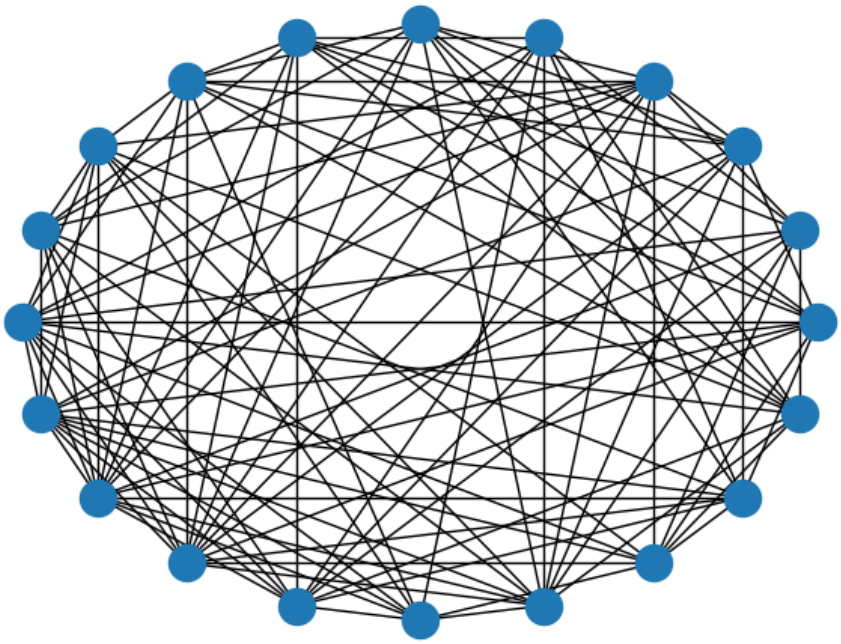
UAB
Universitat Autònoma
de Barcelona

Treball Final de Grau
Grau en Matemàtiques

Teoria de grafs a través de l'aprenentatge per reforç



Anna Aumatell Valls

Tutor: Roberto Rubio Núñez

Febrer 2024

Abstract

In this work we will explore reinforcement learning and show its applicability to find explicit and unique counterexamples to conjectures in graph theory. We will start by establishing the mathematical foundations of reinforcement learning. Then, we will explain neural networks from a mathematical viewpoint, as well as their role in reinforcement learning as means to approximate functions. Lastly, we will present the edge-Szeged index of a graph and we will conduct a practical case study that will consist on applying reinforcement learning to the conjecture that the complete graph attains the maximum edge-Szeged index. More concretely, we improve the known counterexamples from graphs of order more than 100 to order 20.

Resum

En aquest treball estudiarem l'aprenentatge per reforç i mostrarem la seva aplicabilitat per trobar contraexemples inèdits i explícits en teoria de grafs. Començant per definir els fonaments matemàtics de l'aprenentatge per reforç. Seguint per explicar les xarxes neuronals des d'un punt de vista matemàtic, juntament amb el seu paper en l'aprenentatge per reforç com a eina per aproximar funcions. Finalment, introduirem l'edge-Szeged índex d'un graf i durem a terme un cas pràctic que consistirà en l'aplicació de l'aprenentatge per reforç en la conjectura de què el graf complet assoleix el màxim edge-Szeged índex. Més específicament, els contraexemples ja coneguts fins a la data consistien en grafs d'ordre superior a 100, els millorarem trobant grafs que refuten la conjectura a partir d'ordre 20.

Índex

Capítol 1

Introducció

Tant nens petits, com gossos o altres éssers vius capaços d'aprendre, en el seu dia a dia segueixen un procés d'aprenentatge que implica l'ús de l'experiència i la interacció amb l'entorn per adquirir coneixements i comportaments. Quan un nen no endreça les seves joguines rep un càstig o quan un gos aprèn un truc nou rep un premi, aquests són alguns exemples d'aquesta retroalimentació positiva o negativa que fa que aquests individus aprenguin de manera instintiva a distingir entre les accions bones i les accions dolentes. L'aprenentatge per reforç tracta de replicar aquests processos d'aprenentatge que passen pels nostres cervells sense que nosaltres en siguem gairebé conscients, de manera que una màquina guiant-se per l'entorn i l'experiència sigui capaç de triar les accions que donin més premi o recompensa, en cada situació.

L'aprenentatge per reforç, o reinforcement learning en anglès, està basat en diferents principis que fa molts anys que són estudiats, com l'aprenentatge per prova i error o la programació dinàmica. Tot i així, el primer cop que s'utilitza el terme 'aprenentatge per reforç' en l'àmbit de la computació, és l'any 1961 per Marvin Minsky en [Min61]. Però no seria fins als anys 1980 que R.Sutton i A.Barto definirien els fonaments actuals d'aquesta àrea en [RA98]. Aquest va ser el punt de partida d'un sector que segueix en expansió i que compta cada cop amb més aplicacions, sent algunes d'elles la robòtica, els videojocs o la biologia. En aquest treball, però, explorarem una altra aplicació de l'aprenentatge per reforç, en aquest cas serà en la teoria de grafs.

Aquest treball té com a objectiu definir els fonaments matemàtics de l'aprenentatge per reforç i de les xarxes neuronals, explicant com la combinació d'aquestes dues eines pot permetre que un ordinador no només assimili informació de manera similar als humans, sinó que en alguns casos vagi més enllà, arribant a resultats que superin les limitacions dels mètodes computacionals tradicionals.

En el primer capítol, buscarem aprofundir en els fonaments matemàtics de l'aprenentatge per reforç, utilitzant el marc conceptual dels processos de decisió de Màrkov. Més endavant, introduirem les xarxes neuronals com a eina per aproximar funcions, juntament amb una descripció del paper fonamental que juguen en l'aprenentatge per reforç. Per il·lustrar-ho presentarem el mètode de l'entropia creuada, que es basa en l'ús de xarxes neuronals per resoldre problemes d'aprenentatge per reforç. Finalment, en l'últim capítol introduirem els conceptes de teoria de grafs necessaris, per poder plantejar una situació concreta en què utilitzarem el mètode de l'entropia creuada per trobar contraexemples explícits a una conjectura sobre l'edge-Szeged índex d'un graf. Concloem el treball millorant significativament els contraexemples que prèviament refutaven la conjectura, reduint en 80 vèrtexs l'ordre dels anteriors passant d'ordre 100 a 20.

Capítol 2

Aprenentatge per reforç

L'aprenentatge automàtic (machine learning, en anglès) és una branca de la intel·ligència artificial centrada en el desenvolupament de models i algorismes que permeten als ordinadors aprendre patrons i dur a terme tasques de manera autònoma. L'objectiu és que els programes millorin amb l'experiència, a base de repeticions, i puguin fer prediccions o prendre decisions per ells mateixos.

Hi ha tres tipus principals d'aprenentatge automàtic:

- En l'**aprenentatge supervisat** l'algoritme s'entrena utilitzant un conjunt de dades que inclouen exemples d'entrada i les seves sortides corresponents, ja etiquetades amb la classe a la qual pertanyen. L'objectiu és aprendre una relació entre les entrades i les sortides per fer prediccions o classificacions en noves dades.
- Contràriament al cas anterior, en l'**aprenentatge no supervisat** l'algoritme s'entrena amb dades que no estan etiquetades i ha de descobrir patrons i relacions per ell mateix. El model intenta descobrir estructures o agrupacions de manera autònoma, és a dir, sense informació prèvia sobre les sortides.
- En l'**aprenentatge per reforç** hi ha un agent que pren decisions utilitzant informació sobre l'entorn, amb l'objectiu de maximitzar una recompensa acumulativa. Aquest tipus d'aprenentatge consisteix a fer que l'agent aprengui mitjançant prova i error, ajustant les seves decisions de manera que triï aquelles que proporcionin més recompenses.

Cadascun dels tres tipus d'aprenentatge automàtic presentats té les seves fortaleses i debilitats, i és convenient per a diferents problemes i situacions. Aquesta tesi se centrarà en l'aprenentatge per reforç, que és el més adequat per a programes capaços de prendre decisions seqüencials, com per exemple, en jocs d'estratègia en què s'hagin de triar una sèrie d'accions amb l'objectiu de maximitzar una recompensa o puntuació del joc.

En aquest capítol les referències utilitzades han estat [RA18], [Han18] i els capítols 1, 2 i 3 de [Zha24].

2.1 Conceptes bàsics

L'aprenentatge per reforç utilitza l'estructura conceptual dels processos de decisió de Màrkov, que van ser definits a la dècada del 1950 per Bellman [Bel57]. Més endavant, en l'any 1998 Sutton i Barto [RA98] van exposar com els elements i les propietats dels processos de decisió de Màrkov podien ser aplicats per estudiar l'aprenentatge per reforç.

A continuació, introduirem aquests conceptes com a marc conceptual de l'aprenentatge per reforç, que ens permetran entendre quins són els seus elements principals i el seu funcionament.

Definició 1. Una **cadena de Màrkov** és un tipus de procés estocàstic en què un sistema passa d'un estat a un altre amb probabilitats específiques.

La clau d'una cadena de Màrkov és que la probabilitat de transició a un estat futur només depèn de l'estat actual i no dels estats anteriors. A aquesta propietat es coneix com a **propietat de Màrkov** o manca de memòria. Matemàticament, es pot representar com:

$$P(X_{n+1} = x | X_n = x_n, X_{n-1} = x_{n-1}, \ldots, X_0 = x_0) = P(X_{n+1} = x | X_n = x_n),$$

on X_n és l'estat del sistema en el temps n.

Podem ampliar el concepte d'una cadena de Màrkov a una situació en què es pretengui modelar la presa de decisions, aleshores s'obtenen els processos de decisió de Màrkov, que defineixen la interacció entre un agent i el seu entorn en termes d'estats, accions i recompenses.

Definició 2. Un **procés de decisió de Màrkov** és una 4-tupla $(\mathcal{S}, \mathcal{A}, P, \mathcal{R})$ on:

- $\mathcal{S}$ és un conjunt d'**estats**.
- $\mathcal{A}$ és un conjunt d'**accions**, amb una partició en subconjunts $\mathcal{A}(s)$ on $s \in \mathcal{S}$. Cada $\mathcal{A}(s)$ denota les accions possibles en l'estat $s \in \mathcal{S}$.
- La funció de probabilitat de **transició d'estats** $P(s'|s, a)$ es defineix per cada parell (s, a) on $s \in \mathcal{S}$ i $a \in \mathcal{A}(s)$, i ens dona la probabilitat que l'acció a en l'estat s porti a l'estat s'.
- La **funció de recompenses** $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, ens dona la recompensa immediata $\mathcal{R}(s, a)$ que s'obté en prendre l'acció a estant en l'estat s.

La funció de recompenses $\mathcal{R}$ ens permet obtenir una funció de probabilitat de **transició de recompenses**. Que es denota per una distribució de probabilitat $p(r|s, a)$ que representa la probabilitat de rebre una recompensa $r \in \mathcal{R}(s, a)$ estant a l'estat $s \in \mathcal{S}$ i prenent l'acció $a \in \mathcal{A}(s)$.

