van ser inventats per John Holland a finals de la dècada de 1950. El seu llibre "Adaptation in natural and artificial systems" de l'any 1959 i reeditat l'any 1992 [Hol92] és la primera obra que els descriu amb detall. Posteriorment al 1975, Kenneth De Jong, alumne de Holland, va estudiar el comportament dels algorismes genètics recollint els resultats en el treball "An Analysis of the Behavior of a Class of Genetic Adaptive Systems" [Dej75].

L'algorisme de Holland i les seves variants encara s'utilitzen actualment i és el que s'implementarà en aquest treball. En la seva versió fonamental, coneguda com sGA[17] [Saf04], l'algorisme treballa amb un conjunt finit de cromosomes anomenat *població*. Es parteix d'una població inicial de cromosomes aleatoris, es crea per a cada un el seu organisme corresponent i s'avalua una

[17] Un sGA (simple Genetic Algorithm) té les següents característiques: població finita, representació del cromosoma amb bits, creuament d'un sol punt (locus), mutació per inversió de bit, selecció *fitness-proportional* (també coneguda com seleccó per ruleta).

funció d'aptitud o *fitness*. Cada nova generació d'organismes es forma a partir de tres operacions genètiques sobre la generació anterior:

1. **Selecció:** S'elegeixen els progenitors amb probabilitat proporcional al seu fitness.
2. **Creuament:** A partir de dos cromosomes progenitors, es crea un nou cromosoma amb una probabilitat prefixada anomenada *probabilitat de creuament*.
3. **Mutació:** Es modifica el nou cromosoma invertint alguns dels seus bits, amb una probabilitat prefixada anomenada *probabilitat de mutació*.

Es segueix aquest procés per crear noves generacions iterativament, fins que es compleixi un criteri de parada preestablert. Amb aquest algorisme, la optimització es produeix indirectament a través de les operacions genètiques. La selecció afavoreix els millors organismes conduint l'algorisme cap a un òptim, i el creuament i la mutació exploren l'espai de cerca.

Cada cromosoma codifica una posició de l'espai de cerca i conté la següent informació, que correspon als hiperparàmetres necessaris per configurar la topologia i l'aprenentatge d'una xarxa neural:

- Error objectiu de la XN.
- Factor d'aprenentatge λ i momentum μ.
- Topologia de la XN.

Les operacions genètiques de creuament i mutació en el cromosoma alteraran aquests valors, permetent l'aparició de diferents organismes, cada un dels quals correspon a la configuració d'una nova xarxa neural. La xarxa neural d'un nou organisme s'avaluarà i es registraran tres mètriques:

- Error de generalització: mesurat amb l'EQM calculat amb prediccions realitzades sobre el conjunt de validació. Avalua la precisió de la XN amb dades diferents de les del conjunt d'entrenament.
- Nombre de paràmetres: el nombre W de pesos sinàptics i biaixos de la XN, que determina el requeriment de memòria de la xarxa i la complexitat computacional $O(W)$ de les operacions de propagació i retropropagació.
- Nombre d'epochs: avalua el temps necessari per entrenar la XN.

Aquestes tres mètriques es combinaran per donar valor al fitness de l'organisme, un valor escalar que crearà un ordre en la població i determinarà la supervivència de l'organisme o de la seva genètica, vía elitisme o descendència respectivament. Tots els gens poden tenir influència en les tres mètriques de la XN, però les relacions més directes són:

- L'error objectiu permet a l'AG limitar el nombre d'epochs per una banda, i evitar l'overfitting millorant l'EQM de generalització per l'altre.
- λ i μ permeten optimitzar l'aprenentatge de la XN, reduint el nombre d'epoch, millorant l'EQM o totes dues a l'hora.
- La topologia de la xarxa afecta directament al nombre de paràmetres, i indirectament a l'EQM i el nombre d'epochs necessàries per a l'aprenentatge.

Per finalitzar aquest apartat, s'ha recollit a la taula 7 diverses correspondències habituals entre conceptes de genètica biològica i la seva interpretació simplificada en els algorismes genètics. Punts addicionals que cal destacar:

- El genoma està format per la càrrega útil del cromosoma. En els AG normalment tots els bits són útils i per tant cromosoma=genoma. No obstant, segons la codificació que s'utilitzi en els gens, pot passar que dos cromosomes diferents resultin en dos organismes idèntics i en aquests casos la distinció entre cromosoma i genoma és útil.
- En aquest treball, un mateix cromosoma pot generar diferents organismes cada un amb la seva pròpia XN i per tant una avaluació del fitness diferent.

Biologia	Algorisme genètic
base nitrogenada	bit
gen	seqüència de bits
cromosoma	seqüència de gens
locus	posició en el cromosoma
mutació	inversió d'un bit en el cromosoma
creuament	creació d'un nou cromosoma a partir de dos existents
genoma	bits útils en el cromosoma
genotipus	cromosoma descodificat, els valors numèrics dels gens
fenotipus	organisme resultant del cromosoma descodificat
organisme	instància del tipus Organism, realització del fenotipus
població	conjunt d'organismes
generació	població en una iteració de l'algorisme

Taula 7: Correspondència amb els conceptes genètics de biologia.

4.2 Codificació dels hiperparàmetres

Els algorismes genètics codifiquen les coordenades de l'espai de cerca en una cadena de bits. Quan l'espai de cerca és multidimensional, el valor de cada dimensió es codificarà en una seqüència de bits anomenada *gen*, i la concatenació de tots els gens formarà un *cromosoma*. Així, un cromosoma codifica un vector que representa una coordenada o posició de l'espai de cerca.

El cromosoma d'un organisme es forma dels següents gens[18]:

- `TE_GENE` (Target Error) codifica l'error objectiu `tmse` de la XN, que és la mesura que s'usarà com a criteri de parada en l'entrenament de la XN de l'organisme.
- `LR_GENE` (Learning Rate) codifica el factor d'aprenentatge $\lambda \in (0, 2]$.
- `MO_GENE` (MOmentum) codifica el momentum $\mu \in [0, 1]$.
- `NH_GENE` (Number of Hiddens) codifica el nombre de capes ocultes de la XN. Aquest valor regula la topologia de la XN.
- `HS_GENE` (Hidden Size) codifica la mida de les capes ocultes de la XN. Aquest gen es repetirà $2^{\texttt{nh\_bits}} - 1$ vegades, amb `nh_bits` el nombre de bits del gen `NH_GENE`.

Un cop fixat el nombre de bits de `NH_GENE` en iniciar l'AG, el cromosoma té una mida fixa, encara que no necessàriament s'usin tots els gens `HS_GENE`. Aquesta codificació permet un nombre variable de capes ocultes, entre 0 i $\texttt{max\_layers} = 2^{\texttt{nh\_bits}} - 1$.

Tots els gens es codifiquen com a números enters sense signe, usant el sistema numèric posicional en base 2. Addicionalment, els gens `TE_GENE`, `LR_GENE` i `MO_GENE` han de ser convertits a variables de coma flotant, per la qual cosa s'usarà una correspondència lineal [Col99]. Si $z \in \mathbb{Z}$ és un enter representat en el cromosoma amb N bits, i $[r_{\min}, r_{\max}] \subset \mathbb{R}$ és l'interval real que es vol codificar, llavors es pot obtenir el número $r \in \mathbb{R}$ corresponent a z amb

$$r = \frac{r_{\max} - r_{\min}}{2^N - 1} z + r_{\min}$$

Organismes mínim, màxim i rectangular

Amb la codificació especificada, es pot construir una xarxa neural amb qualsevol nombre de capes i número de neurones per cada capa, però un cop iniciat l'AG, la mida del cromosoma queda fixada i a partir d'aquest punt existeix una xarxa de mida màxima la qual no es podrà superar. A [Bra94] aquesta xarxa s'anomena *organisme màxim*, i és útil per evitar que l'AG perdi temps provant topologies excessivament grans per al problema a tractar. Així, l'AG cercarà topologies entre l'organisme mínim i el màxim.

[18] Les variables i constants es corresponen amb els identificadors de la classe Organism

L'organisme mínim està format únicament per la capa de sortida, amb tantes neurones com variables resposta tingui el problema. No té cap capa oculta, i per tant només podrà aprendre problemes linealment separables. En aquest organisme s'ignoren tots els gens `HS_GENE`. L'organisme màxim en canvi es forma del nombre màxim de capes, cada una de les quals té el nombre màxim de neurones.

En la implementació s'ha deixat oberta la possibilitat de crear topologies en les que totes les capes ocultes tinguin el mateix nombre de neurones. Se les ha anomenat *organismes rectangulars* i pot configuar-se aquesta opció amb l'argument `rec=TRUE`. Aquesta opció redueix l'espai de cerca i pot accelerar l'evolució. Quan s'activa, únicament s'utilitza el valor del primer gen `HS_GENE` que determinarà la mida de totes les capes ocultes, la resta de gens `HS_GENE` s'ignorarà.

Rang de valors dels gens

Cada gen admet valors en un interval fitat. Aquests límits s'especifiquen en diverses constants a la classe Organism, i s'han establert en funció de les recomanacions de diversos treballs com [Ben12, Lec98]. S'han usat els intervals més amplis possible, dins dels límits raonables, per ampliar l'espai de cerca. Encara que pugui semblar contradictori, un espai de cerca més gran implica més possibles solucions i més treball per l'AG, però també contindrà més camins cap a l'òptim.

Cada gen correspon a una dimensió de l'espai de cerca. Limitar els valors del gen a un interval més reduït, implica fer més estreta la seva dimensió i eliminar tots els camins que queden fora de l'interval i que podrien conduir a l'òptim amb una pendent més favorable. Un símil en un terreny 2D seria limitar el terreny que un excursionista pot recórrer, a un rectangle en el que la latitud s'ha limitat a uns pocs metres. Si l'excursionista està a l'est d'una muntanya, l'haurà de pujar-la per arribar a la vall que hi ha a l'oest. Ampliant la latitud, l'excursionista pot donar la volta pel nord de la muntanya i arribar a la vall sense haver d'escalar.

Això explica perquè en alguns problemes, especialment els de classificació, l'AG no evoluciona quan s'usen intervals reduïts, encara que la solució òptima estigui contemplada en aquests intervals. Per altra banda, ampliar els intervals mes enllà dels valors que poden prendre els hiperparàmetres produeix un efecte similar, possiblement degut a que s'afegeixen regions sense pendent en la funció fitness, provocant que l'AG s'estanqui quan hi entra.

Inicialització del cromosoma

És habitual inicialitzar el cromosoma amb bits aleatoris, el que genera una població inicial d'organismes molt diversos, i així es fa en aquesta implementació. Una altre opció és iniciar amb tots el cromosomes a zero o amb valors inicials prefixats, així les primeres generacions són molt ràpides perquè els organismes són molt petits, i van creixent segons sigui necessari. Unes poques proves realitzades amb aquesta variant han estat efectives en alguns casos, però en altres s'han compensat amb una evolució més llarga. Caldria realitzar un experiment per comprovar l'eficiència d'aquesta alternativa, i per això s'han deixat comentades en el codi font les línies que permeten inicialitzar un organisme amb una mida mínima.

4.3 Operacions genètiques

Selecció

La selecció és el mecanisme que escull els organismes que passaran a formar part de la propera generació. Habitualment els organismes seleccionats seran sotmesos a altres operacions genètiques com el creuament i la mutació, però també poden passar directament sense cap modificació genètica a la propera generació si no es produeix cap mutació i si el creuament es realitza en els extrems del cromosoma. A més, si s'usa elitisme, els millors individus d'una generació passaran directament a la propera.

Una pressió de selecció alta, en la que els gens dels millors organismes tenen més probabilitat de passar a formar part de la propera generació, implicarà una cerca més dirigida, i pot significar que no s'explora completament l'espai de cerca. Per contra una pressió de selecció baixa ocasionarà una evolució molt lenta. És important disposar d'un mecanisme que permeti regular la pressió de selecció per ajustar adientment aquest comportament.

El mecanisme de selecció *fitness-proportional* descrit a [Tor10] és el més simple i habitual [Col99]. Consisteix en seleccionar els organismes amb una probabilitat proporcional a la seva funció fitness. No obstant, aquest mecanisme de selecció presenta dos problemes. El primer és que si un organisme obté un fitness molt superior a la resta, aquest dominarà la propera generació eliminant la diversitat genètica, un organisme així es coneix com *super organisme*. El segon problema és produeix en les últimes generacions de l'evolució, quan tots els organismes tenen un fitness molt similar, la pressió de selecció es redueix alentint l'evolució.

Per aquest motiu, a [Col99, Ranking Methods] es presenta un mecanisme de selecció alternatiu, en el qual els organismes s'ordenen en una llista segons el seu fitness, i la probabilitat de selecció d'un organisme és funció de la seva posició en la llista en comptes de ser funció directa del fitness. Aquest mecanisme evita els dos problemes mencionats anteriorment. Per una banda es manté una probabilitat de selecció superior en els individus amb millor fitness (que estaran primers en la llista). Per altra banda la pressió de selecció es manté constant en les successives generacions durant l'evolució.

A més, amb aquest mecanisme de selecció és possible escollir una funció de probabilitat que permeti regular la prioritat dels elements de la llista. Una opció és usar la variable aleatòria exponencial, en la que els valors propers a $x = 0$ són els més probables, i la probabilitat es va reduint conforme x creix. A més, el seu paràmetre β permet variar el ritme de decreixement. És doncs una variable aleatòria adient per a prioritzar els primers elements de la llista i configurar la pressió de selecció (veure la figura 10).