Observació 1. *Anomenarem **agent** a l'individu o objecte que ha de realitzar l'acció en els nostres problemes. No és un concepte matemàtic, però és important, ja que serà normalment el centre dels problemes que tractarem.*

Un cop definits els elements principals, el fet de triar una acció per cada estat requereix un mecanisme, que anomenarem política.

Definició 3. Una **política** es defineix com una funció que assigna una probabilitat a cadascuna de les possibles accions en un estat. Denotarem una política com $\pi(a|s)$.

A més a més, en els processos de decisió de Màrkov les dinàmiques i les interaccions estan definides seguint les notacions següents:

Notació 1. *Considerarem que l'agent i l'entorn interaccionen entre ells en una seqüència discreta de temps $t = 0, 1, 2, \ldots$, tal que en cada temps t l'agent es troba a l'estat S_t i l'acció que prendrà seguint una política π és A_t. Aleshores en el següent temps $t + 1$, com a conseqüència de l'acció, l'agent rebrà la recompensa R_{t+1} i es trobarà en l'estat S_{t+1}. Això es pot resumir com*

$$S_t \xrightarrow{A_t} S_{t+1}, R_{t+1}.$$

Aleshores, per definició tenim que S_t, A_t i R_{t+1} són tot variables aleatòries. Són considerades variables aleatòries, ja que totes tres estan definides com a funcions en què a un succés concret (la situació de l'agent en el temps t) se li assigna un valor (que pot ser un estat, una acció o una recompensa en cada cas). Així, en tractar-se de variables aleatòries, cadascuna tindrà associada una distribució de probabilitat que ens permetrà saber en el temps t les probabilitats respectives de trobar-se en un estat concret, de prendre cadascuna de les accions i de rebre una certa recompensa.

Aquestes variables ens permeten definir els següents conceptes:

Definició 4. Una **trajectòria** és una cadena estat-acció-recompensa, passant d'un estat a un altre prenent l'acció i obtenint la recompensa indicades.

$$S_0 \xrightarrow[R_1]{A_0} S_1 \xrightarrow[R_2]{A_1} S_2 \xrightarrow[R_3]{A_2} S_3 \dots$$

Definició 5. S'anomena **episodi** quan aquesta trajectòria acaba en un estat terminal, és a dir, si es tracta d'una trajectòria finita.

Definició 6. Es defineix el **retorn** d'una trajectòria com la suma de les recompenses de tots els passos de la cadena. Es pot anomenar també recompensa total o recompensa acumulada.

$$\text{retorn} = R_1 + R_2 + R_3 + R_4 + \dots \quad \text{on } R_i \text{ és la recompensa en el temps } i.$$

Definició 7. El **retorn amb descompte** és la suma de totes les recompenses obtingudes per l'agent que s'utilitza quan tenim una trajectòria no finita, per evitar divergència. Es calcula utilitzant un factor de descompte $\gamma \in (0, 1)$.

$$G = R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \dots \quad \text{on } R_i \text{ és la recompensa en el temps } i.$$

Observació 2. *De la definició anterior, podem considerar també el retorn amb descompte al llarg de la trajectòria, com:*

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \quad \textit{on } R_i \textit{ és la recompensa en el temps } i,$$

amb un factor de descompte $\gamma \in (0, 1)$.

Exemple 1. *Per visualitzar aquests conceptes podem utilitzar un exemple bàsic, en el qual suposem que hi ha un jugador (**agent**) que mou una peça en un tauler quadriculat format per 4 caselles (**estats**). El tauler té la forma i les accions descrites en les figures 1 i 2.*

Veiem que es defineix el joc de manera que hi ha 4 possibles estats, s_i *per* $i = 1, 2, 3, 4$, *i en cadascun tenim 4 possibles **accions*** a_i *per* $i = 1, 2, 3, 4$, *que són moure la peça cap a qualsevol dels 4 costats del quadrat.*

Figura 1: Tauler de l'Exemple 1.

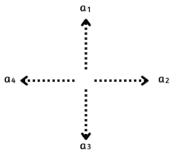


Figura 2: Possibles accions en cada estat.

*A més a més, s'especifiquen les **recompenses** següents:*

- *Si es mou la peça a una casella blava (objectiu) el retorn és de +1.*
- *Si es mou la peça a una casella blanca el retorn és de 0.*
- *Si es mou la peça a una casella vermella (prohibides) aleshores el retorn és de -1.*
- *Si es tracta de moure la peça fora del tauler el retorn és de -1, i la peça es queda en la casella on estava.*

Per tant, l'objectiu és trobar una ***política*** *que maximitzi el* ***retorn****, que consistirà en tractar d'arribar a la casella blava, evitant la casella vermella i evitant sortir del tauler.*

Donem dues possibles polítiques deterministes (en cada estat es pren una acció amb probabilitat 1), en les que les fletxes indiquen l'acció a prendre en cada estat:

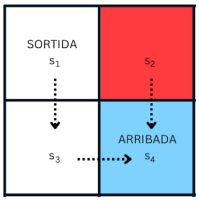


Figura 3: Política π_1.

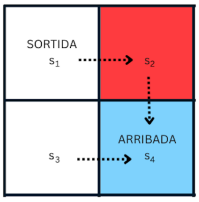


Figura 4: Política π_2.

El retorn per cadascuna d'aquestes polítiques seria:

Donada la política π_1 començant en l'estat s_1, l'agent seguirà la trajectòria

$$s_1 \xrightarrow[r=0]{a_3} s_3 \xrightarrow[r=1]{a_2} s_4,$$

aleshores el retorn i retorn amb descompte seran:

$$retorn = 0 + 1 = 1$$

$$G = 0 + \gamma \cdot 1 = \gamma \in (0, 1)$$

En canvi, donada la política π_2 començant en l'estat s_1, l'agent seguirà la trajectòria

$$s_1 \xrightarrow[r=-1]{a_2} s_2 \xrightarrow[r=1]{a_3} s_4,$$

aleshores el retorn i retorn amb descompte seran:

$$retorn = -1 + 1 = 0$$

$$G = 0 + \gamma \cdot 1 = -1 + \gamma \in (-1, 0).$$

Veiem que hem formalitzat mitjançant els conceptes de política, trajectòria i retorn la intuïció de què utilitzant π_1 obtindríem més recompensa acumulada que utilitzant π_2.

Observació 3. *Un dels reptes que apareix en l'aprenentatge per reforç, és el* ***dilema entre exploració i explotació****. Aquest dilema és una peça clau en la resolució de problemes mitjançant aprenentatge per reforç, ja que implica intentar buscar l'equilibri entre l'explotació, que consisteix en aprofitar aquelles accions que l'agent ja sap que proporcionen una bona recompensa, amb l'exploració que es basa en buscar noves accions per obtindre millors recompenses a llarg termini. El dilema radica en el fet que cap de les dues opcions pot ser utilitzada sense l'altra, ja que això implicaria limitar la capacitat de l'agent d'obtenir la màxima recompensa acumulada possible.*

2.2 Funcions de valor d'estat

Intuïtivament, podem utilitzar la idea que una política serà millor que una altra si el retorn que s'obté començant en el mateix estat, i seguint una política en lloc de l'altra, és superior. Aquesta idea no és errònia, però per formalitzar-la ens cal introduir el concepte de valor d'estat.

Definició 8. Definim el **valor d'estat**, per una política fixada, com l'esperança de tots els possibles retorns que s'obtenen començant en aquest estat. És a dir,

$$v_\pi(s) \doteq \mathbb{E}\left[G_t | S_t = s\right]. \tag{2.1}$$

La funció de valor d'estat està ben definida, ja que podem veure que el retorn amb descompte G_t és una variable aleatòria i, per tant, podem calcular la seva esperança. Podem afirmar que G_t és una variable aleatòria perquè està definida com,

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

i anteriorment hem vist que $R_{t+1}, R_{t+2}, \dots$ eren tot variables aleatòries.

El concepte de valor d'estat és el que generalment utilitzarem per avaluar polítiques: aquelles polítiques que generin valors d'estat majors seran millors.

2.2.1 Equació de Bellman

Una de les eines essencials per al càlcul dels valors d'estat és l'Equació de Bellman. Es tracta d'un conjunt d'equacions lineals que permeten tractar un problema d'optimització com un conjunt de subproblemes més simples, ja que cada equació defineix una relació entre els valors de tots els estats.

Per obtenir l'Equació de Bellman, primerament, veiem que podem escriure:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = R_{t+1} + \gamma G_{t+1}.$$

Substituint aquesta igualtat en la definició de valor d'estat (2.1), obtenim:

$$v_\pi(s) \;=\; \mathbb{E}\left[G_t | S_t = s\right] = \mathbb{E}\left[R_{t+1} | S_t = s\right] + \gamma \mathbb{E}\left[G_{t+1} | S_t = s\right]. \tag{2.2}$$

Utilitzant conceptes de teoria de la probabilitat, com la llei de l'esperança iterada o la independència condicional, podem calcular:

$$\mathbb{E}\left[R_{t+1} | S_t = s\right] = \sum_a \pi(a|s) \sum_r p(r|s,a) r. \tag{2.3}$$

$$\mathbb{E}\left[G_{t+1} | S_t = s\right] = \sum_{s'} v_\pi(s') \sum_a p(s'|s,a)\pi(a|s). \tag{2.4}$$

Aleshores, substituint les igualtats trobades (2.3) i (2.4) en (2.2), obtenim que

$$\begin{aligned} v_\pi(s) &= \sum_a \pi(a|s) \sum_r p(r|s,a) r + \gamma \Big(\sum_{s'} v_\pi(s') \sum_a p(s'|s,a)\pi(a|s)\Big) \\ &= \sum_a \pi(a|s) \left[\sum_r p(r|s,a) r + \gamma \sum_{s'} v_\pi(s') p(s'|s,a)\right], \quad \text{per tot } s \in \mathcal{S}. \end{aligned} \tag{2.5}$$

Aquesta última equació és l'anomenada **Equació de Bellman**.

Podem veure que per calcular el valor d'estat $v_\pi(s)$ per un $s \in \mathcal{S}$ concret, ens cal utilitzar $v_\pi(s')$ de tota la resta d'estats $s' \in \mathcal{S}$ tal que $s' \neq s$. Així, aquest sistema d'equacions ens permet calcular els valors d'estat, i a més a més, caracteritzar les relacions entre els valors dels

diferents estats. En la pràctica podem resoldre l'Equació de Bellman iterativament per obtenir els valors d'estat.

2.2.2 Funcions dels valors d'acció

Utilitzant la mateixa idea del concepte de valor d'estat, podem definir els valors d'acció:

Definició 9. La funció dels **valors d'acció** serà una funció que per cada estat i acció possible en l'estat, ens donarà el l'esperança de tots els possibles retorns que s'obtenen prenent aquella acció. Aleshores, matemàticament la definició del valor d'acció és

$$q_\pi(s,a) \doteq \mathbb{E}\left[G_t | S_t = s, A_t = a\right],$$

per tot estat $s \in \mathcal{S}$ i acció $a \in \mathcal{A}(s)$.

Utilitzant les propietats d'esperança condicional de teoria de probabilitat, deduïm les igualtats següents:

$$v_\pi(s) = \mathbb{E}\left[G_t | S_t = s\right] = \sum_a \mathbb{E}\left[G_t | S_t = s, A_t = a\right] \pi(a|s) = \sum_a q_\pi(s,a)\pi(a|s),$$

així hem trobat la relació entre els valors d'estat i els valors d'acció.

Els valors d'acció són una eina molt important en l'aprenentatge per reforç, ja que sent combinats amb els valors d'estat serveixen com a mètode per trobar polítiques òptimes amb més facilitat.

2.3 Mètodes basats en valors i mètodes basats en polítiques

Sabem que el principal objectiu dels mètodes d'aprenentatge per reforç consisteix en trobar la política òptima π^* que maximitzi el retorn acumulat. Primer, ens cal definir el concepte de política òptima:

Definició 10. Una **política π^* és òptima** si $v_{\pi^*}(s) \geq v_\pi(s)$ per tot $s \in \mathcal{S}$ i per qualsevol altra política π. Els valors d'estat de π^* son els **valors d'estat òptims**.

Existeixen dues maneres diferents d'assolir aquest objectiu: els mètodes basats en valors, que són aquells que busquen trobar la funció dels valors d'estat òptima, i els mètodes basats en polítiques que són els que directament tracten d'aproximar la funció que ens doni la política òptima. Els **mètodes basats en valors** se centren en utilitzar les funcions de valor d'estat i d'acció per avaluar l'atractiu de les diferents accions o estats en un entorn. Així doncs, no t'indiquen quina acció has de prendre, sinó que et permeten saber quina recompensa s'obtindrà en cada cas, i així faciliten la tria de les accions a realitzar.

En els mètodes basats en valors s'utilitza la idea que una funció de valor òptima ens proporciona una política òptima. Això és degut al fet que una política serà òptima si els seus corresponents valors d'estat són majors o iguals que els de la resta de polítiques.
El procés més bàsic per anar actualitzant una política fins a arribar a aquella que sigui òptima consisteix en:

1. Primerament, calculem els valors d'estat de la política inicial donada π_0, utilitzant l'Equació de Bellman: $v_{\pi_0}(s)$.

2. Tot seguit, fixem un estat s_i i calculem els valors d'acció per aquell estat: $q_{\pi_0}(s_i, a)$, per tot $a \in \mathcal{A}(s_i)$.

3. Triem aquella acció $a_j \in \mathcal{A}(s_i)$ que tingui el major valor d'acció,

$$q_{\pi_0}(s_i, a_j) > q_{\pi_0}(s_i, a_k),$$

per tot $a_k \in \mathcal{A}(s_i)$ tal que $a_j \neq a_k$, Aleshores, actualitzem la política de manera que en l'estat s_i triï l'acció a_j. Així doncs, hem actualitzat la política perquè maximitzi el retorn en l'estat s_i, és a dir, hem trobat el valor d'estat òptim per a s_i.

4. Repetim el procés per la resta d'estats.

Aquest esquema presenta de manera general la idea del procés que es segueix en els algoritmes d'aprenentatge per reforç basats en valors. Després, cada mètode té els seus propis detalls específics en el procés, per exemple, pot haver-hi variacions en el criteri de tria de les accions en cada estat.

Per una altra banda, la idea dels **mètodes basats en polítiques** consisteix en parametritzar la política i tractar de maximitzar el seu rendiment per així trobar aquella que sigui òptima. Aquests mètodes només estan interessats en conèixer aquesta funció que ens dirà quina acció prendre en cada estat. Per tant, no sabrem quanta recompensa rebrà l'agent en cada estat, ni en la tria de cada acció, ja que no s'aprèn ni una funció de valor d'estat $v_\pi(s)$ ni una funció de valor d'acció $q_\pi(s, a)$.

Els mètodes basats en polítiques utilitzen diferents maneres d'aproximar la política òptima, una d'elles és la d'utilitzar xarxes neuronals per obtenir en cada estat una distribució de probabilitat de les accions a realitzar que ens aportin el màxim retorn acumulat, més endavant veurem aquest mètode en detall.

Capítol 3

Xarxes neuronals a l'aprenentatge per reforç

Les xarxes neuronals són introduïdes en l'aprenentatge per reforç com a eina per aproximar funcions no lineals, com poden ser les funcions de valor o les polítiques. En aquest capítol, primerament explicarem els conceptes teòrics de les xarxes neuronals, i més endavant, estudiarem les seves aplicacions com a instrument per aproximar polítiques en el context de l'aprenentatge per reforç.

En aquest capítol han sigut utilitzades la referència [GBC16] i els seminaris 1 i 2 en [SMR22].

3.1 Xarxes neuronals

Primerament, introduirem els diferents elements que formen una xarxa neuronal, i després explicarem breument el seu funcionament.

Suposem que tenim una funció $f : \mathbb{R}^n \to \mathbb{R}^m$ en un conjunt de dades $\{(x^{(j)}, y^{(j)})\}$ on j denota l'índex de la mostra, $x^{(j)} \in \mathbb{R}^n$ i $f(x^{(j)}) = y^{(j)} \in \mathbb{R}^m$, i a més a més suposem que volem aproximar aquesta funció f mitjançant una xarxa neuronal. Aleshores, una xarxa neuronal és una funció $\tilde{f} : \mathbb{R}^n \to \mathbb{R}^m$ tal que $\tilde{f}(x^{(j)}) = \tilde{y}^{(j)}$, i l'objectiu principal serà que $\tilde{f}(x^{(j)}) = \tilde{y}^{(j)}$ aproximi el millor possible $f(x^{(j)}) = y^{(j)}$.

En les xarxes neuronals hi ha dos components principals, el primer són els nodes (o neurones) i el segon són les funcions d'activació.

Els **nodes** o neurones simulen la funció de les neurones en el cervell humà. Les xarxes neuronals estan compostes de nodes interconnectats entre ells que es passen informació. Aquests nodes estan organitzats en diferents capes: la capa d'entrada on hi haurà les entrades $x^{(j)}$, les capes ocultes i la capa de sortida on es retornaran les sortides $\tilde{y}^{(j)}$. A més a més, cada connexió entre nodes porta associat un pes, que determina la força de la informació que es passa entre aquells nodes.

Les **funcions d'activació** són funcions no-lineals que s'utilitzen per decidir si un node ha de ser activat o no, basant-se en la suma ponderada dels nodes de la capa anterior. Les funcions d'activació seran denotades φ.

Aleshores, una xarxa neuronal segueix l'esquema següent:

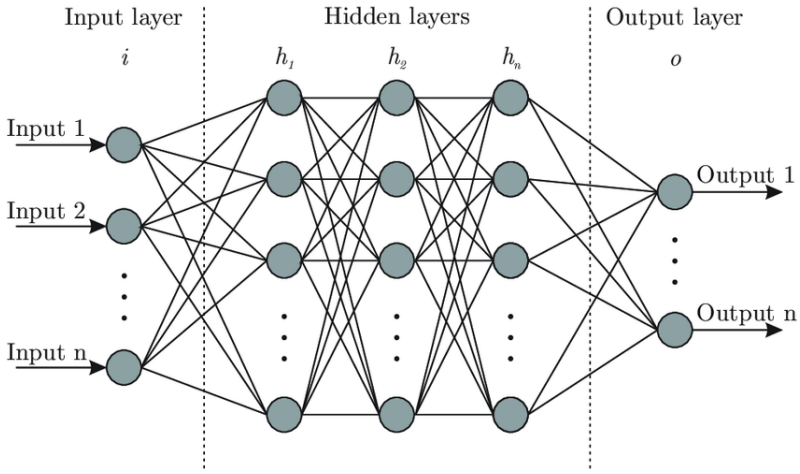


Figura 5: Esquema d'una xarxa neuronal [Noo17].

On en cadascuna de les connexions entre dos nodes s'aplica la funció d'activació de la manera següent:

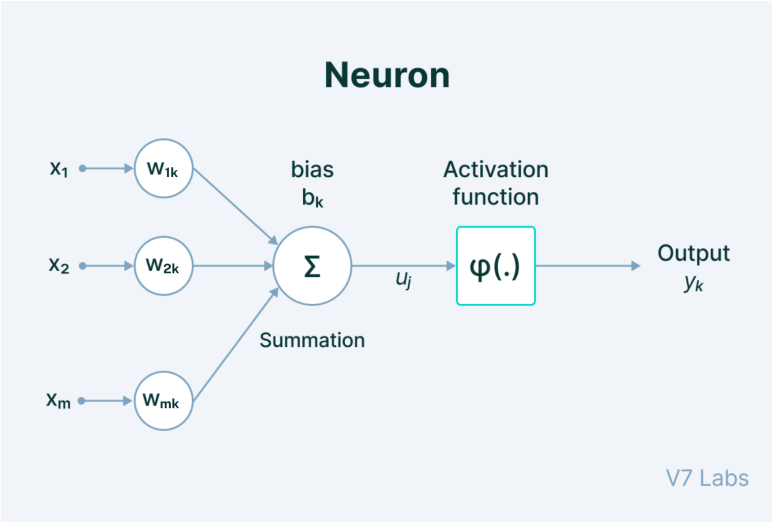


Figura 6: Interior d'una neurona en una xarxa neuronal [Bah21].

Així, utilitzarem la notació següent per als conceptes definits:

Notació 2. *Identificarem els nodes d'entrada per* $x_i^{(j)}$ *on* i *és el nombre del node en la capa d'entrada i* j *és l'índex que identifica el vector dins del conjunt de dades. Els nodes en la capa de sortida es denotaran* $y_i^{(j)}$*. Els nodes de les capes ocultes es denotaran* $h_i^{(l)}$*, que indica que és la* i*-ena neurona en la capa* l*.*

Notació 3. *Els pesos utilitzen la notació* $w_{ij}^{(l)}$ *que representa el pes associat al pas d'informació de la neurona* $h_i^{(l-1)}$ *cap a la neurona* $h_j^{(l)}$*. Els biaixos utilitzen la notació* $b_j^{(l)}$*, que indica que s'aplica en el pas d'informació a la neurona* $h_j^{(l)}$*.*

Observació 4. *De vegades, per simplificar la notació, considerarem que* $x_i^{(j)} = h_{j_i}^{(0)}$*. I de la mateixa manera,* $y_i^{(j)} = h_{j_i}^{(H+1)}$*, on* H *és el total de capes ocultes.*

Formalment, el procés descrit en la Figura 6 s'anomena propagació cap endavant, i es fa iterativament per passar la informació de neurona en neurona avançant per les capes de la xarxa neuronal.

Definició 11. La **propagació cap endavant** és el procés en el qual les dades es mouen a través de la xarxa, començant en la capa d'entrada quan s'introdueix el vector de dades $x^{(j)}$ fins a arribar a la capa de sortida quan s'obté el vector $\tilde{f}(x^{(j)}) = \tilde{y}^{(j)}$.