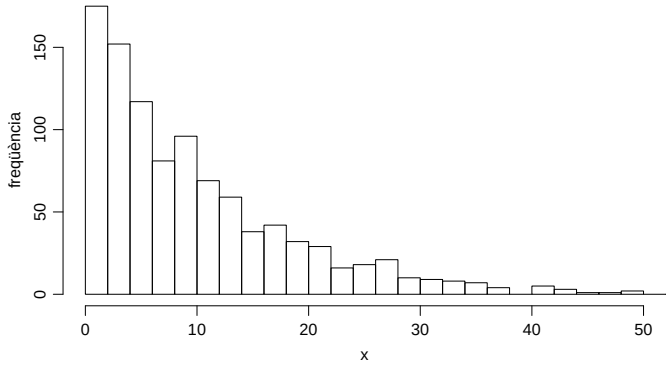

Figura 10: Histograma d'una VA exponencial amb mitja $\beta = 10$. L'eix x representa la posició en la llista, i els més propers al principi de la llista tenen major probabilitat de ser escollits. Amb $\beta = 10$ la probabilitat d'escollir un dels 10 primers és la mateixa que la d'escollir qualsevol dels següents, així, variant β es pot ajustar la pressió de selecció. El gràfic s'ha generat a R amb les comandes:
y = runif(1000); x = -10 * log(1 - y); hist(x, ...)

Per obtenir valors que segueixen aquesta distribució a partir d'un generador de nombres aleatoris uniformes, s'utilitza el mètode de simulació descrit a [Deg12, 3.8 Functions of a Random Variable] basat en el teorema de la transformada integral de probabilitat. La funció de densitat de la VA exponencial és

$$f_X(x \mid \beta) = \frac{1}{\beta} e^{-x/\beta},\ 0 \leq x \leq \infty,\ \beta > 0$$

per tant, la seva funció de distribució és

$$F_X(x \mid \beta) = \int_0^x \frac{1}{\beta} e^{-t/\beta} dt = 1 - e^{-x/\beta}$$

on t és una variable muda d'integració. La seva inversa és

$$y = 1 - e^{-x/\beta} \Rightarrow x = -\beta \ln(1 - y).$$

Així, si $y \sim U(0, 1)$, llavors $x \sim Exp(\beta)$ i permetrà escollir l'element de la llista, sent els primers valors els més probables. Recordem que els moments de la VA exponencial són

$$E(X) = \beta,\ V(X) = \beta^2$$

que facilitaran escollir el punt mig de la distribució per incrementar o decrementar la pressió de selecció. En aquesta distribució els valors grans de x són poc probables però no impossibles. Com que els organismes es numeren en la llista de 0 a N, si $x > N$ s'ignorarà el valor i es tornarà a realitzar la selecció[19], el que és equivalent a tallar la distribució per als valors x superiors a N.

En el problema que aquí es tracta, l'avaluació de cada organisme implica l'entrenament complet d'una XN, que és una operació molt costosa. Per això, és important mantenir una pressió de selecció alta mantenint β més proper a 0 que a N. Així es conduirà la cerca més ràpidament cap a un òptim reduint l'exploració de l'espai de cerca. Però s'ha de tenir en compte que una pressió de selecció massa alta tampoc és convenient, tal com s'ha comentat. A la secció 4.5 s'explorarà diversos valors per a aquest paràmetre.

Elitisme

L'elitisme consisteix en conservar els millors organismes de la població, garantint que passen a la propera generació sense cap mutació. La idea és garantir que un cop s'ha trobat un bon organisme, aquest no es perdi i es pugui mantenir entre generacions. La versió de l'AG amb elitisme convergeix probabilisticament a l'òptim global [Saf04]. En aquesta implementació l'elitisme és configurable via el paràmetre `elitism` que indica quants individus passen a la propera generació. Admet el valor zero, permetent no usar elitisme.

Creuament

El creuament consisteix en formar un cromosoma a partir de dos cromosomes progenitors. S'escull aleatòriament la posició d'un bit, anomenada *locus*, i es copien en el nou cromosoma els bits anteriors al locus del primer progenitor, i els bits posteriors del segon progenitor. Aquesta operació també es coneix com *creuament simple*, perquè existeixen altres variants més complexes com els creuaments múltiples. És habitual afegir una probabilitat de creuament, normalment molt propera a 1, que determinarà si dos gens es creuen o no. Quan no es creuen, es copia idèntic el cromosoma d'un dels progenitors i es continua amb la mutació.

Mutació

La mutació consisteix en invertir cada bit del cromosoma amb una probabilitat anomenada *probabilitat de mutació*, i permet ajustar l'exploració de l'espai de cerca. Una probabilitat alta explorarà més solucions, amb la contrapartida d'alentir l'evolució. Una probabilitat baixa accelerarà l'evolució però pot deixar regions de l'espai de cerca sense explorar, i per tant pot ometre un òptim.

4.4 Funció fitness

Segons els objectius del treball, per a l'algorisme genètic és prioritari maximitzar la capacitat d'aprenentatge i generalització de la XN resultant, o equivalentment, minimitzar el seu EQM sobre el conjunt de validació, tal com es defineix a la secció 3.5. També segons els objectius del treball, cal reduir la mida de la xarxa neural i millorar la seva velocitat d'aprenentatge, trobant un compromís entre els diversos objectius. Es tracta doncs d'una optimització multiobjectiu.

[19] Seria un error escollir l'últim element de la llista, ja que implicaria incrementar la seva probabilitat de selecció.

Per mesurar la mida de la xarxa neural s'usarà el seu nombre de paràmetres, entenent com a paràmetres tots els pesos sinàptics i el biaix de cada neurona. Aquesta mesura és adient perquè la complexitat computacional dels recorreguts de la XN (propagació i retropropagació) són lineals respecte del número de paràmetres. A més, el pes del model en memòria és també lineal respecte del nombre de paràmetres[20]. Per tant, no s'afegeix cap restricció a la organització de les capes ocultes en el fitness, l'AG trobarà les millors topologies o en retornarà una qualsevol quan siguin equivalents en rendiment general.

Per mesurar la velocitat d'aprenentatge es comptaran les epochs que han estat necessàries durant l'entrenament de la XN fins arribar a un EQM objectiu sobre el conjunt d'entrenament, que es denota per `tmse` en la implementació de l'AG i es codifica en un dels gens.

És senzill i una pràctica habitual usar una funció lineal per a fitness multiobjectiu. Els diversos termes es poden ponderar amb un factor que opcionalment pot ser configurable [Sri94]. Si aquests factors són més grans que zero, llavors es demostra que les solucions que es trobaran formen part del conjunt de solucions òptimes de Pareto, és a dir, les solucions no dominades [Bra08].

La funció objectiu és

$$\min f = EQM + \psi NP + \omega NE,\ \psi > 0,\ \omega > 0$$

on NP es la mida de la XN mesurada en nombre de paràmetres, i NE el nombre d'epochs requerides per al seu entrenament. Els factors ψ i ω són configurables i es corresponen respecivament amb els paràmetres `npf` i `nef` de la implementació de l'AG.

Amb la representació escollida, tots els termes estan en unitats d'EQM, i els dos factors configurables permeten ajustar el compromís entre els diversos objectius de la següent manera:

- ψ es mesura en unitats EQM/paràmetre, i es pot interpretar com la raó de l'EQM que es sacrifica per paràmetre estalviat. S'ha deixat com a valor per defecte 0.01/10k, és a dir, que l'AG sacrificarà 0.01 unitats d'EQM per cada 10k paràmetres de la XN que es puguin estalviar.
- ω es mesura en unitats EQM/epoch, i s'interpreta anàlogament, com les unitats d'EQM que es sacrifiquen per estalviar una epoch. El valor per defecte és també 0.01/10k, indicant que l'AG sacrificarà 0.01 unitats d'EQM per cada 10k epochs que sigui possible estalviar.

Els valors ψ i ω són en realitat factors de conversió, que transformen respectivament el nombre de paràmetres i el nombre d'epochs en unitats d'EQM. Els valors que s'han deixat per defecte, equilibren la funció fitness prioritzant la minimització de l'EQM, que és un dels objectius fixats. Però fàcilment es pot configurar l'AG per prioritzar les xarxes neurals petites i que aprenen ràpid, escollint per exemple l'assignació $\psi = \omega = 0.01/1k$, que admetrà menys paràmetres i epochs per la mateixa quantitat d'EQM. Els paràmetres ψ i ω es poden interpretar també com el preu o cost de l'EQM. Amb les dades que s'han usat en aquest treball s'ha comprovat que:

- Per dades reals de l'estil dels problemes de PROBEN1, el valor que ha quedat per defecte de 0.01/10k és adient, doncs s'espera una XN menor de 10k paràmetres i epochs.
- Per un problema de dimensions superiors com MNIST, l'assignació que s'ha usat ha estat $\psi = 0.01/10M$ i $\omega = 0.01/10k$, ja que s'espera treballar amb XN grans, encara que no superiors a 1M paràmetres. En canvi, degut al cost d'entrenar la XN amb 60k exemples, cal un aprenentatge ràpid, i per això es valoren 1k epochs en 0.01 EQM.
- Quan s'ha necessitat reduir al màxim la mida de la XN per comprovar límits teòrics, com s'ha fet a la secció 5.1, l'assignació $\psi = \omega = 0.1/1k$ ha estat adient.

Els anteriors punts il·lustren que la optimització de la XN és relativa a cada problema i aplicació, i per tant no es pot prendre una decisió a priori de quins seran els objectius de cada ús que es faci de l'AG. Així es justifica deixar aquests dos paràmetres configurables per l'usuari final.

[20] Un cop generat el model, es podrà guardar amb un pes de NP*sizeof(double).

Evolució dinàmica

Les xarxes neurals utilitzen números aleatoris per inicialitzar els pesos sinàptics i per determinar l'ordre de presentació dels exemples durant una epoch. En conseqüència, dues execucions en dues XN igualment configurades, i sobre unes mateixes dades, retornaran xarxes amb paràmetres (pesos sinàptics) diferents, amb diferents mesures de l'EQM i del nombre d'epochs. En el context de l'algorisme genètic, això implica que un mateix cromosoma generarà organismes que obtindran valors diferents per a la funció de fitness, ja que cada un entrenarà una XN diferent en l'execució del seu mètode lifespan.

Com que, en efecte, els cromosomes es poden repetir durant l'evolució, si no es controlés aquesta situació l'AG prioritzaria l'organisme que obtingués la millor avaluació. Això no seria correcte ja que conduiria l'evolució cap a un resultat esbiaixat. A més, degut a l'elitisme, un organisme sobrevalorat es perpetuaria generació rere generació.

Per resoldre aquest problema, cal calcular la mitja de tots els fitness d'un mateix cromosoma. Una taula hash manté tots els cromosomes que han aparegut durant l'evolució, i es mantenen dades resum per a cada un d'ells, de manera que en tot moment es pot calcular el fitness mig d'un cromosoma donat. En concret, es manté el nombre de repeticions del cromosoma n, i les sumes Σ_{EQM}, Σ_{NP} i Σ_{NE}. El fitness mig d'un cromosoma es calcula a partir d'aquestes dades resum com

$$\bar{f} = \frac{\Sigma_{EQM}}{n} + \psi\frac{\Sigma_{NP}}{n} + \omega\frac{\Sigma_{NE}}{n}$$

Per ordenar la població, l'AG recupera aquesta mitja via el mètode `getFitness()`. A més, es força una reavaluació tornant a cridar el mètode lifespan() de l'organisme quan aquest passa directament a la següent generació per elitisme, evitant així la seva perpetuació si aquest estes sobrevalorat, i també en finalitzar l'evolució, per evitar que un organisme sobrevalorat aparegut durant l'última generació quedi com el millor.

Amb aquest càlcul, el fitness d'un cromosoma anirà variant durant l'evolució conforme aquest es vagi repetint. Així, l'AG no està optimitzant una funció fixa, sinó un procés dinàmic. Aquest és el motiu pel qual la gràfica de l'evolució d'$f_{\max}$ no és monòtona, com passa quan s'optimitzen funcions fixes. Veure la figura 11.

Encara que una de les principals aplicacions dels algorismes genètics és precisament optimitzar funcions (fixes), Holland els va concebre per optimitzar processos canviants, inspirat en la capacitat d'adaptació als canvis de l'evolució natural [Dej92]. Així, l'AG és especialment adient per optimitzar aquest procés dinàmic.

Criteri de parada

Un dels problemes clau dels algorismes genètics és decidir quan parar. Es tracta d'un tema de recerca [Saf04, Gre00] i encara actualment s'utilitzen tècniques que es remonten als orígens dels AG. La opció més simple és deixar l'AG funcionant durant un nombre prefixat de generacions o de temps, en un intent de visitar el màxim espai de cerca possible. Si s'usa elitisme, els millors organismes trobats durant l'evolució es conservaran en la població final.