Bàsicament, en cada neurona $h_k^{(l)}$ es reben les entrades de $h^{(l-1)}$, es calcula una suma ponderada $z_k^{(l)}$ amb els pesos associats, $w_{ik}^{(l)}$ i $b_k^{(l)}$, i s'aplica una funció d'activació $\varphi(z_k^{(l)})$ per obtenir aquesta neurona $h_k^{(l)}$. Aleshores, fent propagació cap endavant cada neurona es computa com:

$$h_k^{(l)} = \varphi(z_k^{(l)}), \text{ on } z_k^{(l)} = \sum_i w_{ik}^{(l)} h_i^{(l-1)} + b_k^{(l)}.$$

Un cop s'ha completat la propagació cap endavant, les neurones de la capa de sortida ens donen el resultat de la funció que s'està tractant d'aproximar.

Depenent de quina sigui la situació en la qual estiguem utilitzant xarxes neuronals com a eina per aproximar una funció, la qualitat de l'aproximació i la precisió variaran. Però, gràcies al **Teorema Universal d'Aproximació** [Les+93], sota unes condicions bàsiques, podem assegurar que una xarxa neuronal serà capaç d'aproximar qualsevol funció continua en $\mathcal{C}(\mathbb{R}^n, \mathbb{R}^m)$. Resumidament, el Teorema Universal d'Aproximació estableix que una xarxa neuronal amb com a mínim una capa oculta i una funció d'activació no-lineal, pot aproximar qualsevol funció continua fins a un nivell arbitrari de precisió.

3.1.1 L'algorisme del gradient descendent i retropropagació

Tant l'algorisme del gradient descendent com la retropropagació són passos clau en el funcionament de les xarxes neuronals. Aquests processos fan que la xarxa neuronal aprengui mitjançant la comparació entre les seves prediccions i les sortides desitjades, ajustant-se iterativament per millorar el seu rendiment al llarg del temps.

Sabem que l'objectiu és que les sortides de la xarxa neuronal $\tilde{f}(x^{(j)}) = \tilde{y}^{(j)}$ aproximin el millor possible el vector $f(x^{(j)}) = y^{(j)}$, per fer-ho utilitzarem una funció de pèrdua

$$L(\tilde{f}) : \mathbb{R}^p \to \mathbb{R}$$

que avaluï quant s'equivoquen les prediccions de la xarxa respecte a les sortides desitjades. Les variables de la funció $L(\tilde{f})$ seran els paràmetres $w_{ik}^{(l)}$ i $b_k^{(l)}$, i considerem p el total de paràmetres de la xarxa. Aleshores, buscarem minimitzar aquesta funció ajustant els pesos i biaixos. Utilitzarem l'algorisme del gradient i retropropagació per fer-ho.

L'**algorisme del gradient descendent** és un algorisme d'optimització que consisteix en tractar de trobar el mínim de la funció de pèrdua utilitzant el gradient $\nabla L(\tilde{f})$. Es fa servir la idea que el gradient ens indica la direcció del creixement més ràpid de la funció.

Per tant, matemàticament l'algorisme del gradient descendent consisteix en inicialitzar de manera aleatòria els paràmetres de la xarxa neuronal, obtenint $\tilde{f}^0$ on 0 denota el nombre de la iteració, de manera que les prediccions inicials no seran gaire bones, i aplicar iterativament la fórmula següent per actualitzar els paràmetres:

$$\tilde{f}^{i+1} = \tilde{f}^i - \lambda \cdot \nabla L(\tilde{f}^i).$$

Aquest ajust és proporcional al gradient i a una **taxa d'aprenentatge** λ, que determina la longitud del pas en la direcció del gradient.

Per calcular el gradient en l'algorisme del gradient descendent, utilitzarem la **retropropagació**. Aquest algorisme es basa en aplicar la regla de la cadena per facilitar el càlcul del gradient, això permet calcular la derivada parcial de la funció de pèrdua respecte als paràmetres $w_{ik}^{(l)}$ i $b_k^{(l)}$ de la xarxa neuronal. Utilitzar la retropropagació fa que els càlculs siguin més eficients, i a més a més, obtenim un procés recursiu que comença en l'última capa i propaga els paràmetres actualitzats cap enrere a través de la xarxa neuronal.

Per actualitzar els pesos i biaixos, volem calcular $\nabla L(\tilde{f})$ respecte els paràmetres $w_{ik}^{(l)}$ i $b_k^{(l)}$, doncs el que haurem de calcular és:

$$\nabla L(\tilde{f}) = \left(\frac{\partial L(\tilde{f})}{\partial w_{11}^{(1)}}, \frac{\partial L(\tilde{f})}{\partial w_{12}^{(1)}}, \ldots, \frac{\partial L(\tilde{f})}{\partial w_{n_H m}^{(H+1)}}, \frac{\partial L(\tilde{f})}{\partial b_1^{(1)}}, \frac{\partial L(\tilde{f})}{\partial b_2^{(1)}}, \ldots, \frac{\partial L(\tilde{f})}{\partial b_m^{(H+1)}} \right),$$

on H és el nombre total de capes ocultes i n_H denota el nombre de neurones en la capa H. Aleshores, el procés de càlcul de la retropropagació consisteix en aplicar la regla de la cadena de manera que

$$\frac{\partial L(\tilde{f})}{\partial w_{pq}^{(l)}} = \sum_k \frac{\partial L(\tilde{f})}{\partial z_k^{(l)}} \frac{\partial z_k^{(l)}}{\partial w_{pq}^{(l)}}, \qquad \text{on } \; z_k^{(l)} = \sum_i w_{ik}^{(l)} h_i^{(l-1)} + b_k^{(l)}.$$

La primera derivada parcial dependrà de la funció de pèrdua triada, però la segona la podem calcular:

$$\frac{\partial z_k^{(l)}}{\partial w_{pq}^{(l)}} = \frac{\partial (\sum_i w_{ik}^{(l)} h_i^{(l-1)} + b_k^{(l)})}{\partial w_{pq}^{(l)}} = \sum_i h_i^{(l-1)} \frac{\partial w_{ik}^{(l)}}{\partial w_{pq}^{(l)}} = \sum_i h_i^{(l-1)} \delta_{ip} \delta_{kq} = \delta_{kq} \cdot h_p^{(l-1)}.$$

on δ_{kp} representa la funció Delta de Kronecker: $\delta_{kp} = \begin{cases} 1, & \text{if } k = p, \\ 0, & \text{if } k \neq p. \end{cases}$

De la mateixa manera podem aplicar la regla de la cadena per calcular les derivades parcials respecte als biaixos:

$$\frac{\partial L(\tilde{f})}{\partial b_q^{(l)}} = \sum_k \frac{\partial L(\tilde{f})}{\partial z_k^{(l)}} \frac{\partial z_k^{(l)}}{\partial b_q^{(l)}}, \qquad \text{on } \; z_k^{(l)} = \sum_i w_{ik}^{(l)} h_i^{(l-1)} + b_k^{(l)}.$$

Calculem la segona part de la regla de la cadena:

$$\frac{\partial z_k^{(l)}}{\partial b_q^{(l)}} = \frac{\partial (\sum_i w_{ik}^{(l)} h_i^{(l-1)} + b_k^{(l)})}{\partial b_q^{(l)}} = \sum_i \frac{\partial b_k^{(l)}}{\partial b_q^{(l)}} = \delta_{kq}.$$

Finalment tenim:

$$\frac{\partial L(\tilde{f})}{\partial w_{pq}^{(l)}} = \sum_k \frac{\partial L(\tilde{f})}{\partial z_k^{(l)}} \frac{\partial z_k^{(l)}}{\partial w_{pq}^{(l)}} = \sum_k \frac{\partial L(\tilde{f})}{\partial z_k^{(l)}} \cdot (\delta_{kq} h_p^{(l-1)}) = h_p^{(l-1)} \sum_k \frac{\partial L(\tilde{f})}{\partial z_k^{(l)}} \cdot \delta_{kq} = h_p^{(l-1)} \frac{\partial L(\tilde{f})}{\partial z_q^{(l)}}$$

En el cas dels biaixos:

$$\frac{\partial L(\tilde{f})}{\partial b_q^{(l)}} = \sum_k \frac{\partial L(\tilde{f})}{\partial z_k^{(l)}} \frac{\partial z_k^{(l)}}{\partial b_q^{(l)}} = \sum_k \frac{\partial L(\tilde{f})}{\partial z_k^{(l)}} \cdot \delta_{kq} = \frac{\partial L(\tilde{f})}{\partial z_q^{(l)}}.$$

Aleshores, hem calculat

$$\frac{\partial L(\tilde{f})}{\partial w_{pq}^{(l)}} = h_p^{(l-1)} \frac{\partial L(\tilde{f})}{\partial z_q^{(l)}} \qquad \text{i} \qquad \frac{\partial L(\tilde{f})}{\partial b_q^{(l)}} = \frac{\partial L(\tilde{f})}{\partial z_q^{(l)}}.$$

Pel moment fins aquí és on podem calcular, ja que l'altra part de la regla de la cadena que faltaria per calcular depèn de la funció de pèrdua que es decideixi utilitzar.

Més endavant, en la secció 3.3.2 veurem un exemple de com s'acaben aquests càlculs i com s'apliquen en l'algorisme del gradient descendent per una funció de pèrdua concreta.

3.2 Aplicació de les xarxes neuronals a l'aprenentatge per reforç

Arran del procediment explicat en la secció 3.1.1, és clar que aquestes xarxes neuronals són una eina que pot ser aplicada amb gran facilitat per aproximar funcions i fer prediccions en contextos d'aprenentatge supervisat, ja que s'entrena la xarxa neuronal amb un conjunt de dades $\{(x^{(j)}, y^{(j)}\}$ que inclouen les entrades $x^{(j)}$ amb les seves sortides corresponents $y^{(j)}$. Però, en el cas de l'aprenentatge per reforç no disposem d'aquest conjunt de sortides desitjades, aleshores haurem d'adaptar el problema de manera que encara així la xarxa neuronal sigui capaç de millorar les seves prediccions en cada iteració.

Adaptarem les situacions d'aprenentatge per reforç tal que les entrades de la xarxa neuronal seran els estats i les sortides seran l'aproximació o bé de la funció de valor o bé de la política. És a dir, en cada passada per la xarxa neuronal, els nodes de sortida seran o el valor d'acció per cada acció possible en aquell estat o la probabilitat de prendre cada acció possible en aquell estat, això dependrà de si estem utilitzant un mètode basat en valors o un mètode basat en polítiques.

Així doncs, cada passada per la xarxa neuronal ens permetrà triar una acció per a l'estat d'entrada, i es completarà un episodi quan havent seguit una trajectòria d'estats i accions, s'arribi a la condició de parada. Un cop tinguem l'episodi li podrem calcular el retorn, ja que el nostre objectiu serà maximitzar aquest valor.

3.2.1 Procés de tria de les sortides desitjades

Ja estem familiaritzats amb els diferents elements del procés, com el funcionament de les funcions d'activació o el càlcul del gradient per l'actualització dels pesos. Però, hi ha una qüestió que ens queda per tractar, que és el fet que en l'aprenentatge per reforç no tenim un conjunt definit de sortides desitjades $y^{(t)}$.

El concepte de **'sortides desitjades'** fa referència als valors d'acció o a la distribució de probabilitats associada a cada possible acció d'un estat determinat, que voldríem que la xarxa neuronal predigués quan s'utilitza aquest estat com a entrada.