De Jong descriu en [Dej75] dues mesures del rendiment d'un AG, el rendiment offline i el rendiment online, que poden utilitzar-se com a criteri de parada. Un criteri adequat és parar quan sembla que el rendiment offline s'ha estabilitzat [Gre00]. Les mesures són:

$$f_{\text{off}}(g) = \frac{1}{g}\sum_{j=1}^{g} f_{\max}(j)$$

$$f_{\text{on}}(g) = \frac{1}{g}\sum_{j=1}^{g}\left[\frac{1}{N}\sum_{i=1}^{N} f_i(j)\right]$$

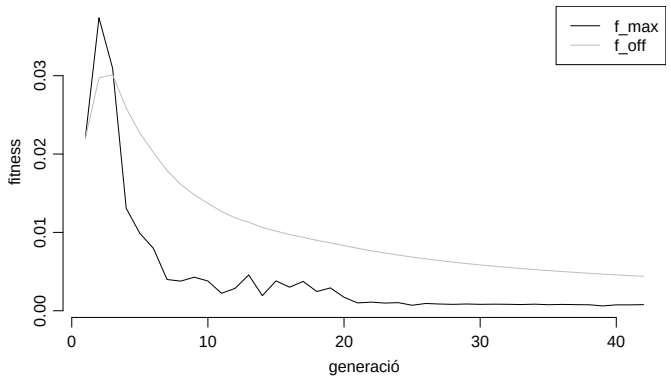

Figura 11: Evolució detinguda amb el criteri de la mitja mòbil f_{off}. L'evolució correspon a la funció trigonomètrica de les dades sintètiques.

on g és la generació actual, N és la mida de la població, $f_{\max}(j)$ és el fitness màxim de la generació j, $f_i(j)$ és el fitness de l'organisme i de la generació j. El rendiment offline f_{off} és una mitja mòbil del fitness del millor organisme. El rendiment online f_{on} és la mitja de tots els valors fitness calculats durant l'evolució de l'AG. La implementació d'aquest treball proporciona totes dues mesures durant l'evolució.

Per detectar la generació en que el rendiment offline s'estabilitza, es guarden els $f_{\text{off}}(j)$ per a les últimes H generacions $j = g - H, \ldots, g$, i es calcula el coeficient de variació $cv = \bar{x}/s$, on $\bar{x}$ és la mitja dels $f_{\text{off}}(j)$, i s la seva desviació típica. L'avantatge d'usar el coeficient de variació en comptes de la mateixa desviació típica, és que cv és adimensional i no es veurà afectat per canvis en la magnitud de la funció fitness, que en efecte pot variar segons el problema o segons els paràmetres `npf` i `nef` que permeten ajustar la funció fitness per prioritzar els seus objectius.

Quan cv sigui menor que un valor donat l'algorisme pararà. Ara bé, cal usar aquesta funcionalitat amb precaució, ja que és possible que l'AG estigui temporalment estancat però no es trobi encara en un punt òptim, tal com s'adverteix a [Gre00]. Unes mesures que han donat bons resultats durant les proves han estat $H = 7$ i $cv = 0.002$, que s'interpreta com que f_{off} ha variat molt poc en les 7 generacions precedents. La figura 11 mostra una evolució típica detinguda amb aquest criteri de parada.

Finalment, indicar que com que l'AG és un algorisme general, no es pot usar el valor de fitness com a criteri de parada i per això s'usa f_{off}. Si per a un problema concret s'opta per no usar el criteri de parada, existeix la possibilitat d'executar l'AG indefinidament, i usar el fitness o qualsevol dels seus tres termes per parar l'evolució manualment, quan s'hagi aconseguit un valor que sigui satisfactori per al problema en qüestió.

Multimodalitat

Una funció multimodal no té un únic òptim (màxim o mínim), sinó diversos iguals o molt propers. El fitness que s'ha descrit prèviament és una funció lineal que òbviament no és multimodal. No obstant, la funció que l'AG optimitza no és la funció de fitness anterior, sinó la funció que transforma els gens en el fitness final, i aquesta és una funció composta de l'algorisme genètic g, que sí pot ser multimodal, en el fitness f descrit anteriorment. La composició és

$$\mathit{fitness} = f \circ g = f(g(tmse, \lambda, \mu, nh, hs))$$

amb

$$\begin{aligned} g\colon \mathbb{R}^5 &\to \mathbb{R}^3 \\ (tmse, \lambda, \mu, nh, hs) &\mapsto (mse, np, ne) \end{aligned}$$

$$\begin{aligned} f\colon \mathbb{R}^3 &\to \mathbb{R} \\ (mse, np, ne) &\mapsto \mathit{fitness} \end{aligned}$$

on $(tmse, \lambda, \mu, nh, hs)$ és el vector format pels gens descodificats, i (mse, np, ne) és el vector que s'obté en avaluar la XN de l'organisme.

D'aquesta manera, la funció $f \circ g$ és una funció que pot ser multimodal segons el dataset que es processi. I llavors l'AG pot retornar organismes amb valors de fitness aproximadament iguals per a organismes molt diferents. En la interpretació dels resultats de l'AG, és important recordar que no és correcte fer una mitja de les variables originals, els vectors $(tmse, \lambda, \mu, nh, hs)$, encara que els seus valors de fitness siguin propers, ja que al ser la funció $f \circ g$ multimodal, el fitness de la mitja no ha d'estar en un punt òptim.

4.5 Ajust de l'algorisme genètic

Amb la implementació realitzada, queden lliures els següents paràmetres de l'AG que cal ajustar:

- Descendència: nombre d'organismes que s'obtenen a partir de la generació actual.
- Elitisme: nombre d'organismes que passen directament a la propera generació.
- Pressió de selecció.
- Probabilitat de creuament.
- Probabilitat de mutació.

Idealment, aquests paràmetres únicament afecten al rendiment del propi algorisme genètic, no a la xarxa neural resultant, ja que la funció de fitness es manté. Ara bé, poden existir valors que impedeixin l'evolució o la ralentitzin i per això és important estudiar-los. Interessarà trobar uns valors per aquests paràmetres, tals que l'algorisme genètic sigui eficient amb el major nombre de conjunts de dades. Amb aquest objectiu es realitzaran diverses proves amb diferents conjunt de dades, variant els paràmetres anteriors i mesurant:

- El fitness obtingut en el millor organisme de la última generació.
- El nombre d'avaluacions de l'organisme realitzades durant l'evolució de l'AG, i.e. les crides al mètode `lifespan`, que és l'operació més costosa de l'AG i el valor que cal minimitzar.

Aquestes mesures són contraries, per una banda interessa reduir el nombre de crides a lifespan, però això produeix un increment en el fitness el qual interessa reduir per arribar a una XN propera a l'òptima. Per resoldre-ho s'utilitzarà Pareto-rank [Fon93, Col99], que trobarà les configuracions no dominades i permetrà escollir la que presenti un millor compromis entre lifespans i fitness.

Mètode seguit

Per trobar una configuració adient per a l'AG es realitzarà un *grid search* no exhaustiu[21] sobre diversos valors per als paràmetres esmentats. La taula 8 mostra els valors que s'examinaran per a cada paràmetre, i en negreta apareix la configuració base, que ha donat resultats acceptables en proves prèvies. La resta de valors són múltiples i submúltiples de la configuració base.

	Paràmetre	Valors		
D	Descendència	8	**16**	32
E	Elitisme	1	**3**	9
S	Pressió de selecció	2	**4**	8
C	Probabilitat de creuament	0.90	**0.95**	0.99
M	Probabilitat de mutació	0.01	**0.05**	0.10

Taula 8: Taula de variacions per als valors de la configuració de l'algorisme genètic. En negreta els valors que formaran la configuració base.

Per a cada configuració i per a cada dataset, s'obtindrà una observació per al fitness i una altre per al nombre de crides a lifespan. Com que l'AG pot retornar resultats diferents en cada execució, per obtenir el valor d'una observació, s'executa l'AG tres vegades i es calcula la mitja de les tres execucions. Els valors obtinguts es mostren a les taules de resultats 10 i 11. A continuació es calcula la mitja per a tots els datasets d'una mateixa configuració, que es mostra a la última columna de les taules de resultats.

En una primera iteració, s'examina la configuració base i totes les configuracions que resulten de modificar un únic paràmetre sobre aquesta, són les files 1–11 de les taules 10 i 11. Amb els resultats obtinguts s'elabora el Pareto-rank per trobar les configuracions no dominades, i s'obté que:

[21] Amb la discretització de 3 valors per dimensió escollida a la taula 8, la versió exhaustiva hauria d'avaluar 243 configuracions. Si l'execució per 11 configuracions ha requerit unes 10h, serien necessaris uns 9 dies per al *grid search* exhaustiu.

- Descendència: Convé incrementar-la per millorar el fitness, un valor entre 16 i 32 sembla adient. S'escull D=24.
- Elitisme: El seu increment fa augmentar el nombre de lifespans. El valor extrem 9 és excessiu i inclús ha empitjorat el fitness. S'escull E=2.
- Pressió de selecció: valors entre 4 i 8 donen resultats propers, valors massa baixos com 2 incrementen excessivament el fitness. Es prenen S=6,8.
- Probabilitat de creuament: convé que sigui propera a 1. Es deixa C=0.99.
- Probabilitat de mutació: convenen valors entre 0.05 i 0.10. Valors massa petits incrementen excessivament el fitness. Es pren un valor mig M=0.07.

En una segona iteració s'afegeix les configuracions D=24; E=2; S=6,8; C=0.99; M=0.07, que corresponen a les files 12–13 de les taules de resultats, i a continuació es proven diverses configuracions addicionals intentant millorar els resultats, que es mostren a les files 14–17.

Consideracions addicionals

- Interessa que l'algorisme pari per convergència i no per esgotar el màxim de generacions, per tant es configura un nombre de generacions màximes prou alt amb `maxge=200`. El criteri de parada és el descrit en la secció 4.4, amb els valors $H = 7$ i $cv = 0.002$.
- Els resultats són dependents del criteri de parada que s'està usant. Un altre criteri de parada pot donar resultats diferents.
- Com a funció d'activació de la capa de sortida de la XN, es prendrà la que correspongui als rangs de les variables resposta de cada conjunt de dades.
- Els valors obtinguts en aquest experiment, s'utilitzaran com a valors per defecte en l'algorisme genètic, deixant oberta l'opció de que l'usuari pugui canviar-los amb facilitat.

Resultats

A continuació es mostren i s'interpreten els resultats obtinguts. Les taules 10 i 11 mostren els resultats numèrics. Cada fila correspon a una mateixa configuració. Les cinc primeres columnes de cada taula indiquen el valor dels paràmetres, les següents columnes corresponen a una per dataset, finalment la última columna mostra la mitja per a tots els datasets. La taula 9 mostra el Pareto-rank, on les configuracions no dominades valen rank=0. La figura 12 mostra el Pareto-front amb les configuracions no dominades unides per una línia, que facilita l'elecció d'una configuració. De l'anàlisi es poden extreure les següents conclusions:

- Més execucions de lifespan impliquen menor fitness i per tant millor xarxa neural.
- Les configuracions que obtenen millor fitness es corresponen amb les que generen poblacions més grans, incrementant el nombre de descendents.
- Valors d'elitisme com 2, 3 i 5 han donat bons resultats. Valors massa alts com E=9 han empitjorat el fitness.
- Si es vol reduir el temps d'execució de l'AG a costa d'empitjorar el fitness, es poden provar configuracions entre 4, 11, 7 i 12.
- Els pitjors fitness s'han produït per reduir excessivament la mida de la població a 8 organismes, per reduir la probabilitat de mutació a 0.01, i per incrementar la pressió de selecció amb $\beta = 2$.
- La gràfica 12 té forma l'L, per sota de 1200 lifespans el fitness empitjora. Les configuracions del punt anterior indiquen els límits dels paràmetres de l'AG.

- Si es disposa de temps suficient, pot ser interessant executar l'AG el màxim temps possible i amb una descendència el més gran possible, potser desactivant el criteri de parada per assegurar que es superen les 1200 execucions de lifespan.
- Observant la taula 10 per columnes, es veu que el fitness obtingut es depenent del problema (conjunt de dades), i que aquest no es veu substancialment afectat per la configuració de l'AG, excepte en els casos que coincideixen amb les pitjors configuracions 2 i 10.

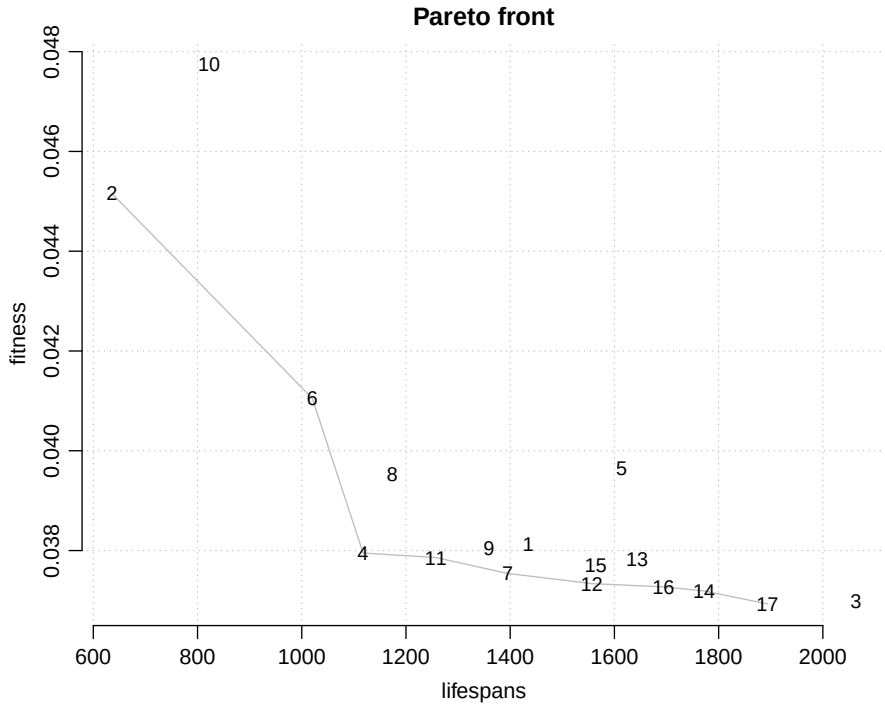

Figura 12: La línia gris mostra el Pareto-front, unint les configuracions no dominades (rank=0) que ofereixen les millors combinacions de fitness i lifespans. Cada punt numerat correspon a una configuració, que pot localitzar-se fàcilment a les taules de resultats.

	lifespans	fitness	rank
1	1435	0.038129	4
2	636	0.045158	0
3	2064	0.036984	1
4	1118	0.037950	0
5	1614	0.039645	8
6	1020	0.041049	0
7	1395	0.037542	0
8	1174	0.039526	1
9	1360	0.038052	2
10	822	0.047757	1
11	1257	0.037860	0
12	1557	0.037337	0
13	1643	0.037832	3
14	1772	0.037190	0
15	1564	0.037706	2
16	1694	0.037272	0
17	1894	0.036930	0

Taula 9: Pareto-rank per a les configuracions provades en l'ajust de l'AG.

	configuració					dataset							
	Descendència	Elitisme	Selecció	Creuament	Mutació	sinfunc	discfunc	building1	flare1	card1	heart1	thyroid1	mitja
1	16	3	4	0.95	0.05	0.9	29.2	7.1	3.5	77.6	129.5	19.1	38.1
2	8	3	4	0.95	0.05	26.5	33.3	8.0	3.6	80.2	131.3	33.4	45.2
3	32	3	4	0.95	0.05	0.8	29.3	7.0	3.5	75.3	129.9	13.1	37.0
4	16	1	4	0.95	0.05	1.0	31.1	7.7	3.5	76.2	129.7	16.4	38.0
5	16	9	4	0.95	0.05	0.9	30.7	7.9	3.5	76.3	129.3	28.9	39.6
6	16	3	2	0.95	0.05	1.0	31.2	7.1	3.5	75.9	130.7	38.0	41.0
7	16	3	8	0.95	0.05	1.0	30.3	6.5	3.5	76.2	129.7	15.5	37.5
8	16	3	4	0.90	0.05	1.1	29.9	7.1	3.5	76.5	129.8	28.8	39.5
9	16	3	4	0.99	0.05	1.0	32.4	7.4	3.5	77.5	129.3	15.2	38.1
10	16	3	4	0.95	0.01	51.0	31.5	7.1	3.5	77.3	131.4	32.5	47.8
11	16	3	4	0.95	0.10	1.2	31.4	7.1	3.6	75.8	130.0	16.0	37.9
12	24	2	6	0.99	0.07	1.0	29.5	7.4	3.5	74.5	130.1	15.3	37.3
13	24	2	8	0.99	0.07	0.9	32.1	7.4	3.5	76.6	128.8	15.6	37.8
14	24	2	6	0.99	0.05	1.1	30.1	6.9	3.4	76.2	129.0	13.6	37.2
15	24	2	6	0.99	0.09	1.1	31.3	7.6	3.5	75.3	129.1	16.1	37.7
16	24	3	6	0.99	0.07	1.0	29.6	6.9	3.5	76.2	129.3	14.5	37.3
17	24	5	6	0.99	0.07	1.0	28.5	6.9	3.4	75.7	127.8	15.2	36.9

Taula 10: Fitness de la millor XN trobada en l'evolució. Els valors de fitness s'han multiplicat per 10^3 per millorar la claredat de la taula, estan doncs en unitats mEQM.

	configuració					dataset							
	Descendència	Elitisme	Selecció	Creuament	Mutació	sinfunc	discfunc	building1	flare1	card1	heart1	thyroid1	mitja
1	16	3	4	0.95	0.05	3816	1941	833	415	364	333	2340	1435
2	8	3	4	0.95	0.05	1585	1090	298	331	257	279	609	636
3	32	3	4	0.95	0.05	7032	2412	685	592	382	510	2832	2064
4	16	1	4	0.95	0.05	3416	1546	594	226	379	316	1348	1118
5	16	9	4	0.95	0.05	5016	2274	791	466	474	483	1791	1614
6	16	3	2	0.95	0.05	3816	1409	497	377	409	320	314	1020
7	16	3	8	0.95	0.05	3816	2157	839	314	383	345	1910	1395
8	16	3	4	0.90	0.05	3816	1745	643	352	428	307	928	1174
9	16	3	4	0.99	0.05	3816	1447	459	345	282	339	2828	1360
10	16	3	4	0.95	0.01	1561	1865	523	352	377	402	675	822
11	16	3	4	0.95	0.10	3816	1428	820	390	345	288	1713	1257
12	24	2	6	0.99	0.07	5224	2130	847	414	423	414	1445	1557
13	24	2	8	0.99	0.07	5224	2286	951	449	405	553	1636	1643
14	24	2	6	0.99	0.05	5224	2633	856	475	327	423	2468	1772
15	24	2	6	0.99	0.09	5224	1272	995	509	388	431	2130	1564
16	24	3	6	0.99	0.07	5424	1941	942	447	330	510	2265	1694
17	24	5	6	0.99	0.07	5824	2296	865	498	343	546	2885	1894

Taula 11: Nombre d'execucions del mètode lifespan.

4.6 Detalls de la implementació

La figura 13 mostra les components més importants de l'algorisme genètic i la relació amb la xarxa neural. La classe GenAlg implementa l'AG en el seu mètode evolve(), manté una població d'organismes i és la responsable de l'operació genètica de selecció. La classe Organism manté un cromosoma i és responsable de les operacions genètiques de creuament i mutació. A més, realitza l'entrenament d'una xarxa neural de classe MlpBp en el mètode lifespan() i calcula el seu fitness. El cromosoma descodificat i el fitness del l'organisme es poden consultar amb els getters getDecodedChromosome() i getFitness(), la classe GenAlg els utilitzarà durant l'evolució per ordenar els organismes i per retornar els hiperparàmetres del millor.

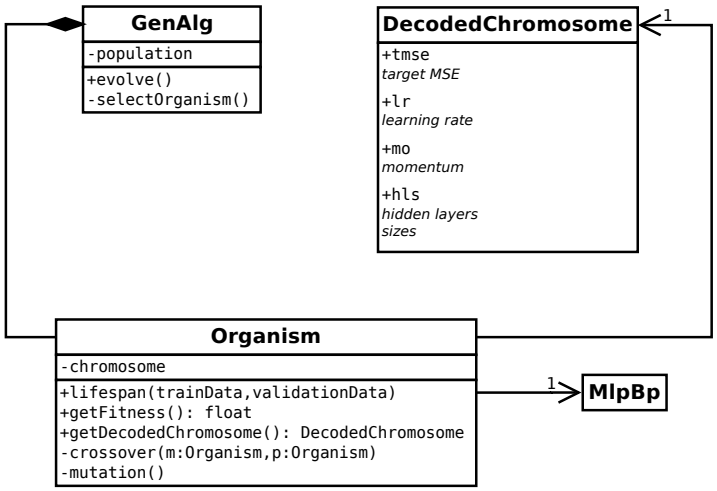

Figura 13: Esquema UML amb els principals conceptes de l'AG.

Ús de l'algorisme genètic des de C++

L'algorisme genètic es troba encapsulat en la classe GenAlg, que s'ocuparà de crear i mantenir la població d'organismes. La classe Organism encapsula un organisme, amb la seva xarxa neural i el càlcul del fitness. La gestió dels organismes la realitza íntegrament GenAlg per la qual cosa no caldrà instanciar-la directament. Els paràmetres de l'únic constructor de la classe GenAlg són:

- `uint maxge`: El nombre màxim de generacions a executar durant l'evolució.
- `uint osiz`: Offspring size, la mida de la descendència.
- `uint esiz`: Elitism size, el nombre d'organismes que passen directament a la propera generació.
- `double sbeta`: El paràmetre β que permet regular la pressió de selecció.
- `bool foffstop`: Activar o no el criteri de parada via `f_off`.
- `uint foffhs`: Mida de l'història per al criteri de parada.
- `double fofftcv`: Coeficient de variació cv on parar.
- `uint nh_bits`: Nombre de bits per al gen que regula el nombre de capes ocultes.
- `uint min_hls`: Nombre de neurones de la menor capa oculta.
- `uint max_hls`: Nombre de neurones de la major capa oculta.
- `bool rec`: La xarxa ha de ser rectangular? Una XN és rectangular si totes les capes ocultes tenen el mateix número de neurones.

- `double cp`: Probabilitat de creuament.
- `double mp`: Probabilitat de mutació
- `double pf`: Factor ψ per al nombre de paràmetres de la funció fitness.
- `double ef`: Factor ω per al nombre d'epochs de la funció fitness.
- `uint maxep`: Nombre màxim d'epoch a executar en les xarxes neurals dels organismes.
- `ActivatorEnum at`: Funció d'activació de la última capa de les xarxes neurals dels organismes.

A continuació es presenta un exemple simplificat del seu ús. A la unitat de compilació rentry.cpp es pot consultar un exemple funcional complet.

```
GenAlg ga(max_generations, offspring_size, elitism_size, selection_beta,
          stop, stophs, stopcv, hlbits, minhls, maxhls, rectangular,
          crossover_pr, mutation_pr, pf, ef, maxep, lact);
```

un cop es disposa d'un objecte, s'inicia l'evolució cridant al mètode `evolve`, que rep com arguments els conjunts d'entrenament i de validació. Per cada conjunt es passa un enter amb el nombre d'exemples, i dues referències a `std::vector<double*>`, una per les variables explicatives i l'altre per les respostes.

```
int in_size  = ...
int out_size = ...
vector<double*> tra_in  = ...
vector<double*> tra_out = ...
vector<double*> val_in  = ...
vector<double*> val_out = ...

ga.evolve(in_size, out_size, tra_in, tra_out, val_in, val_out);
```

En finalitzar l'execució, es podran recuperar diversos resultats de l'evolució, entre ells la configuració del millor organisme de la població final:

```
Organism::DecodedChromosome best_config = ga.getBestConfig();
```

Ús de l'algorisme genètic des d'R

Els paràmetres són els mateixos que al codi C++. Des d'R es pot obtenir una explicació de cada un d'ells usant el sistema d'ajut d'R amb `?gann`. La sintaxi de l'AG a R és:

```
evol <- gann(tx, ty, vx, vy,
             maxge=50, offspring=24, elitism=3, sbeta=6, cp=0.99, mp=0.07,
             stop=TRUE, stophs=7, stopcv=0.002,
             npf=0.01/1E4, nef=0.01/1E4,
             maxep=2000, lact="linear",
             hlbits=3, minhls=2, maxhls=32, rec=FALSE)
```

a continuació es descriuen breument:

- `tx, ty, vx, vy` són els conjunts d'entrenament `t*`, i validació `v*`, separant les variables explicatives `*x` i respostes `*y`.
- `maxge` determina el nombre màxim de generacions a executar.
- `offspring` determina quants individus es crearan per generació.
- `elitism` determina quants organismes passen directament a la propera generació, s'afegiran als anteriors.
- `sbeta` assigna valor al paràmetre β que regula la pressió de selecció.
- `cp` i `mp` configuren les probabilitats de creuament i mutació.

- `stop, stophs, stopcv` configuren el criteri de parada.
- `npf, nef=` correspnen als paràmetres ψ i ω de la funció fitness.
- `maxep, lact` es passen directament a les xarxes neurals de cada organisme.
- `hlbits, minhls, maxhls, rec` determinen els límits de la topologia de les xarxes neurals dels organismes. hlbits són els bits que s'usaran per al gen del número de capes ocultes, minhls i maxhls determinen el nombre mínim i màxim de neurones per cada capa oculta, i rec determina si la topologia de les capes ocultes serà rectangular, i.e. que totes les capes són de la mateixa mida.

L'objecte R que retorna `gann` conté informació sobre l'evolució. Sobre aquest objecte es poden aplicar mètodes sobrecarregats com

```
print(evol)  # mostra informació de l'evolució
plot(evol)   # genera un gràfic amb les mètriques f_max i f_off
```

o recuperar els hiperparàmtres de la XN del millor organisme de la població final amb:

```
evol$besttmse
evol$bestlr
evol$bestmo
evol$besthls
```

5 Resultats

Amb l'algorisme genètic ajustat a la secció 4.5, ja podem passar a usar-lo amb l'objectiu per al que ha estat dissenyat: configurar xarxes neurals. Els conjunts de dades escollits i descrits a la secció 2.3 han estat provats i el seu rendiment documentat en diversos treballs, tant amb xarxes neurals com amb altres algorismes d'aprenentatge automàtic. Així, es disposa de material suficient per comparar el rendiment de les XN configurades per l'AG.

Cal destacar que, com que la funció fitness pot ser multimodal, dues execucions de l'AG poden trobar resultats diferents, i en alguns casos l'AG pot estancar-se en un òptim local. És habitual realitzar diverses execucions d'un AG i escollir els organismes amb millor fitness [Col99], sempre tenint en compte que no és correcte fer una mitja dels hiperparàmetres obtinguts; veure la secció 4.4.

Quan el rendiment i mida de les xarxes neurals obtingudes amb l'AG és similar a l'obtingut en altres treballs, el que cal valorar és que gràcies a l'AG, no ha estat necessari buscar manualment la configuració adient per a la XN, sinó que s'ha obtingut el mateix rendiment de manera automàtica.