En les situacions en les quals no disposem d'un conjunt definit de sortides desitjades, optarem per fer diverses iteracions de prediccions de la xarxa neuronal. De manera que en cada iteració establirem un nou conjunt de sortides desitjades amb l'objectiu d'utilitzar-les per entrenar el model durant aquella iteració. A continuació, explicarem el procés que es segueix per definir aquest conjunt de sortides desitjades en cada iteració. Donarem l'explicació pel cas en què ens trobem en un mètode basat en polítiques, és a dir, estarem en el cas en què les sortides seran una distribució de probabilitat en les accions de l'estat.

En cada iteració farem que la xarxa neuronal produeixi n episodis, el conjunt d'aquests n episodis és anomenat lot. Després s'efectua el càlcul del retorn per a cada episodi en el lot.

Seguint l'esquema de la Figura 7, identificarem el x% dels episodis amb millor retorn, descartant la resta. El percentatge 'x' que marcarà la frontera del conjunt d'episodis que mantindrem, serà un hiperparàmetre que s'haurà de fixar en plantejar la resolució del problema, normalment aquest percentatge acostuma a estar entre el 30% i el 10%. A continuació, aquests episodis amb millor retorn que mantenim, ens permetran definir el conjunt de sortides desitjades. Per cada estat $x^{(t)}$ que es visita en aquests episodis, ens quedarem amb la distribució de probabilitat $y^{(t)}$ que retorna la xarxa neuronal, aquestes es convertiran en les sortides desitjades per a aquesta iteració, que utilitzarem per entrenar el model.

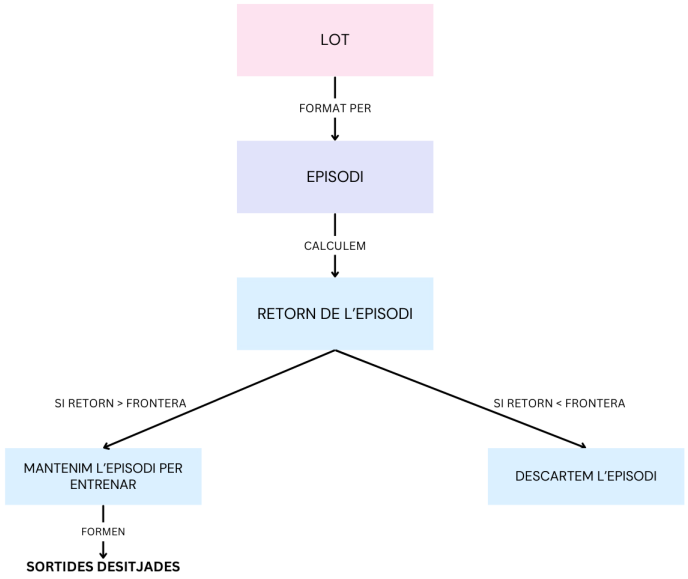


Figura 7: Tria de les sortides desitjades.

Un cop ja tenim definit un conjunt de sortides desitjades per aquella iteració, es realitza una nova passada a través de la xarxa neuronal. Les sortides d'aquesta passada per la xarxa neuronal les denotarem $\tilde{y}^{(t)}$. Aleshores, podem considerar que tenim definida una distribució $y^{(t)}$ de les sortides desitjades i una distribució $\tilde{y}^{(t)}$ de les sortides obtingudes en la nova passada per la xarxa neuronal. Això ens permet utilitzar la funció de pèrdua $L(\tilde{f})$ per comparar totes les sortides estat a estat. Finalment, s'aplica l'algorisme del gradient descendent per ajustar els pesos i biaixos de la xarxa neuronal.

De manera iterativa, es repetirà el procés per nous lots, tal que en cada iteració les prediccions milloraran significativament. Aquest procés assegura que, en un estat concret, la xarxa neuronal opti per ajustar els pesos de manera que resulti més probable que la sortida sigui la distribució que ha demostrat un millor rendiment, afavorint així l'aprenentatge del model.

3.3 Mètode de l'entropia creuada

En aquesta secció introduirem el mètode de **l'entropia creuada** (o mètode de cross-entropy en anglès). Aquest consisteix en utilitzar la funció de l'entropia creuada com a funció de pèrdua quan s'entrena la xarxa neuronal, això ens dona un mètode basat en polítiques.

El mètode de l'entropia creuada es basa en la idea de mesurar la divergència entre dues distribucions de probabilitat. En el context d'aprenentatge per reforç, aquestes dues distribucions solen ser la distribució de probabilitats de les accions predites pel model actual i la distribució de probabilitats desitjada que maximitzaria les recompenses.

La funció de l'entropia creuada que utilitzarem com a funció de pèrdua, està basada en el principi de màxima versemblança, i té la forma següent:

$$CE(P,Q) = -\sum P(x)log(Q(x))$$

per a dues distribucions de probabilitat P i Q.

En la nostra notació es tradueix a

$$CE(\tilde{y}^{(j)}, y^{(j)}) = CE(\tilde{f}(x^{(j)}), f(x^{(j)})) = -\sum_k f_k(x^{(j)}) \cdot \log\left(\tilde{f}_k(x^{(j)})\right),$$

que ens dona la pèrdua de l'entropia creuada per un vector d'entrada $x^{(j)}$.

Si volem calcular la pèrdua de tot el model tenim

$$CE(\tilde{f}) = -\frac{1}{|D|}\sum_{x^{(j)}\in D} CE(\tilde{f}(x^{(j)}), f(x^{(j)})) = -\frac{1}{|D|}\sum_{x^{(j)}\in D}\sum_k f_k(x^{(j)}) \cdot \log\left(\tilde{f}_k(x^{(j)})\right)$$

on D és el conjunt de vectors d'entrada.

Introduïm el mètode de l'entropia creuada, ja que serà el que utilitzarem per analitzar i tractar de buscar contraexemples en una conjectura en teoria de grafs. Utilitzarem aquest mètode, ja que a part de la seva simplicitat i facilitat d'implementació, és un mètode molt útil per la facilitat de càlcul del gradient de la funció i per poca sensibilitat als hiperparàmetres. A més a més, per les propietats matemàtiques del logaritme, petits errors en les prediccions poden ser penalitzats de manera significativa, afavorint la convergència ràpida cap a solucions més precises.

3.3.1 Funcions d'activació amb l'entropia creuada

Anteriorment, hem introduït el concepte de **funcions d'activació** en xarxes neuronals, aquestes s'apliquen en xarxes neuronals per introduir no-linearitats i permetre a la xarxa aprendre relacions més complexes i no lineals en les dades.Existeixen múltiples funcions d'activació diferents, que segons les seves característiques són utilitzades en unes situacions o altres, les més utilitzades són les següents.

En capes ocultes s'acostuma a utilitzar la funció **ReLU** (Rectified Linear Unit)

$$\text{ReLU}(z_k^{(l)}) = \max(0, z_k^{(l)}),$$

ja que permet introduir no-linearitats de manera senzilla i els càlculs resulten molt eficients.

Una altra funció d'activació sovint utilitzada en capes ocultes és la **funció tangent hiperbòlica**:

$$\tanh(z_k^{(l)}) = \frac{e^{(z_k^{(l)})} - e^{-(z_k^{(l)})}}{e^{(z_k^{(l)})} + e^{-(z_k^{(l)})}},$$

que retorna sortides tal que $\tanh(z_k^{(l)}) = h_k^{(l)} \in [-1, 1]$, així doncs és útil en capes ocultes per mantenir les sortides centrades al voltant de zero.

Per una altra banda, per la capa de sortida s'acostuma a utilitzar la funció Sigmoide o bé la funció Softmax. La funció **Sigmoide** s'utilitza en l'última capa quan es vol obtenir sortides

entre 0 i 1, com per exemple probabilitats.

$$\text{sigmoid}(z_k^{(H+1)}) = \frac{1}{1+e^{z_k^{(H+1)}}},$$

de manera que: $\text{sigmoid}(z_k^{(H+1)}) = h_k^{(H+1)} = \tilde{y}_k^{(j)} \in [0,1]$.

La funció **Softmax** és una funció d'activació que retorna les sortides en forma de distribució de probabilitat. La seva fórmula és:

$$\text{softmax}(z_k^{(H+1)}) = \frac{e^{z_k^{(H+1)}}}{\sum_i e^{z_i^{(H+1)}}}$$

doncs, veiem que es compleix que: $\text{softmax}(z_k^{(H+1)}) = h_k^{(H+1)} = \tilde{y}_k^{(j)} \in [0,1]$ i a més $\sum_k \tilde{y}_k^{(j)} = 1$.

En el nostre cas, és a dir, utilitzant la funció de pèrdua de l'entropia creuada, la funció d'activació més convenient per la capa de sortida és la funció Softmax. La funció Softmax retorna les sortides de la xarxa neuronal en forma de distribució de probabilitat, que és el que necessitem passar a la funció de pèrdua per calcular l'error en les prediccions de la xarxa neuronal.

Per les capes ocultes, utilitzarem la funció d'activació ReLU, ja que és una funció que ens permet afegir no-linearitat de manera simple.

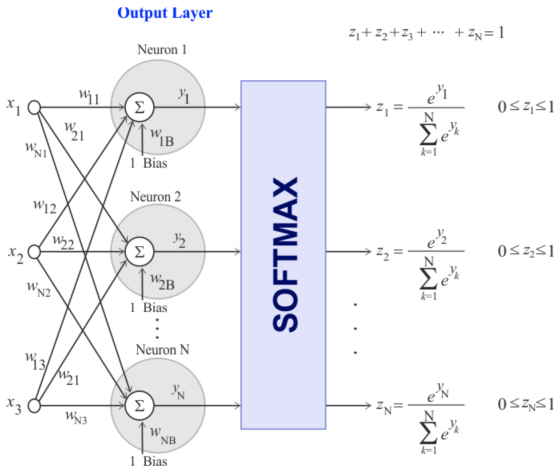


Figura 8: Funció Softmax en una xarxa neuronal [Win21].

3.3.2 Retropropagació i l'algorisme del gradient descendent amb l'entropia creuada

Anteriorment, en l'apartat 3.1.1 hem definit la retropropagació i l'algorisme del gradient descendent. A més a més, hem començat el càlcul del gradient de la funció de pèrdua utilitzant retropropagació fins arribar a

$$\frac{\partial L(\tilde{f})}{\partial w_{pq}^{(l)}} = h_p^{(l-1)} \frac{\partial L(\tilde{f})}{\partial z_q^{(l)}} \quad \text{i} \quad \frac{\partial L(\tilde{f})}{\partial b_q^{(l)}} = \frac{\partial L(\tilde{f})}{\partial z_q^{(l)}}$$

Així que a continuació acabarem de calcular aquestes derivades parcials en el cas en què s'utilitza la funció de pèrdua de l'entropia creuada, aquests càlculs estan basats en els realitzats en [Pun].

Utilitzarem la funció de pèrdua de l'entropia creuada de manera que:

$$\frac{\partial CE}{\partial w_{pq}^{(l)}} = h_p^{(l-1)} \frac{\partial CE}{\partial z_q^{(l)}} \qquad \text{i} \qquad \frac{\partial CE}{\partial b_q^{(l)}} = \frac{\partial CE}{\partial z_q^{(l)}} \tag{3.1}$$

Per l'entropia creuada utilitzarem la notació següent:

$$CE(\tilde{y}^{(j)}, y^{(j)}) = -\sum_k y_k^{(j)} log(\tilde{y}_k^{(j)})$$

on $y^{(j)}$ és el vector de valors de la predicció desitjada i $\tilde{y}^{(j)}$ és el vector de sortides calculades per la xarxa neuronal.

Per simplificar utilitzarem la notació $\tilde{y}_k^{(j)} = h_{j_k}^{(H+1)}$, on

$$h_{j_k}^{(H+1)} = \varphi(z_k^{(H+1)}) = \varphi(\sum_i w_{ik}^{(l)} h_i^{(l-1)} + b_k^{(l)}),$$

podem no utilitzar el sufix j ja que no tindrà influència en els càlculs quina mostra de les dades estiguem utilitzant.

Per calcular necessitem utilitzar la funció d'activació φ així que separarem el càlcul en dos casos. El cas en què ens trobem en la capa de sortida i utilitzem la funció d'activació Softmax. I després el cas en què ens trobem en una capa oculta i utilitzem una funció d'activació φ.

Softmax:
Primer realitzarem els càlculs amb Softmax

$$\tilde{y}_k^{(j)} = h_{j_k}^{(H+1)} = \varphi(z_k^{(H+1)}) = \frac{e^{z_k^{(H+1)}}}{\sum_i e^{z_i^{(H+1)}}}$$

llavors substituint-ho en la funció de pèrdua de l'entropia creuada tenim:

$$CE(\tilde{y}^{(j)}, y^{(j)}) = -\sum_k y_k^{(j)} log(\tilde{y}_k^{(j)}) = -\sum_k y_k^{(j)} log(\frac{e^{z_k^{(H+1)}}}{\sum_i e^{z_i^{(H+1)}}}) = -\sum_k y_k^{(j)} (z_k^{(H+1)} - log(\Omega))$$

on $\Omega = \sum_i e^{z_i^{(H+1)}}$. Aleshores, ara calcularem la derivada parcial que ens falta en 3.1

$$\frac{\partial CE}{\partial w_{pq}^{(H+1)}} = h_p^{(H)} \frac{\partial CE}{\partial z_q^{(H+1)}} \tag{3.2}$$

tenint en compte que estem en la capa de sortida i doncs $l = H + 1$,

$$\begin{aligned}\frac{\partial CE}{\partial z_q^{(H+1)}} =& \frac{\partial(-\sum_k y_k^{(j)}(z_k^{(H+1)} - log(\Omega)))}{\partial z_q^{(H+1)}} = -\sum_k y_k^{(j)} \frac{\partial(z_k^{(H+1)} - log(\Omega))}{\partial z_q^{(H+1)}} \\ =& -\sum_k y_k^{(j)} \left(\delta_{kq} - \frac{1}{\Omega} \frac{\partial \Omega}{\partial z_q^{(H+1)}}\right)\end{aligned}$$

on δ_{kq} representa la funció Delta de Kronecker: $\delta_{kq} = \begin{cases} 1, & \text{if } k = q, \\ 0, & \text{if } k \neq q. \end{cases}$

Per una altra banda, sabem que $\Omega = \sum_i e^{z_i^{(H+1)}}$, aleshores

$$\frac{\partial \Omega}{\partial z_q^{(H+1)}} = \frac{\partial (\sum_i e^{z_i^{(H+1)}})}{\partial z_q^{(H+1)}} = \sum_i e^{z_i^{(H+1)}} \delta_{iq} = e^{z_q^{(H+1)}}.$$

Així, substituint aquesta igualtat i el fet que hem definit que $\tilde{y}_k^{(j)} = \frac{e^{z_q^{(H+1)}}}{\sum_i e^{z_i^{(H+1)}}}$, tenim que:

$$\begin{aligned}\frac{\partial CE}{\partial z_q^{(H+1)}} &= -\sum_k y_k^{(j)} \left(\delta_{kq} - \frac{1}{\Omega}\frac{\partial \Omega}{\partial z_q^{(H+1)}}\right) = -\sum_k y_k^{(j)} \left(\delta_{kq} - \frac{1}{\Omega} e^{z_q^{(H+1)}}\right) = -\sum_k y_k^{(j)} \left(\delta_{kq} - \tilde{y}_q^{(j)}\right) \\ &= \sum_k y_k^{(j)} \left(\tilde{y}_q^{(j)} - \delta_{kq}\right) = \tilde{y}_q^{(j)} \left(\sum_k y_k^{(j)}\right) - y_q^{(j)} = \tilde{y}_q^{(j)} \tau - y_q^{(j)}, \qquad \text{on } \tau = \sum_k y_k^{(j)}.\end{aligned} \tag{3.3}$$

Però, veiem que $\tau = \sum_k y_k^{(j)} = 1$, ja que hem considerat que $y^{(j)}$ és una distribució de probabilitat. Per tant,

$$\frac{\partial CE}{\partial z_q^{(H+1)}} = \tilde{y}_q^{(j)} - y_q^{(j)}. \tag{3.4}$$

Finalment, podem substituir en (3.2) aquesta derivada parcial

$$\frac{\partial CE}{\partial w_{pq}^{(H+1)}} = h_p^{(H)} \frac{\partial CE}{\partial z_q^{(H+1)}} = h_p^{(H)} (\tilde{y}_q^{(j)} - y_q^{(j)}), \qquad \text{on } \tilde{y}_q^{(j)} = h_{j_q}^{(H+1)}.$$

Per tant, el gradient de la funció de pèrdua respecte al pes $w_{pq}^{(H+1)}$ és

$$\boxed{\frac{\partial CE}{\partial w_{pq}^{(H+1)}} = h_p^{(H)} (\tilde{y}_q^{(j)} - y_q^{(j)}),} \tag{3.5}$$

per p i q índexs qualsevols de les neurones de sortida i arribada del pes. I considerant que estem en una mostra j del conjunt de dades.

Utilitzant el mateix procediment, podem calcular el gradient de la funció de pèrdua respecte del biaix en 3.1:

$$\frac{\partial CE}{\partial b_q^{(H+1)}} = \frac{\partial CE}{\partial z_q^{(H+1)}}. \tag{3.6}$$

Dels càlculs anteriors que acaben en (3.4), sabem que

$$\frac{\partial CE}{\partial b_q^{(H+1)}} = \frac{\partial CE}{\partial z_q^{(H+1)}} = \tilde{y}_q^{(j)} - y_q^{(j)}.$$

Per tant, el gradient de la funció de pèrdua respecte al biaix $b_q^{(H+1)}$ és

$$\boxed{\frac{\partial CE}{\partial b_q^{(H+1)}} = \tilde{y}_q^{(j)} - y_q^{(j)},} \tag{3.7}$$

on q és l'índex de la neurona d'arribada de la suma ponderada amb els pesos i el biaix $b_q^{(H+1)}$.

Hem calculat utilitzant retropropagació les derivades parcials de la funció de pèrdua de l'en-

tropia creuada respecte als paràmetres de la capa de sortida $w_{pq}^{(H+1)}$ i $b_q^{(H+1)}$.

Aleshores aplicant aquestes derivades parcials a l'algorisme del gradient descendent, les equacions d'actualització dels pesos i biaixos de la capa de sortida seran:

$$\boxed{\begin{aligned} w_{ij}^{(H+1)} &= w_{ij}^{(H+1)} - \lambda \cdot h_p^{(H)}(\tilde{y}_q^{(j)} - y_q^{(j)}), \\ b_j^{(H+1)} &= b_j^{(H+1)} - \lambda \cdot \tilde{y}_q^{(j)} - y_q^{(j)}. \end{aligned}}$$

Així doncs, veiem que les equacions per actualitzar els paràmetres de la capa de sortida del model, és a dir, els pesos i biaixos de l'última capa de la xarxa neuronal, depenen de la diferència que hi hagi entre els valors de les dues distribucions de probabilitat: les sortides predites $\tilde{y}^{(j)}$ i les sortides desitjades $y^{(j)}$. Així que en principi progressivament en cada iteració el canvi serà menor.

Capes ocultes:
Farem els càlculs per actualitzar els paràmetres de les capes ocultes, suposant que s'utilitza una funció d'activació φ. Tractarem de calcular les derivades parcials pels pesos i biaixos de la capa H utilitzant les derivades parcials ja calculades de la capa $H+1$.
Sabem que

$$\frac{\partial CE}{\partial w_{pq}^{(H)}} = \sum_k \frac{\partial CE}{\partial z_k^{(H)}} \frac{\partial z_k^{(H)}}{\partial w_{pq}^{(H)}}. \tag{3.8}$$

Donarem la primera derivada parcial utilitzant $\frac{\partial CE}{\partial z_k^{(H+1)}}$ que ja hem calculat anteriorment:

$$\frac{\partial CE}{\partial z_k^{(H)}} = \frac{\partial CE}{\partial z_k^{(H+1)}} \cdot \frac{\partial z_k^{(H+1)}}{\partial h_k^{(H)}} \cdot \frac{\partial h_k^{(H)}}{\partial z_k^{(H)}}, \tag{3.9}$$

i tenim

$$\frac{\partial CE}{\partial z_k^{(H+1)}} = \tilde{y}_k^{(j)} - y_k^{(j)}.$$

A més sabem que

$$z_k^{(H+1)} = \sum_i w_{ik}^{(H+1)} h_i^{(H)} + b_k^{(H+1)} \implies \frac{\partial z_k^{(H+1)}}{\partial h_k^{(H)}} = \frac{\partial(\sum_i w_{ik}^{(H+1)} h_i^{(H)})}{h_k^{(H)}} = w_{kk}^{(H+1)}.$$

I per una altra banda,

$$h_k^{(H)} = \varphi(z_k^{(H)}) \implies \frac{\partial h_k^{(H)}}{\partial z_k^{(H)}} = \varphi'(z_k^{(H)}).$$

Posant aquests càlculs junts en 3.9 obtenim

$$\frac{\partial CE}{\partial z_k^{(H)}} = (\tilde{y}_k^{(j)} - y_k^{(j)}) \cdot w_{kk}^{(H+1)} \cdot \varphi'(z_k^{(H)}) \tag{3.10}$$

Calculem la segona derivada parcial de la regla de la cadena en 3.8:

$$\frac{\partial z_k^{(H)}}{\partial w_{pq}^{(H)}} = \frac{\partial(\sum_i w_{ik}^{(H)} h_i^{(H-1)} + b_k^{(H)})}{\partial w_{pq}^{(H)}} = \sum_i h_i^{(H-1)} \frac{\partial w_{ik}^{(H)}}{\partial w_{pq}^{(H)}} = \sum_i h_i^{(H-1)} \delta_{ip} \delta_{kq} = \delta_{kq} \cdot h_p^{(H-1)}$$

on δ_{kp} representa la funció Delta de Kronecker.