5.1 Conjunts de dades sintètiques

Funcions lògiques

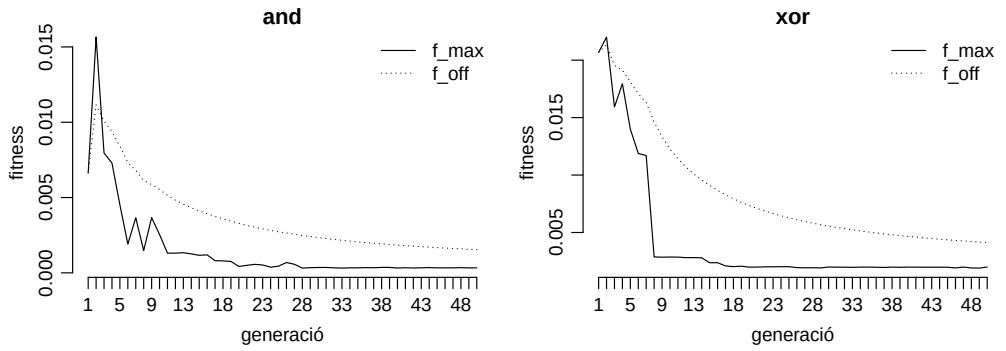

Figura 14: Evolució de l'AG sobre funcions lògiques.

Les funcions lògiques AND i XOR són interessants perquè varen constituir un dels primers problemes de xarxes neurals. La funció AND és un problema linealment separable, i per tant la pot aprendre un perceptró sense cap capa oculta. En canvi, la funció XOR no és linealment separable i necessita com a mínim una capa oculta amb 2 neurones [Bis06, Roj96, Rus10].

L'AG arriba a aquesta mateixa conclusió retornant una configuració sense cap capa oculta per al problema AND, i una configuració amb 2 capes ocultes de 2 neurones per al problema XOR. En aquest últim cas queda patent que l'AG no retorna la solució òptima sinó una aproximació prou bona. Cal tenir present que aquest AG optimitza per nombre de paràmetres i no de capes o neurones. La XN mínima teòrica es forma de 9 paràmetres (6 pesos sinàptics + 3 biaixos), la XN que ha trobat l'AG es forma de 15 paràmetres (10 pesos i 5 biaixos).

Per obtenir aquest resultat s'ha modificat el paràmetre `npf` (number of parameters factor) per alterar la prioritat de la funció fitness, incrementant el cost dels paràmetres respecte de l'EQM, prioritzant així les XN més petites durant l'evolució de l'AG.

dataset	neurones ocultes	topologia	λ	μ	tmse
and		2–1	1.116	0.997	0.000046
xor	2-2	2-2-2-1	0.784	0.730	0.000084

Taula 12: Resultats de l'evolució amb funcions lògiques.

Funcions matemàtiques

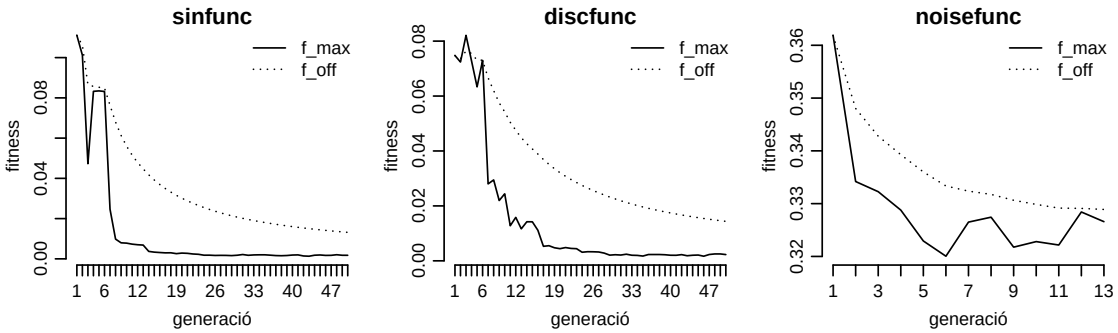

Figura 15: Evolució de l'AG sobre funcions matemàtiques.

El teorema de Cybenko afirma que una XN amb una única capa oculta permet aproximar quasevol funció contínua, per representar funcions discontínues caldran capes ocultes addicionals. Si es configura l'AG de manera que el fitness prioritzi la reducció de la mida de la XN, llavors es troben les topologies esperades per als datasets sintètics *sinfunc* i *discfunc*, que són funcions contínua i discontinua respectivament, com es comprova a la taula 13.

Per al dataset *noisefunc* de dades soroll, l'AG no és capaç de trobar una XN que convergeixi, com es pot veure en última fila de la taula 14. La figura 15 mostra les evolucions de l'AG per a tots tres datasets, i a l'última gràfica es veu que l'evolució de *nosiefunc* s'ha estancat en un fitness alt.

dataset	neurones ocultes	topologia	λ	μ	tmse
sinfunc	8	1-8-1	0.182	0.038	0.001068
discfunc	3-5-4	1-3-5-4-1	0.068	0.058	0.001816
noisefunc	6-23	1-6-23-1	0.136	0.424	0.188205

Taula 13: Resultats de l'evolució amb funcions matemàtiques.

	EQM entr		EQM val		EQM test		epochs	
dataset	$\bar{x}$	s	$\bar{x}$	s	$\bar{x}$	s	$\bar{x}$	s
sinfunc	0.172	0.102	0.198	0.108	0.185	0.090	503	216
discfunc	0.212	0.065	0.189	0.063	0.112	0.076	156	23
noisefunc	41.473	4.751	38.292	7.065	33.324	7.663	5000	0

Taula 14: Avaluació de la XN resultant de l'evolució amb funcions matemàtiques.
Els EQM es mostren multiplicats per 100.

5.2 Xarxes neurals saturades

En aquest apartat es realitzarà una prova per verificar que l'algorisme genètic tracta correctament les xarxes neurals amb saturació. S'utilitzarà un conjunt de dades que no compleix cap de les recomanacions per a les variables explicatives ni per a les respostes. Les entrades no estan normalitzades ni estandaritzades, són sempre positives i surten de l'interval recomanat. Les sortides també surten de l'interval $[-1, 1]$ fet que obliga a usar la funció d'activació lineal a l'última capa.

Igual que s'ha fet en les altres proves, es parteix el dataset original *satfunc* en tres, un subconjunt per l'entrenament de la XN, l'altre per la validació a l'AG, i finalment un altre per provar que el model és correcte. La figura 16 mostra els resultats, on es pot comprovar que l'AG ha estat capaç de trobar una configuració de XN que no es satura, a pesar d'haver trobat moltes que es saturen durant l'evolució. Per permetre l'evolució, l'AG detecta[22] els organismes amb una XN saturada i els

[22]L'AG considera que una XN està saturada únicament quan una o més de les seves sortides retornen ∞, que es tradueix en un EQM=NaN.

penalitza amb un error de generalització de 99, afavorint així la supervivència dels que no tenen aquest problema. Es poden veure aquests valors en la sortida de l'evolució, que tindrà un aspecte com el següent. La sortida completa d'aquesta evolució es pot trobar en el producte del treball:

```
Generation 12:

  fitness       mse      fnp   np      fne   ne | ann config
99.000196 99.000000 0.000192  192 0.000004    4 | tmse=0.0373, lr=0.1897, mo=0.9138, hls=c(11, 13)
 2.142239  2.138790 0.001449 1449 0.002000 2000 | tmse=0.0686, lr=0.0630, mo=0.7965, hls=c(25, 27, 24)
99.001344 99.000000 0.001343 1343 0.000001    1 | tmse=0.0066, lr=0.5102, mo=0.5387, hls=c(25, 26, 22)
 0.187564  0.185921 0.001022 1022 0.000621  621 | tmse=0.0607, lr=0.0054, mo=0.2341, hls=c(10, 31, 20)
 0.039503  0.037867 0.000237  237 0.001399 1399 | tmse=0.0452, lr=0.0023, mo=0.6632, hls=c(6, 28)
99.000838 99.000000 0.000837  837 0.000001    1 | tmse=0.0418, lr=0.5026, mo=0.6784, hls=c(26, 28)
...
```

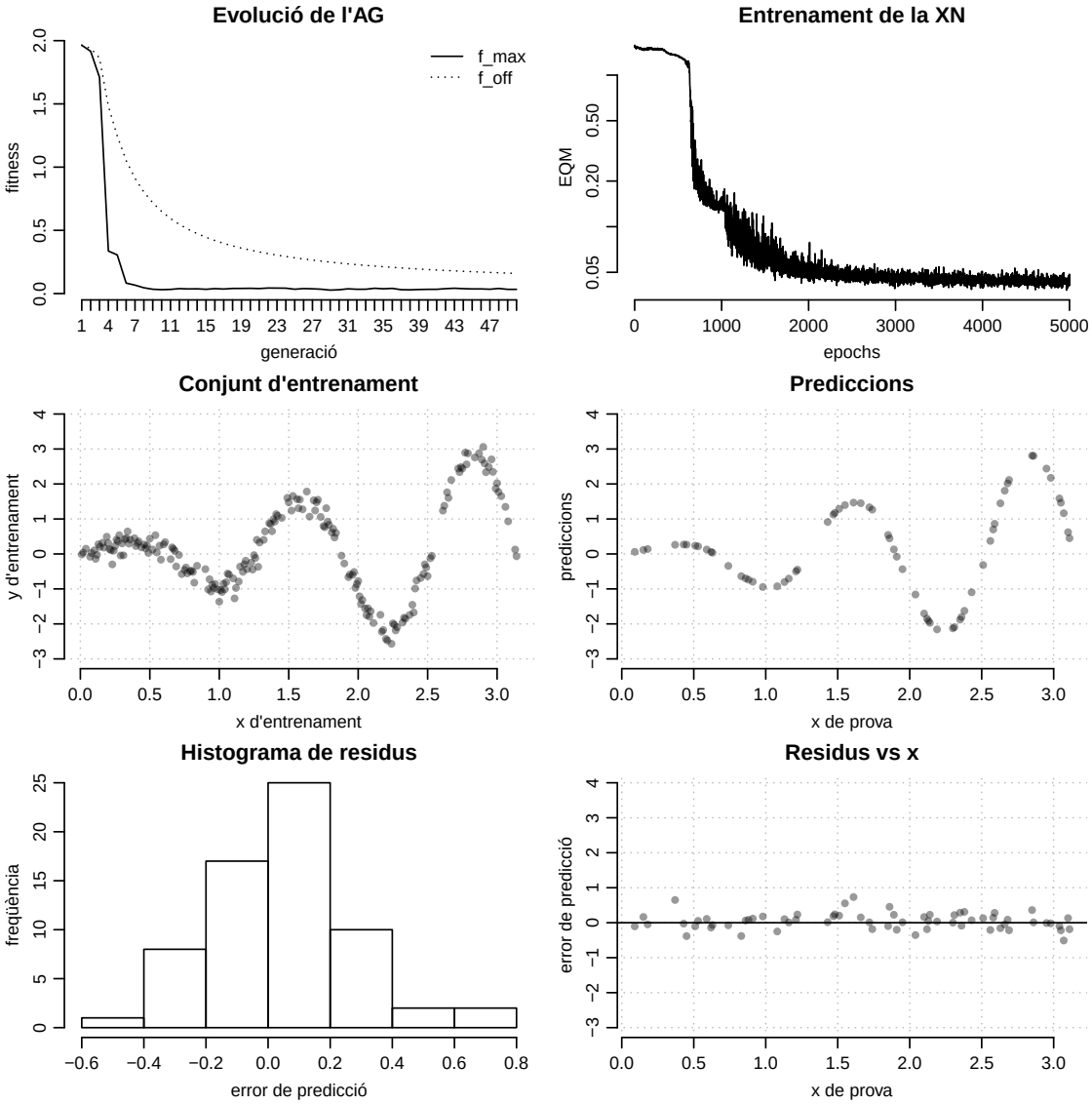

Figura 16: Avaluació del model resultant de la millor XN trobada per l'AG. L'ajust és correcte, els residus segueixen una distribució normal i només representen el soroll de les dades, no segueixen cap patró.

5.3 Conjunts de dades de PROBEN1

A continuació es presenten els resultats de l'algorisme genètic sobre els diversos conjunts de dades de PROBEN1 que s'han presentat a la secció 2.3. La figura 17 mostra els gràfics d'evolució i les taules 15 i 16 els resultats.

S'ha deixat la configuració per defecte de l'AG, i s'ha escollit el millor organisme de l'última generació. Aquest organisme correspon a una configuració de xarxa neural que es mostra en la taula 15. A continuació s'ha creat una xarxa neural amb aquesta configuració i s'ha entrenat 10 vegades. S'ha calculat la mitja i desviació típica de l'EQM dels 10 entrenaments, per al conjunt d'entrenament, per al de validació i per al de prova. També s'ha calculat la mitja i desviació típica del nombre d'epochs requerides per a l'entrenament. Els resultats es mostren a la taula 16.

Resultats obtinguts amb l'ús de l'AG

- La millora més destacable és que no ha estat necessària cap experimentació ni prova. Únicament s'ha executat l'AG i s'ha entrenat una XN configurada tal com l'AG ha recomanat. De fet, tot el procés per obtenir les dades d'aquesta secció es troba automatitzat en un script R en el producte del treball.
- Quan es va provar la XN a la secció 3.7 es va usar una topologia 8-8 per a les capes ocultes. Gracies a l'AG es pot comprovar que era excessiva per a la majoria d'aquests datasets.
- La precisió de les XN resultants és molt similar als resultats que obté [Pre94] amb les mateixes dades, però el nombre d'epochs executades és molt menor en els problemes de regressió i el problema de classificació *card* configurats amb l'AG. En aquests casos l'AG ha optimitzat la velocitat d'aprenentatge al no poder reduir més l'EQM ni la mida.
- Per contra, el problema *thyroid* està clarament per sota del rendiment que obté [Pre94]. És possible que aquest dataset requereixi una configuració diferent de l'AG per obtenir resultats equivalents, o simplement un major recorregut de l'espai de cerca, doncs com s'observa a les gràfiques de la figura 17, ha estotat les 50 generacions màximes, el que pot significar que l'AG podria haver trobat una configuració millor amb més generacions.
- Els resultats obtinguts depenen del criteri de parada de l'AG, i podrien millorar si aquest es desactives.

Nota sobre els problemes de classificació

Els problemes de classificació *card*, *heart* i *thyroid* han obtingut uns valors molt alts d'EQM en comparació amb els de regressió *building* i *flare*, però aquest resultat necessita interpretació. No es pot comparar l'EQM d'un problema de regressió amb un de classificació. Si be l'EQM serveix per al correcte entrenament de la XN independentment del tipus de problema, com a mesura d'error en els problemes de classificació cal usar la matriu de confusió i les mètriques que s'en deriven.

En els problemes de classificació les variables resposta estan polaritzades en valors propers als extrems de l'interval de sortida. Cada error suma a l'EQM una part important de l'interval, incrementant l'EQM molt més que en els problemes de regressió, on la distància entre el valor esperat i l'obtingut serà petita. Com que l'EQM eleva al quadrat aquests errors magnificant encara més aquest fet, el resultat és que un problema de classificació pot tenir un EQM molt alt i en canvi ser un bon classificador.

En aquest treball l'objectiu no és l'anàlisi de les dades, sinó mesurar el rendiment de l'AG i de la XN. Per això no s'ha realitzat una anàlisi més acurada dels anteriors conjunts de classificació. Tanmateix, a la propera secció 5.4, on es tracten les dades MNIST, s'analitza el resultat obtingut amb més profunditat i es pot comprovar aquest raonament.

dataset	neurones ocultes	topologia	λ	μ	tmse
building1	8	14-8-3	1.699	0.199	0.002335
building2	4	14-4-3	0.952	0.296	0.031586
building3	5	14-5-3	0.212	0.377	0.002876
flare1		24–3	1.732	0.564	0.192332
flare2		24–3	1.948	0.472	0.475479
flare3		24–3	1.585	0.525	0.329824
card1		51–2	0.227	0.279	0.099466
card2		51–2	0.630	0.329	0.276913
card3		51–2	0.704	0.351	0.120081
heart1		35–2	0.021	0.291	0.104631
heart2	29	35-29-2	0.030	0.415	0.091279
heart3		35–2	0.023	0.758	0.102182
thyroid1	2-19	21-2-19-3	0.074	0.024	0.000061
thyroid2	2	21-2-3	0.366	0.021	0.011200
thyroid3	2	21-2-3	0.137	0.103	0.011620

Taula 15: Resultats de l'evolució amb dades de PROBEN1. Les topologies que no contenen cap neurona oculta, són degudes a que el problema és linealment separable i les capes ocultes no són necessàries. Conseqüentment, l'AG les ha descartat.

	EQM entr		EQM val		EQM test		epochs	
dataset	$\bar{x}$	s	$\bar{x}$	s	$\bar{x}$	s	$\bar{x}$	s
building1	0.192	0.033	0.825	0.155	0.738	0.180	2	0
building2	0.407	0.045	0.445	0.045	0.413	0.044	1	0
building3	0.284	0.004	0.285	0.004	0.284	0.006	76	19
flare1	0.416	0.010	0.361	0.011	0.560	0.026	1	0
flare2	0.509	0.014	0.469	0.022	0.268	0.009	1	0
flare3	0.442	0.021	0.511	0.028	0.374	0.023	1	0
card1	9.640	0.396	7.754	0.239	10.458	0.520	11	1
card2	9.138	1.909	10.875	2.546	13.842	1.282	1	0
card3	10.105	1.170	7.930	1.112	13.556	1.552	2	0
heart1	10.393	0.008	12.844	0.034	14.429	0.019	194	8
heart2	8.852	0.623	11.978	0.554	13.946	0.710	65	5
heart3	10.222	0.119	10.717	0.340	16.833	0.241	110	14
thyroid1	0.889	0.229	1.352	0.238	1.509	0.331	5000	0
thyroid2	1.254	0.319	1.214	0.305	1.302	0.323	2656	313
thyroid3	1.209	0.347	1.158	0.365	1.728	0.430	1736	101

Taula 16: Avaluació de la XN resultant de l'evolució amb dades de PROBEN1. Els EQM es mostren en multiplicats per 100, com a [Pre94].

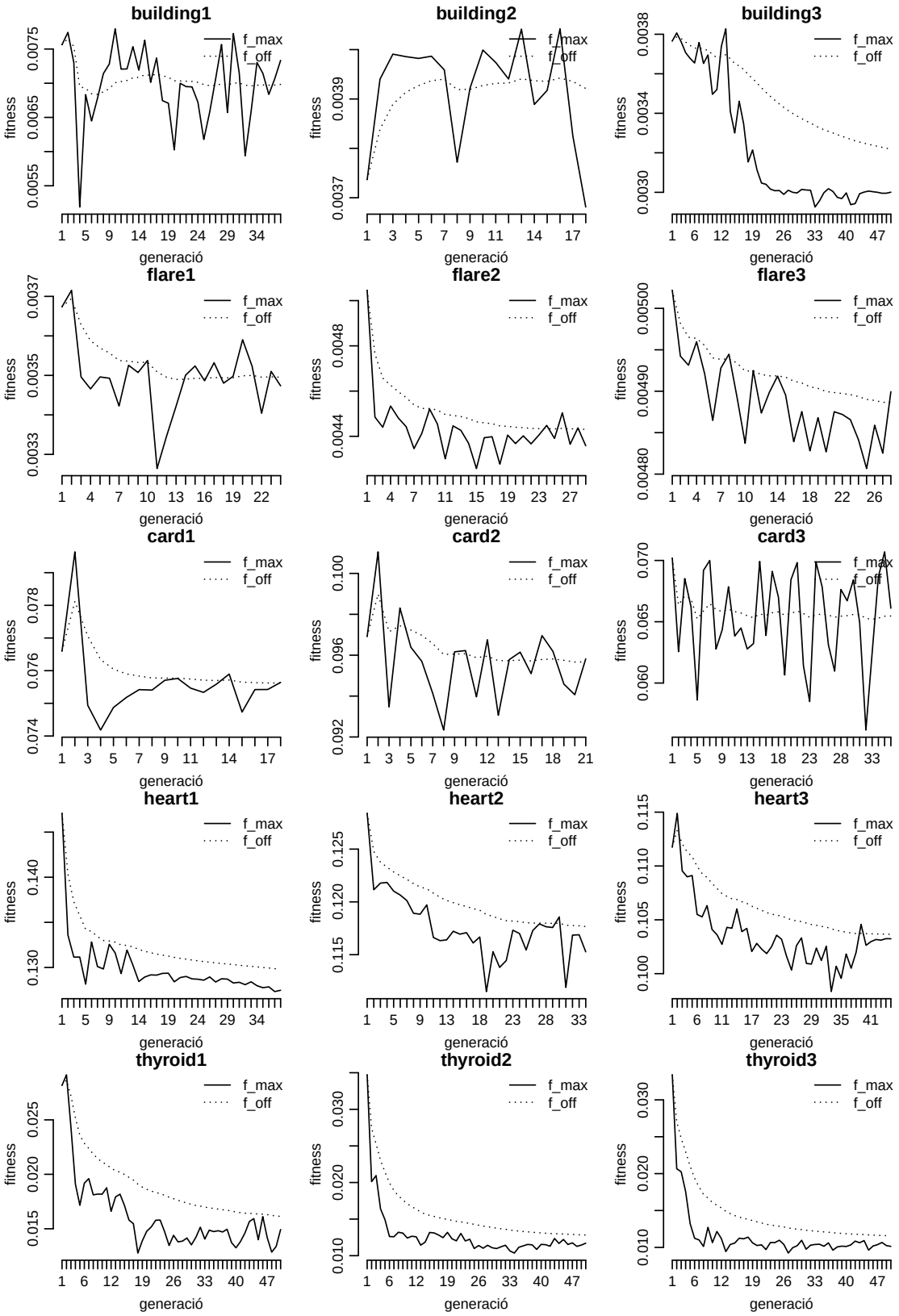

Figura 17: Evolució de l'AG sobre dades de PROBEN1. Algunes evolucions com building1–2 i card2–3 que semblen no ser correctes han resultat en models vàlids. L'AG no ha evolucionat perquè ha trobat un bon organisme des d'el principi.

5.4 Conjunt de dades MNIST

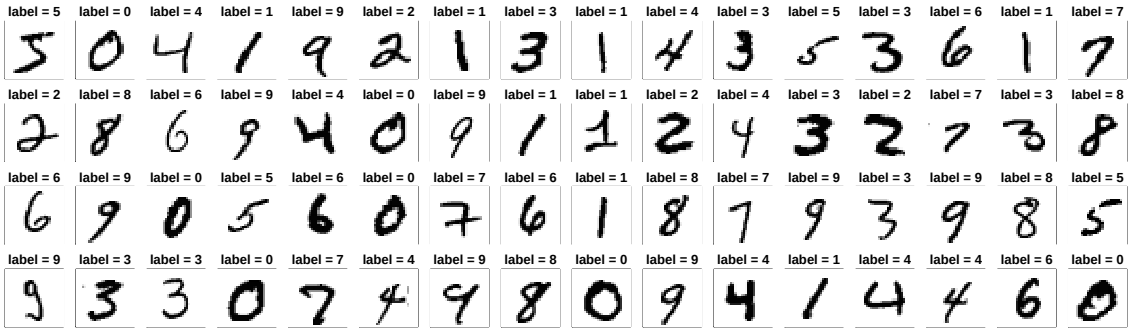

Figura 18: Imatges dels 64 primers dígits del conjunt d'entrenament de MNIST.

En els datasets anteriors s'ha executat l'AG usant tots els exemples del conjunt d'entrenament. S'ha fet així perquè són conjunts relativament petits. Però en el conjunt d'entrenament d'MNIST hi ha 60k exemples, cada un dels quals és una imatge de 28×28 píxels en escala de gris amb un dígit escrit a ma. El problema consisteix en reconèixer el dígit escrit, és per tant una classificació i les variables resposta seran binàries. Per a la XN, el problema és trobar un model per a una funció $\mathbb{R}^{784} \rightarrow \mathbb{R}^{10}$, és a dir 784 variables explicatives i 10 respostes. Amb aquest volum és inviable l'execució de l'AG en un PC amb el temps disponible per a la realització d'aquest treball.

Tal com s'indica[23] a [Col99], quan la funció fitness és molt costosa, com és el cas que ens ocupa, pot iniciar-se l'evolució usant només una mostra representativa dels exemples, i anar incrementant el nombre d'exemples conforme l'evolució avança. Certament, l'AG podria retornar els cromosomes finals, i aquests ser usats com a població inicial en una segona execució, en la que es podrien variar els paràmetres i dades. Encara que aquesta funcionalitat no ha estat implementada en aquesta primera versió, no significa que no sigui vàlid usar un subconjunt prou representatiu del dataset per a l'evolució de l'AG, i posteriorment entrenar una xarxa neural amb la configuració que ha trobat l'AG i el dataset complet. Aquesta ha estat l'aproximació que s'ha seguit en aquesta secció.

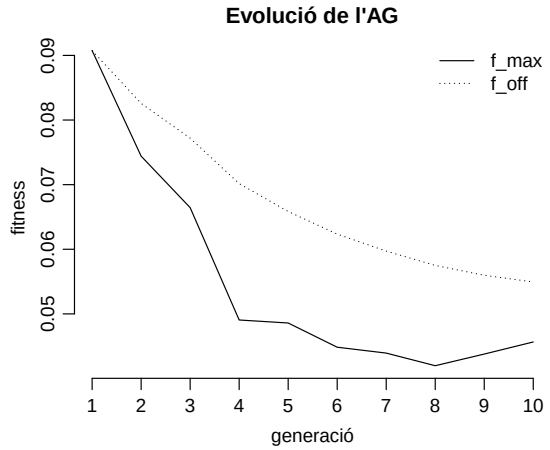

Figura 19: Evolució de l'AG amb una mostra del conjunt d'entrenament MNIST.