Substituïm tot en 3.8

$$\frac{\partial CE}{\partial w_{pq}^{(H)}} = \sum_k \frac{\partial CE}{\partial z_k^{(H)}} \frac{\partial z_k^{(H)}}{\partial w_{pq}^{(H)}} = \sum_k (\tilde{y}_k^{(j)} - y_k^{(j)}) \cdot w_{kk}^{(H+1)} \cdot \varphi'(z_k^{(H)}) \cdot (\delta_{kq} \cdot h_p^{(H-1)})$$
$$= (\tilde{y}_q^{(j)} - y_q^{(j)}) \cdot w_{qq}^{(H+1)} \cdot \varphi'(z_q^{(H)}) \cdot h_p^{(H-1)}$$

Per els biaixos podem fer el càlcul directament ja que ja hem calculat en 3.10 que

$$\frac{\partial CE}{\partial z_k^{(H)}} = (\tilde{y}_k^{(j)} - y_k^{(j)}) \cdot w_{kk}^{(H+1)} \cdot \varphi'(z_k^{(H)})$$

i sabem que

$$\frac{\partial z_k^{(H)}}{\partial b_q^{(H)}} = \frac{\partial (\sum_i w_{ik}^{(H)} h_i^{(H-1)} + b_k^{(H)})}{\partial b_q^{(H)}} = \delta_{kq}.$$

Doncs podem utilitzar aquestes dues derivades parcials per aplicar la regla de la cadena següent:

$$\frac{\partial CE}{\partial b_q^{(H)}} = \sum_k \frac{\partial CE}{\partial z_k^{(H)}} \frac{\partial z_k^{(H)}}{\partial b_q^{(H)}} = \sum_k (\tilde{y}_k^{(j)} - y_k^{(j)}) \cdot w_{kk}^{(H+1)} \cdot \varphi'(z_k^{(H)}) \cdot \delta_{kq} = (\tilde{y}_q^{(j)} - y_q^{(j)}) \cdot w_{qq}^{(H+1)} \cdot \varphi'(z_q^{(H)})$$

Això ens permet veure que per calcular el gradient de la funció de pèrdua respecte dels paràmetres en la capa H, ens cal tenir els paràmetres actualitzats $w_{qq}^{(H+1)}$ de la capa $H+1$. Per tant, com hem comentat en la retropropagació, podem repetir aquest procés iterativament de manera que anirem aplicant els paràmetres actualitzats de cada capa per actualitzar els de les capes anteriors.

Hem calculat les equacions de l'algorisme del gradient descendent per un cas concret en el qual s'utilitza l'entropia creuada com a funció de pèrdua i Softmax com a funció d'activació. Si les funcions fossin unes altres, els resultats variarien lleugerament.

Capítol 4

Part Pràctica

Per finalitzar el treball tractarem de posar en pràctica un mètode d'aprenentatge per reforç, aquest serà el descrit en la secció 3.3, que utilitza la funció de pèrdua de l'entropia creuada. L'objectiu serà de veure com definir els diferents elements i conceptes que hem explicat en els anteriors capítols, en una situació concreta. A més a més, veurem quin és el procés que segueix el model per aprendre.

Per fer això, voldrem que l'ordinador trobi un contraexemple a una conjectura de teoria de grafs. Aquesta conjectura va ser proposada en [GA08]:

Conjectura 1. *El graf complet K_n té el major edge-Szeged índex (índex Szeged per arestes) d'entre tots els grafs simples amb n vèrtexs.*

Poc temps més tard aquesta va ser refutada en [Vuk09], on es dona una descripció d'una família de grafs que eventualment contradiuen la conjectura. Per tant, el nostre propòsit serà de trobar exemples concrets de grafs que refuten aquesta conjectura amb un ordre molt més petit.

4.1 Teoria de Grafs

Abans d'estudiar la conjectura i com podem aplicar l'aprenentatge per reforç, cal introduir una sèrie de conceptes sobre teoria de grafs. Per aquesta secció s'ha utilitzat la referència [Cat11].

La **teoria de grafs** és una branca de les matemàtiques que estudia els grafs, estructures matemàtiques utilitzades per a modelitzar relacions entre parelles d'objectes.

Definició 12. Un **graf** és un parell ordenat $G = (V, E)$, on V és un conjunt d'elements anomenats vèrtexs i E és un conjunt format per parelles no ordenades $\{v_1, v_2\}$ de vèrtexs diferents, anomenats arestes.

Aquests grafs es representen típicament mitjançant una sèrie de punts (els vèrtexs) connectats per línies (les arestes), que componen estructures de la forma següent:

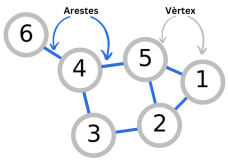


Figura 9: Exemple de graf.

El nombre de vèrtexs $|V|$ i el nombre d'arestes $|E|$ d'un graf, s'anomenen respectivament **ordre** i **mida** del graf.

Definició 13. Anomenem **camí** a una seqüència de vèrtexs d'un graf tal que hi ha una aresta que connecta cada vèrtex i el següent.

Formalment, denotarem un camí com un graf P amb vèrtexs $V(P) = \{v_0, v_1, \cdots, v_l\}$ i arestes $E(P) = \{\{v_0, v_1\}, \{v_1, v_2\}, \ldots, \{v_{l-1}, v_l\}\}$. La longitud del camí és el nombre d'arestes, $|E(P)|$.

Observació 5. *Hi ha altres conceptes que es poden considerar en teoria de grafs, com els bucles (que són arestes que connecten un vèrtex amb ell mateix), arestes múltiples (quan hi ha més d'una aresta connectant dos vèrtexs) o arestes dirigides (arestes que tenen un vèrtex de sortida i un d'arribada definits). Però, per la definició de graf i la notació triada, els grafs que considerarem no podran tenir cap d'aquests elements.*

És a dir, d'ara en endavant només parlarem de grafs simples, que són aquells que no tenen bucles ni arestes múltiples, i grafs no dirigits que són els que no tenen arestes dirigides.

A continuació, introduirem algunes definicions o conceptes que cal tenir presents per entendre la conjectura a la qual tractarem de trobar un contraexemple:

Observació 6. *La màxima mida d'un graf simple d'ordre n és $\frac{n(n-1)}{2}$.*

Definició 14. Direm que un graf és **complet** si es tracta d'un graf simple on hi ha una aresta connectant cada parell de vèrtexs.

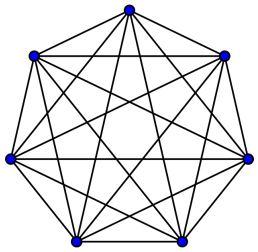


Figura 10: Exemple de graf complet.

Notació 4. *Denotarem K_n el graf complet amb n vèrtexs, que aleshores tindrà $\frac{n(n-1)}{2}$ arestes.*

A continuació, introduirem una sèrie de nocions que són les que ens permeten calcular distàncies en els grafs. Necessitarem aquestes nocions per treballar amb la conjectura:

Definició 15. En un graf, la **distància entre dos vèrtexs** del graf és el menor nombre d'arestes que hem de recórrer en un camí per connectar aquells dos vèrtexs.

És a dir, si tenim dos vèrtexs u i w la seva distància d_v serà:

$$d_v(u, w) = \text{nombre d'arestes en el camí més curt entre } u \text{ i } w.$$

Que també es pot escriure com:

$$d_v(u, w) = \min_{P \in P_{uw}} |E(P)| \quad \text{on } P \in P_{uw} \text{ si } V(P) = \{u, \cdots, w\}.$$

Podem ampliar aquest concepte a la distància entre un vèrtex i una aresta del graf.

Definició 16. La **distància entre un vèrtex i una aresta**, si tenim el vèrtex $u \in V$ i l'aresta $e = \{x, y\} \in E$, es defineix com:

$$d_e(e, u) = \min\{d_v(x, u), d_v(y, u)\}.$$

És a dir, és la mínima distància entre les dues que queden definides en considerar l'aresta com una composició de dos vèrtexs, i calcular la distància entre cadascun dels vèrtexs de l'aresta amb el vèrtex original.

Ara ja podem definir l'índex Szeged i la seva extensió l'edge-Szeged índex, que és el que utilitzarem en la conjectura:

Definició 17. L'**índex Szeged** d'un graf G es defineix com:

$$Sz(G) = \sum_{\{u,v\} \in E(G)} n_u(uv|G) \cdot n_v(uv|G),$$

on $n_u(uv|G)$ i $n_v(uv|G)$ són respectivament el nombre de vèrtexs de G que es troben més a prop del vèrtex u que del vèrtex v i el nombre de vèrtexs de G que es troben més a prop del vèrtex v que del vèrtex u.

Definició 18. L'**índex edge-Szeged** (l'índex Szeged per arestes) és una modificació de l'anterior índex però tenint en compte la distància de les arestes respecte a els vèrtexs, que es defineix com:

$$Sz_e(G) = \sum_{\{u,v\} \in E(G)} m_u(uv|G) \cdot m_v(uv|G),$$

on $m_u(uv|G)$ i $m_v(uv|G)$ són respectivament el nombre d'arestes de G que es troben més a prop del vèrtex u que del vèrtex v i el nombre d'arestes de G que es troben més a prop del vèrtex v que del vèrtex u.

Observació 7. *Notem que les arestes equidistants a u i v no són contades. Doncs, quan diem 'és l'aresta més a prop del vèrtex u' és considerat en el sentit estricte.*

Cal destacar que aquests dos índexs tenen importància en aplicacions fora de la teoria de grafs. El Szeged-índex i l'edge-Szeged índex són rellevants en bioquímica [WL12] perquè permeten descriure la connectivitat estructural d'una molècula. Així doncs, proporcionen informació útil sobre les propietats moleculars que poden influir en funcions biològiques.

Per tant, el fet que una màquina pugui ser capaç de refutar conjectures sobre les característiques d'aquests índexs, pot tenir una influència més enllà del simple estudi de certes propietats només aplicables en teoria de grafs, també podria ser útil en el camp de la bioquímica.

4.2 Aplicació a l'edge-Szeged índex

Com ja hem mencionat anteriorment, l'objectiu de la part pràctica d'aquest treball consisteix en provar que podem definir un mètode d'aprenentatge per reforç que vagi millorant el seu rendiment a partir de l'experiència, de manera que acabi trobant un graf que refuti una conjectura de teoria de grafs. Presentem la conjectura a continuació:

Conjectura 1. *El graf complet K_n té el major edge-Szeged índex (índex Szeged per arestes) d'entre tots els grafs simples amb n vèrtexs.*

Doncs, el propòsit serà trobar un graf simple amb n vèrtexs tal i que el seu edge-Szeged índex sigui més gran que el del graf complet K_n.

Observació 8. *Ha sigut demostrat que $Sz_e(K_n) = \frac{1}{2}n(n-1)^3$. [Vuk09]*

Per tant,

$$\text{Objectiu: trobar un graf simple G tal que } Sz_e(G) > Sz_e(K_n) = \frac{1}{2}n(n-1)^3.$$

Per poder plantejar aquesta situació com un problema que es resolgui amb un model d'aprenentatge automàtic, primer haurem de definir quines seran les entrades, les sortides, la funció de pèrdua i la funció de recompenses.