S'ha seleccionat una mostra aleatòria de 2000 exemples per entrenament, i 4000 exemples addicionals per validació. Amb totes dues mostres s'ha executat l'AG, limitant el nombre d'epoch a 50, el nombre de generacions a 10, i ampliant el límit de neurones per capa a l'interval [80, 800]. El gràfic que mostra de l'evolució de l'AG és a la figura 19. Les millors configuracions que ha trobat l'AG són:

[23] La referència ho aplica a mínims quadrats, però també serà vàlid per a regressió/classificació amb xarxes neurals.

```
FINAL POPULATION

 fitness      mse      fnp       np      fne ne | ann config
0.043084 0.042339 0.000732  732123 0.000013 13 | tmse=0.011162, lr=0.004395, mo=0.078080, hls=c(288, 596, 328, 407)
0.045574 0.044677 0.000886  886009 0.000011 11 | tmse=0.014870, lr=0.004150, mo=0.077134, hls=c(334, 596, 418, 407)
0.045852 0.044163 0.001677 1677421 0.000011 11 | tmse=0.014900, lr=0.004303, mo=0.257069, hls=c(766, 472, 709, 526)
0.047509 0.046672 0.000825  825495 0.000012 12 | tmse=0.015053, lr=0.004150, mo=0.077134, hls=c(288, 596, 421, 407)
```

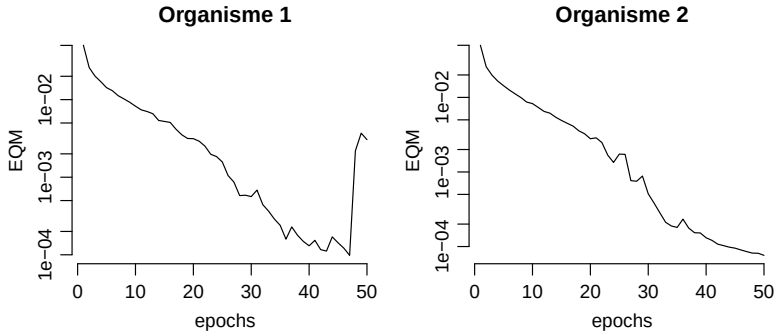

Figura 20: Entrenament de les XN dels dos millors organismes de l'AG. EQM en escala logarísmica.

A continuació s'ha entrenat una xarxa neural per a les dos millors configuracions (la figura 20 mostra el gràfic de cada entrenament), i s'ha avaluat el rendiment de cada una amb el conjunt de test que proporciona MNIST. Les avaluacions es mostren a la taula 17, on es comprova que el segon organisme és el que obté millors resultats.

XN	Precisió %	EQM
org1	97.26	0.144
org2	98.28	0.085

Taula 17: Resultats de l'avaluació dels dos millors organismes per a MNIST.

La configuració escollida és la del segon organisme:

$$\text{Topologia=784-334-596-418-407-10, } \lambda = 0.004150, \mu = 0.077134$$

i amb aquesta es realitza una anàlisi més detallada de la seva precisió. La taula 18 mostra la matriu de confusió del model, amb la classificació que ha realitzat de cada dígit. Aquí es comprova que l'error més freqüent de la XN han estat confondre un 7 per un 2 (9 vegades). La taula 19 llista els valors PPV (Positive Preditive Value) i la sensitivitat de cada dígit, tal com es defineixen a [Bel13]. El PPV d'un dígit Li és la proporció de dígits correctament identificats com Li, entre tots els dígits que el classificador retorna com Li. La sensitivitat del dígit Li és la proporció de dígits correctament identificats, entre tots els dígits que realment són Li.

La precisió global del model classificador és 98.28%, que implica una taxa d'error de classificació del 1.72%, en la línia de molts dels resultats presentats a [Lec99], però lluny encara dels millors. Novament, cal valorar que aquesta configuració de XN s'ha aconseguit de manera automàtica, en poques hores, i amb unes dades (entrenament i validació) que són només el 10% del conjunt d'entrenament MNIST. Amb un conjunt d'entrenament més gran i més temps, és molt probable que l'AG trobi una configuració que permeti millorar aquesta precisió.

Comparatives

Al llibre de Russell i Norvig [Rus10, secció 18.11.1] es proporciona una taula comparativa amb els *millors resultats* obtinguts per diversos algorismes, amb una versió anterior d'aquest mateix problema anomenada NIST. Encara que els resultats no són totalment comparables amb MNIST, si que poden servir per a una comparativa aproximada que es mostra en la taula 20.

	L0	L1	L2	L3	L4	L5	L6	L7	L8	L9
L0	968	0	1	0	0	3	5	1	2	0
L1	0	1128	1	1	0	0	2	1	2	0
L2	3	1	1018	1	1	0	1	3	4	0
L3	0	0	4	991	0	3	0	5	5	2
L4	1	1	0	1	965	0	6	0	0	8
L5	2	0	0	5	1	873	2	0	6	3
L6	3	2	2	0	2	3	943	0	3	0
L7	1	3	9	2	0	0	0	1004	3	6
L8	3	1	3	3	2	2	1	2	955	2
L9	1	2	0	5	7	2	1	5	3	983

Taula 18: Matriu de confusió del model classificador per MNIST. Les etiquetes de fila són els valors reals, les etiquetes de comuna són les prediccions.

	L0	L1	L2	L3	L4	L5	L6	L7	L8	L9
PPV	0.986	0.991	0.981	0.982	0.987	0.985	0.981	0.983	0.972	0.979
Sen	0.988	0.994	0.986	0.981	0.983	0.979	0.984	0.977	0.980	0.974

Taula 19: Valors de PPV i sensitivitat per cada dígit del classificador MNIST.

S'han afegit a la taula els resultats obtinguts amb la XN configurada per l'AG. La memòria requerida pel model es calcula tenint en compte que cada paràmetre es guarda en un float de 4 bytes[24]. S'ha d'advertir que la mesura del temps d'entrenament és orientativa, ja que el rendiment dels equips hardware dels altres algorismes serà probablement diferent.

L'AG ha necessitat 64h = 2.7 dies d'experimentació automàtica per determinar la configuració de la XN. Un cop determinada, la XN aprèn el conjunt d'entrenament de 60k exemples en 3h, i el model obtingut realitza les 10k prediccions del conjunt test en 14s. El model es forma de 886k paràmetres en coma flotant, guardats en un float de 4 bytes (com es fa a la comparativa) resulta en 3.4MiB. En els altres algorismes no s'indica el temps d'experimentació prèvia, però en alguns casos com LeNet (1990s) varen ser necessaris anys d'experimentació i desenvolupament manual [Rus10].

	error %	runtime (ms/digit)	memòria (MiB)	entrenament (dies)	experimentació
XN Boosted LeNet	0.7	50	0.21	30	—
XN LeNet	0.9	30	0.01	14	anys (humà)
SVM	1.1	2000	11	10	no requerida
XN 300 neurones ocultes	1.6	10	0.49	7	—
Classificador 3-NN	2.4	1000	12	0	no requerida
AG+XN	**1.7**	**1.4**	**3.4**	**0.1**	**2.7 dies (auto)**
Humà	0.2–2.5	—	—	—	—

Taula 20: Resultats obtinguts per diversos algorismes en el problema NIST, que és molt similar a MNIST i pot servir com a comparativa aproximada.

A [Lec99] es proporciona un llistat molt més complert de treballs publicats, en els que s'han aplicat diverses tècniques d'aprenentatge automàtic per classificar els dígits MNIST. A la taula 21 es proporciona una llista resumida.

[24] A [Rus10] s'indica que una XN de 123k pesos sinàptics necessita 0.49MiB

classificador	% error
Comitè de 35 XN convolucionals	0.23
XN 7500 neurones en 6 capes (GPU)	0.45
XN convolucional LeNet-5	0.95
SVM kernel polinòmic grau 4 (deskewing)	1.10
SVM kernel gaussià	1.40
XN convolucional LeNet-1	1.70
AG+XN, 1559 neurones en 4 capes	**1.72**
XN 300+100 neurones ocultes (+distortions)	2.50
K-NN, L3	2.83
XN 500+150 neurones ocultes	2.95
40 PCA + quadratic classifier	3.30
XN 300 neurones	4.70
XN 1 capa (classificador lineal)	12.00

Taula 21: Comparativa de la precisió obtinguda en el problema MNIST, per diversos algorismes llistats a [Lec99].

5.5 Automatització amb scripts R

L'obtenció de totes les dades, gràfiques i resultats d'aquesta memòria està automatitzada en diversos scripts R, els quals son la millor documentació possible dels procediments seguits. A més, permeten tornar a obtenir qualsevol resultat amb facilitat. A continuació s'enumeren i es dona una breu descripció dels procediments més destacats.

Nota: s'ha marcat amb ** els scripts que requereixen diverses hores d'execució. No s'indica cap quantitat de temps perquè òbviament depèn de la màquina i de la seva càrrega. Alguns d'aquests scripts han necessitat fins 24h i l'evolució de MNIST necessita 2.7 dies. Les sortides de tots els scripts es lliuren amb el producte del treball en el directori `out`, on R ha afegit al final el temps d'execució en segons.

Scripts d'ajust de l'AG

Per ajustar l'AG es genera una observació per cada configuració de l'AG i cada dataset. La funció `ganntunning_sample` genera una observació per a cada dataset i per una configuració donada. Els paràmetres per defecte d'aquesta funció especifiquen la configuració base, i les variacions sobre aquesta configuració cal especificar-les amb paràmetres en la crida. S'han executat en paral·lel totes les mostres mitjançant un script bash `run_ganntunning.sh` que es lliura amb el producte, i que guarda totes les sortides generades en el directori `out`. No obstant, es poden executar des d'R amb:

```
source("load_workspace.R")
library(genann)

ganntunning_sample()  # configuració base **

# variant offspring **
ganntunning_sample(of=8)
ganntunning_sample(of=32)

# variant elitisme **
ganntunning_sample(el=1)
ganntunning_sample(el=9)

# variant pressió de selecció **
ganntunning_sample(sb=2)
ganntunning_sample(sb=8)
```

```
# variant probabilitat de creuament **
ganntunning_sample(cp=0.90)
ganntunning_sample(cp=0.99)

# variant probabilitat de mutació **
ganntunning_sample(mp=0.01)
ganntunning_sample(mp=0.10)

# variacions addicionals **
ganntunning_sample(of=24,el=2,sb=6,cp=0.99,mp=0.07)
ganntunning_sample(of=24,el=2,sb=8,cp=0.99,mp=0.07)
ganntunning_sample(of=24,el=2,sb=6,cp=0.99,mp=0.05)
ganntunning_sample(of=24,el=2,sb=6,cp=0.99,mp=0.09)
ganntunning_sample(of=24,el=3,sb=6,cp=0.99,mp=0.07)
ganntunning_sample(of=24,el=5,sb=6,cp=0.99,mp=0.07)

source("gann_tunning_report.R")
```

Les observacions generades es guarden en el directori `gann_tunning_data` en format .Rdata, que posteriorment llegeix l'script `gann_tunning_report.R` per generar els gràfics i resultats que es mostren a la secció 4.5.

Scripts per MNIST

Es comença carregant l'espai de treball, que conté la funció `readMNIST` que unifica la lectura de les dades d'entrenament i de test d'MNIST. A continuació, l'script `gann_mnist_evol.R` selecciona dues mostres aleatòries representatives del conjunt d'entrenament MNIST. Una s'usarà per entrenament i l'altre per validació en l'AG. S'executa l'AG amb els dos subconjunts i es guarda el resultat de l'evolució al fitxer `mnist_evol.Rdata`. L'script `gann_mnist_model.R` entrena les xarxes neurals, i guarda els models obtinguts en el fitxer `mnist_models.Rdata`. Finalment `gann_mnist_report.R` llegeix totes les dades anteriors i crea les taules de resultats i gràfiques en el directori `report`. A continuació es deixa indicada la seqüència d'instruccions R:

```
source("load_workspace.R")
source("gann_mnist_evol.R")   # **
source("gann_mnist_model.R")  # **
source("gann_mnist_report.R")
```

Scripts per a la resta de tasques

La resta de tasques s'executen cada una en un únic script:

```
source("ann_synthmath.R")    # proves de XN amb dades sintètiques
source("ann_proben1.R")      # proves de XN amb PROBEN1
source("gann_synthlogic.R")  # proves de l'AG amb funcions lògiques
source("gann_synthmath.R")   # proves de l'AG amb funcions matemàtiques
source("gann_saturated.R")   # proves de l'AG amb xarxes saturades
source("gann_proben1.R")     # proves de l'AG amb dades PROBEN1 **
```

6 Conclusions

Aquest treball és una primera versió del programari i certament admet moltes millores i molta experimentació addicional. Ara bé, el que s'ha fet és suficient per confirmar els resultats que apunten alguns treballs, sobre la viabilitat d'usar algorismes genètics per configurar hiperparàmtres de xarxes neurals, i pot per servir de base a desenvolupaments addicionals. Tenint en compte que des de l'inici aquest treball es va plantejar com una prova de concepte, i en efecte s'ha aconseguit un algorisme genètic que troba configuracions funcionals, sovint bones, per a xarxes neurals, es pot considerar que aquest objectiu s'ha aconseguit.

El desenvolupament podria continuar amb unes proves més exhaustives, provant més conjunts de dades i de procedències més variades, que permetrien ajustar millor les diferents components de l'algorisme genètic, possiblement millorant els seus resultats. Les proves realitzades han estat limitades en gran part pel consum de temps de còmput de l'algorisme genètic. Encara que s'ha disposat d'un servidor remot prou potent a on poder executar diverses proves en paral·lel, moltes d'aquestes execucions han necessitat nits senceres.

A part de la millora de l'algorisme genètic per se, hi ha una sèrie d'ampliacions que poden ser interessants i útils, tant de la xarxa neuronal com de l'algorisme genètic. En el proper apartat s'enumeraran les més significatives.

Possibles extensions i desenvolupament futur

L'objectiu d'automatitzar completament el procés de trobar uns hiperparàmetres òptims per a una XN no s'ha aconseguit completament, ja que l'AG depèn de dos factors que regulen la funció fitness i que varien el resultat de l'evolució. També depèn de que l'usuari indiqui una topologia màxima per a la xarxa neural i possiblement altres paràmetres que configuren l'evolució. Un primer esforç hauria d'anar en la línia d'intentar reduir al màxim la configuració de l'algorisme genètic, doncs no és raonable usar un AG per no haver de configurar una xarxa neural, i acabar havent que configurar l'AG. Encara que la configuració de l'AG és mínima i en molts conjunts de dades no serà necessari tocar-la, la situació ideal seria que aquesta configuració fos inexistent. En aquesta línia, es podrien provar algorismes híbrids que adapten la configuració de l'AG durant l'evolució.

El segon punt a tractar en un desenvolupament futur és intentar millorar la velocitat de l'evolució. Hi ha diversos punts on això es factible, el més destacat dels quals seria detectar a temps una XN que no convergeix. Els organismes que no convergeixen aporten un valor molt limitat a l'evolució, servint únicament de barrera d'un camí que no s'ha de seguir en la optimització. Per contra, són els més costosos ja que esgoten totes les epochs.

Una altre possible optimització que s'ha comentat a la secció 5.4, és l'opció de realitzar evolucions parcials amb l'AG. Caldria que l'AG retornes els cromosomes finals, i que aquests es poguessin tornar a passar a una propera execució de l'AG, en la qual es poden modificar alguns paràmetres o els datasets. Això permetria l'evolució per etapes que es comenta a [Col99], començant per mostres petites del dataset a l'iniciar l'evolució, i anar afegint dades conforme l'evolució avança i cal afinar més la configuració de la XN.

Finalment, no es pot passar per alt la possibilitat de paral·lelitzar tant la xarxa neural com l'algorisme genètic. En la XN es pot escriure una classe addicional derivada de Trainer, que implementi un mini-batch paral·lel. Encara que treballs com [Wil03] indiquen que el benefici d'aquesta aproximació és limitat, ja que part del temps que es guanya per paral·lelisme es perd en un temps d'aprenentatge major degut al mini-batch. L'actual línia d'investigació per paralelitzar xarxes neurals consisteix en aprofitar les GPUs per al càlcul vectorial i matricial [Sei14].

L'AG, per les seves característiques, seria un bon candidat a una reimplementació com a sistema distribuït. Per una banda, l'execució del mètode lifespan de cada organisme és independent i no necessita més informació que el cromosoma de l'organisme. És la operació més costosa i s'ha de repetir per a tots els organismes d'una població. Per això, podrien executar-se els lifespans en nodes diferents d'un sistema distribuït. A més, l'intercanvi d'informació entre nodes seria mínim, únicament caldria enviar els cromosomes als diferents nodes, i recollir els fitness.

Aplicacions al mon real

Les xarxes neurals són molt versàtils i poden aplicar-se a un gran nombre de problemes de regressió i classificació, però presenten l'inconvenient de requerir coneixement sobre el seu funcionament a més de temps per experimentar amb la seva configuració. Sovint no s'usen o es prefereixen altres mètodes d'aprenentatge computacional per aquest motiu. Per això, si existeix una manera automàtica de superar aquestes dificultats, el seu ús podria prioritzar-se.

El programari desenvolupat pot ser d'utilitat en problemes de regressió o classificació on una xarxa neural és adient. Serà especialment útil en problemes d'una mida petita o moderada, per exemple de l'ordre dels problemes de PROBEN1, ja que en aquests casos, l'AG s'executa en uns minuts i dona una configuració per a la xarxa neural quasi immediatament. Problemes més grans requeriran molt més temps d'execució de l'AG i poden no ser viables. Finalment, xarxes neurals molt grans, que ja per si mateixes estiguin en el límit de les màquines disponibles, no podran aprofitar l'AG degut al seu cost d'execució. Les dades MNIST indiquen aproximadament el límit del que pot tractar aquest algorisme genètic en un PC.

Glossari

AG: Algorisme Genètic.

ANN: Artificial Neural Network, veure *XN*.

Aprenentatge inductiu: Procés que construeix un model a partir d'exemples, usant un mètode d'optimització que trobarà el millor model possible en base a una funció d'aptitud. Veure secció 3.1.

Aptitud (funció de): També funció d'*idoneïtat* o d'*adequació*. És una funció que quantifica com d'òptima és una solució. En un problema d'optimització, s'intentarà trobar el mínim o el màxim d'aquesta funció per trobar la millor solució del problema. En anglès *fitness function*.

Cerca estocàstica: Algorisme de cerca en el que s'usen variables aleatòries en la presa de decisions. Els algorismes genètics són un cas particular de cerca estocàstica.

Dataset: Anglicisme que s'usa com a sinònim curt de *conjunt de dades*.

Elitisme: En un AG, l'elitisme consisteix en mantenir en la propera generació, els millors individus de la generació actual.

Època: Una ronda de presentació de tots els exemples a una XN. En anglès *epoch*.

EQM: Error Quadràtic Mig. En anglès *MSE (Mean Squared Error)*.

Factor d'aprenentatge: Quantitat que regula la variació en els pesos sinàptics, en l'algorisme de retropropagació. Es denota per λ en aquest text. En anglès *learning rate*.

Fan-in, fan-out: Son el nombre d'entrades i de sortides d'una neurona, respectivament.

Fitness (function): Anglicisme, veure funció d'*aptitud*.

Grid search: Mètode per de cerca per força bruta, que explora totes les combinacions de diverses posicions fixes en les dimensions d'un espai de cerca multidimensional.

Hiperparàmetres: Valors que configuren la topologia i les característiques d'aprenentatge d'una xarxa neural. En contrapossició als *paràmetres* que són els valors que troba la XN durant el procés d'aprenentatge, per crear un model matemàtic a partir de dades.

Interpolació: És l'obtenció de nous punts (dades), dins del rang d'un conjunt discret de punts coneguts. En contraposició a *extrapolació*, que és la estimació de nous punts fora del rang dels punts originals. Els dos conceptes són similars però la extrapolació està subjecta a major incertesa.

Mètrica: Anglicisme que s'usa sovint com a sinònim de *mesura*.

MLP: Multi Layer Perceptron. Veure *perceptró multicapa*.

Moment o *momentum*: Quantitat que actua creant un efecte d'inèrcia al el procés d'aprenentatge d'una XN, igual que ho fa la massa en els objectes físics. Es denota per μ en aquest treball.

MSE: Mean Squared Error. Veure *EQM*.

Paràmetres: Valors que configuren un model matemàtic. Es distingeixen de les variables, en que un cop fixats, els paràmetres es comporten com a constants en el model.

Perceptró: Neurona artificial. Model matemàtic, inspirat en les neurones biològiques. Es distingeixen diversos tipus de perceptrons segons la seva funció d'activació. El context ha de deixar clar de quin tipus es tracta.

Perceptró multicapa: Tipus de xarxa neural, on les neurones s'organitzen com un graf dirigit acíclic. En anglès *Multi Layer Perceptron (MLP)*.

VA: Variable Aleatòria.

XN: Xarxa Neural. Conjunt de neurones artificials interconnectades. En anglès: *NN (Neural Network)*, també *ANN (Artificial NN)*.

Referències

[Alm08] **Almeida, Leandro M.; Ludermir, Teresa B. (2008):** *Tuning artificial neural networks parameters using an evolutionary algorithm.* IEEE Computer Society, 978-0-7695-3326-1/08.

[Bel13] **Beleites, Claudia; Salzer, Reiner; Sergo, Valter (2013):** *Validation of Soft Classification Models using Partial Class Memberships: An Extended Concept of Sensitivity & Co. applied to the Grading of Astrocytoma Tissues.* Elsevier, Chemometrics and Intelligent Laboratory Systems, 122 (2013), 12–22, arXiv:1301.0264v2 [stat.ML] .

[Ben12] **Bengio, Yoshua (2012):** *Practical Recommendations for Gradient-Based Training of Deep Architectures.* arXiv:1206.5533v2 [cs.LG].

[Bis06] **Bishop, Christopher M. (2006):** *Pattern Recognition and Machine Learning.* Springer, ISBN-13: 978-0387-31073-2.

[Bra08] **Branke, Jürgen; et.al. (2008):** *Multiobjective Optimization, Interactive and Evolutionary Approaches.* Springer-Verlag, ISBN-13: 978-3-540-88907-6.

[Bra94] **Braun, Heinrich; Zagorski, Peter (1994):** *ENZO-M: A Hybrid Approach for Optimizing Neural Networks by Evolution and Learning.* Parallel Problem Solving from Nature, Springer-Verlag, Berlin, pp. 440–451..

[Col99] **Coley, David A. (1999):** *An Introduction to Genetic Algorithms for Scientists and Engineers.* World Scientific Publishing Co. Pte. Ltd. ISBN 98 1-02-3602-6.

[Cur11] **Curteanu, S.; Cartwright, H. (2011):** *Neural networks applied in chemistry. I. Determination of the optimal topology of multilayer perceptron neural networks.* Journal of Chemometrics, vol. 25, num. 1, pages 527–549, DOI 10.1002/cem.1401.

[Cyb89] **Cybenko, George (1989):** *Approximation by Superpositions of a Sigmoidal Function.* Mathematics of Control, Signals, and Systems 2:303-314.

[Deg12] **DeGroot, Morris H.; Schervish, Mark J. (2012):** *Probability and Statistics.* 4a edició, Addison-Wesley, ISBN 13: 978-0-321-50046-5.

[Dej75] **De Jong, Kenneth Alan (1975):** *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* University of Michigan.

[Dej92] **De Jong, Kenneth Alan (1992):** *Genetic Algorithms are NOT Function Optimizers.* Proceedings of the Second Workshop on Foundations of Genetic Algorithms. Vail, Colorado, USA, July 26-29 1992, pages 5–17.

[Fon93] **Fonseca, Carlos M.; Fleming, Peter J. (1993):** *Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization.* Genetic Algorithms: Proceedings of the 5th International Conference, Forrest, S., (Ed.) pages 416–423.

[Goo16] **Goodfellow et. al. (2016):** *Deep Learning.* Book in preparation for MIT Press, `http://www.deeplearningbook.org`.

[Gre00] **Greenhalgh, David; Marshall, Stephen (2000):** *Convergence Criteria for Genetic Algorithms.* SIAM J. COMPUT. Vol. 30, No. 1, pp. 269–282, Society for Industrial and Applied Mathematics.

[Hol92] **Holland, John H. (1992):** *Adaptation in natural and artificial systems.* MIT Press Cambridge, MA, USA, ISBN:0-262-58111-6.

[Hun11] **Hundt, Robert (2011):** *Loop Recognition in C++/Java/Go/Scala.* Google Research, `http://research.google.com/pubs/archive/37122.pdf`.

[Koe94] **Koehn, Philipp (1994):** *Combining Genetic Algorithms and Neural Networks: The Encoding Problem.* University of Tennessee, Knoxville.

[Lec98] **LeCun, Yann et. al. (1998):** *Efficient BackProp.* Neural Networks: Tricks of the trade, Orr, G. and Muller K., Springer.

[Lec99] **LeCun, Yann et. al. (1999):** *The MNIST database of handwritten digits.* `http://yann.lecun.com/exdb/mnist/`.

[Lic13] **Lichman, M. (2013):** *UCI Machine Learning Repository.* University of California, Irvine, School of Information and Computer Sciences, `http://archive.ics.uci.edu/ml`.

[Mol10] **Molina, Luis Carlos; Sangüesa, Ramon (2010):** *Avaluació de models (Mineria de dades).* UOC, PID00165725.

[Pre94] **Prechelt, Lutz (1994):** *PROBEN1 – A Set of Benchmarks and Benchmarking Rules for Neural Network Training Algorithms.* Fakultät für Informatik, Universität Karlsruhe, `ftp://ftp.ira.uka.de/pub/neuron/proben1.tar.gz`.

[Qia98] **Qian, Ning (1998):** *On the momentum term in gradient descent learning algorithms.* Elsevier, Neural Networks 12 (1999) 145–151.

[Rct16] **R Core Team (2016):** *Writing R Extensions.* Documentació en línia, `http://cran.r-project.org/doc/manuals/r-release/R-exts.html`.

[Roj96] **Rojas, Raúl (1996):** *Neural Networks, A Systematic Introduction.* Springer-Verlag, ISBN-13: 978-3540605058.

[Rus10] **Russell, Stuart J.; Norvig, Peter (2010):** *Artificial Intelligence - A Modern Approach.* 3a edició, Pearson, ISBN-13: 978-0-13-604259-4.

[Saf04] **Safe, Martín et.al. (2004):** *On Stopping Criteria for Genetic Algorithms.* Advances in Artificial Intelligence - SBIA, Springer-Verlag, ISBN: 3-540-23237-0, 405-413.

[Sar02] **Sarle, Warren S. et. al. (2002):** *Neural Network FAQ.* `ftp://ftp.sas.com/pub/neural/FAQ.html`.

[Sei14] **Seide, Frank; et.al. (2014):** *On Parallelizability of Stochastic Gradient Descent for Speech DNNS.* IEEE International Conference on Acoustic, Speech and Signal Processing (ICASSP), 235-239.

[Sri94] **Srinivas, N.; Deb, K. (1994):** *Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms.* Evolutionary Computation, Vol. 2.

[Tor10] **Torra, Vicenç (2010):** *Resolució de problemes i cerca (Intel·ligència artificial).* UOC, PID00163090.

[Tor13] **Torra, Vicenç; Masip, David (2013):** *Aprenentatge (Aprenentatge computacional).* UOC, PID00200707.

[Tur50] **Turing, Alan Mathison (1950):** *Computing Machinery and Intelligence.* Mind, vol. LIX. No. 236, pages 433-460.

[Wil03] **Wilson, D. Randall; Martinez, Tony R. (2003):** *The general inefficiency of batch training for gradient descent learning.* Elsevier, Neural Networks, 16 (2003) 1429-1451.