Primerament, com acabem de definir, la nostra funció de recompenses que buscarem maximitzar serà:

$$Sz_e(G) = \sum_{\{u,v\}\in E(G)} m_u(uv|G) \cdot m_v(uv|G).$$

D'aquesta manera, trobarem un contraexemple si en maximitzar la funció $Sz_e(G)$ es supera el valor fix de $Sz_e(K_n)$.

Seguint el plantejament en [Wag21], les entrades de la nostra xarxa neuronal seran dos vectors que descriuran l'estat. Aquests dos vectors tenen longitud $\frac{n(n-1)}{2}$, que ja hem vist que és el nombre màxim d'arestes que pot haver-hi en un graf simple d'ordre n, i estaran formats exclusivament per 0 i 1. El primer vector servirà per indicar l'estat actual del graf, hi haurà un 0 en aquelles posicions del vector que representin les arestes que no estan al graf i un 1 en aquelles que representin les arestes que l'agent sí ha decidit agafar. El segon vector servirà per indicar quina aresta considerem en aquesta passada per la xarxa neuronal, així serà un vector amb un 0 en totes les posicions, excepte en la posició que indiqui l'aresta que considerem, que tindrà un 1. Les sortides de la xarxa neuronal ens indicaran l'acció a prendre, que pot ser o bé incloure l'aresta al graf o no. Per tant, la sortida serà simplement la probabilitat de prendre l'aresta.

A més a més, sabem que serà un model seqüencial, de manera que quan obtenim una sortida l'aplicarem al graf (incloent-hi l'aresta o no), i aquest nou estat que obtenim és el que tornem a passar a la xarxa neuronal com a entrada, considerant una nova aresta. Així contínuament passarem l'estat actualitzat com a entrada fins a haver considerat totes les $\frac{n(n-1)}{2}$ possibles arestes.

Com ja hem comentat, utilitzarem la funció de pèrdua de l'entropia creuada:

$$CE(\tilde{f}) = -\sum_k f_k(x^{(j)}) \cdot \log\left(\tilde{f}_k(x^{(j)})\right),$$

i la funció d'activació Softmax o Sigmoide en l'última capa, ja que ambdues retornen valors entre 0 i 1, aleshores en xarxes neuronals en què només hi ha una classe de sortida podem utilitzar qualsevol de les dues. Aquestes ens donaran la probabilitat $\tilde{f}_k(x^{(j)}) = \varphi(z_k^{(H+1)})$ predita per la xarxa neuronal:

$$\text{Softmax: } \varphi(z_k^{(H+1)}) = \frac{e^{z_k^{(H+1)}}}{\sum_i e^{z_i^{(H+1)}}} \quad \text{o} \quad \text{Sigmoide: } \varphi(z_k^{(H+1)}) = \frac{1}{1+e^{z_k^{(H+1)}}}.$$

En les capes ocultes, utilitzarem com a funció d'activació la funció ReLU:

$$\text{ReLU}(z_k^{(l)}) = \max(0, z_k^{(l)}),$$

ja que és una funció simple, però que ens permet afegir no-linearitat, que és l'objectiu principal de les funcions d'activació.

La funció d'activació Softmax ens permet afegir variació als grafs que formarà la xarxa neuronal, i així balancejar el dilema d'exploració i explotació mencionat en l'Observació 3. Softmax ens retorna una distribució de probabilitat sobre les accions a prendre, d'aquesta manera ens permet definir que la probabilitat de seleccionar cada acció correspongui a aquella que ens retorna la xarxa neuronal. Aquest enfocament permet l'exploració de diverses accions, fins i tot aquelles que no han demostrat ser les més recompensadores fins al moment. Però al mateix temps, el nombre de cops que es triarà cadascuna serà proporcional a la recompensa, aleshores això facilita la continuació de l'explotació d'accions que han demostrat ser més favorables per a l'agent en termes de recompensa.

Per tant, interpretem la probabilitat que ens retorna la xarxa neuronal de prendre una aresta, com la freqüència de vegades que l'inclourem. És a dir, si obtenim un $0,6$ de probabilitat de prendre l'aresta, aleshores el 60% de vegades passarem al següent estat el graf incloent l'aresta i el 40% dels cops sense incloure-la.

4.3 Codi

El codi utilitzat per trobar el contraexemple en Python, està basat en el codi realitzat en [Wag]. S'utilitza la mateixa base, però aplicant una sèrie de modificacions en les parts més importants, que permeten assolir l'objectiu plantejat.

A continuació, veurem aquestes seccions del codi que són més importants, en les quals s'apliquen els elements definits anteriorment.

- Secció en la qual es calculen les distàncies entre els diferents elements d'un graf:

```
#DISTANCIA ENTRE UN VERTEX I UN ALTRE
def leastDistanceV(graph, source, altre):
    Q = Queue()
    # create a dictionary with large distance(infinity) of each vertex from
    source
    distance = {k: 9999999 for k in graph.nodes()}
    visited_vertices = set()
    Q.put(source)
    visited_vertices.update({0})
    while not Q.empty():
        vertex = Q.get()
        if vertex == source:
            distance[vertex] = 0
        for u in graph[vertex]:
            if u not in visited_vertices:
                # actualitzem la distancia
                if distance[u] > distance[vertex] + 1:
                    distance[u] = distance[vertex] + 1
                Q.put(u)
                visited_vertices.update({u})
    return distance[altre]
```

Listing 4.1: Codi per calcular la distància entre dos vèrtexs.

Calculem la distància entre dos vèrtexs, ja que és necessària per a calcular la distància entre una aresta i un vèrtex.

```
#DISTANCIA ENTRE UNA ARESTA I UN VERTEX (l'aresta s'ha d'introduir per els
    seus vertexs)
def leastDistanceE(graph, edgeV1, edgeV2, vertex):
    if leastDistanceV(graph, vertex, edgeV1)< leastDistanceV(graph, vertex,
    edgeV2):
        distanceEV=leastDistanceV(graph, vertex, edgeV1)
```

```
    if leastDistanceV(graph, vertex, edgeV1)> leastDistanceV(graph, vertex,
    edgeV2):
        distanceEV=leastDistanceV(graph, vertex, edgeV2)
    else: distanceEV=leastDistanceV(graph, vertex, edgeV1)
    return distanceEV
```

Listing 4.2: Codi per calcular la distància entre un vèrtex i una aresta.

Comprovem quin dels dos vèrtexs de l'aresta té una distància menor respecte al vèrtex fix, i definim aquesta com la distància entre l'aresta i el vèrtex.

- Secció en la qual es calcula l'edge-Szeged índex:

```
#CALCUL DEL EDGE-SZEGED INDEX:
    Sze = 0
    for e in G.edges:
        n1=0
        n2=0
        for e2 in G.edges:
            if leastDistanceE(G, e2[0],e2[1],e[0])-leastDistanceE(G, e2[0],
    e2[1],e[1])<0: n1=n1+1
            if leastDistanceE(G, e2[0],e2[1],e[0])-leastDistanceE(G, e2[0],
    e2[1],e[1])>0: n2=n2+1
        Sze=Sze+n1*n2
```

Listing 4.3: Codi per calcular l'edge-Szeged índex.

Definim dos valors n_1 i n_2 que representen els comptadors d'arestes que es trobaran més a prop de cadascun dels dos vèrtexs que estem considerant. Després de comprovar totes les arestes amb aquests dos vèrtexs fixats, realitzem el mateix procés canviant de vèrtexs.

- Secció en la qual indiquem que s'imprimeixi un contraexemple si es troba:

```
#COMPARACIO AMB SZ(KN)
myScore=1/2*N*(N-1)**3 -Sze-80
    if myScore<0:
        #Hem trobat un contraexemple. L'imprimim:
            print(state)
            np.set_printoptions(threshold=np.inf)
            incidence_matrix = nx.incidence_matrix(G, oriented=False)
            with open("resultat_contraexemple"+str(N)+".txt", "w") as fp:
                fp.write("s'ha trobat un contraexemple! \n")
                fp.write("myScore:"+str(myScore)+"\n")
                fp.write("Sze:"+str(Sze)+"\n")
                fp.write("State:"+str(state)+"\n")
                fp.write("Matriu d'incidencia:\n")
                fp.write(str(incidence_matrix.toarray()))
            nx.draw(G)
            plt.savefig("resultat_contraexemple"+str(N)+".png")
            plt.show()
            exit()
    return myScore
```

Listing 4.4: Codi per imprimir el contraexemple.

Un cop ja tenim calculat el $Sz_e(G)$, aleshores definim una funció en la qual es resta aquest valor al valor fix de $Sz_e(K_n)$, és a dir, myScore $= Sz_e(K_n) - Sz_e(G)$. Aleshores, si myScore < 0, voldrà dir que hem trobat un graf G tal que $Sz_e(K_n) < Sz_e(G)$, i així hem trobat el contraexemple que buscàvem. Quan es troba aquest contraexemple indiquem al codi que es creïn dos documents: un .txt amb les informacions bàsiques com l'edge-Szeged índex del graf i la matriu d'incidència, entre altres, i també es crea un .png amb la foto del graf que és un contraexemple de la conjectura.

- Secció en la qual es defineix la xarxa neuronal:

```
model = Sequential()
model.add(Dense(FIRST_LAYER_NEURONS,  activation="relu"))
model.add(Dense(SECOND_LAYER_NEURONS, activation="relu"))
model.add(Dense(THIRD_LAYER_NEURONS, activation="relu"))
model.add(Dense(1, activation="sigmoid"))
model.build((None, observation_space))
model.compile(loss="binary_crossentropy", optimizer=SGD(learning_rate =
    LEARNING_RATE))
```

Listing 4.5: Codi per definir la xarxa neuronal.

Definim una xarxa neuronal seqüencial amb tres capes ocultes, funció d'activació Sigmoide (o Softmax) a la capa de sortida i en les altres ReLU. Com a funció de pèrdua definim 'binary cross-entropy', i com a optimitzador 'SGD' que vol dir 'Stochastic Gradient Descent', que és un tipus d'algorisme del Gradient descendent.

4.4 Resultats

Després de dur a terme múltiples proves per diferents valors de N, podem assegurar que l'ordinador aprèn i que fins i tot, en certs casos, troba grafs que refuten la Conjectura 1.

Hem fet proves amb $N = 15, 18, 19$ i 21, en aquests casos hem pogut veure que l'ordinador aprèn fàcilment en cada repetició, però no arriba a trobar un contraexemple. Va maximitzant la recompensa $Sz_e(G_N)$ fins al punt que troba grafs amb $Sz_e(G_N) = Sz_e(K_N)$, però no n'aconsegueix trobar cap que superi aquest valor.

En canvi, per valors de N parells i superiors a 20 en executar el codi l'ordinador va millorant la màxima recompensa de cada repetició, fins al moment en què troba un graf tal que $Sz_e(G_N) > Sz_e(K_N)$. Concretament, podem fixar-nos en el cas de $N = 20$ i $N = 22$,

$$Sz_e(G_{20}) = 68690 > 68590 = \frac{1}{2} \cdot 20 \cdot (20-1)^3 = Sz_e(K_{20}).$$

$$Sz_e(G_{22}) = 108349 > 101871 = \frac{1}{2} \cdot 22 \cdot (22-1)^3 = Sz_e(K_{22}).$$

Els grafs trobats amb aquests edge-Szeged índexs són:

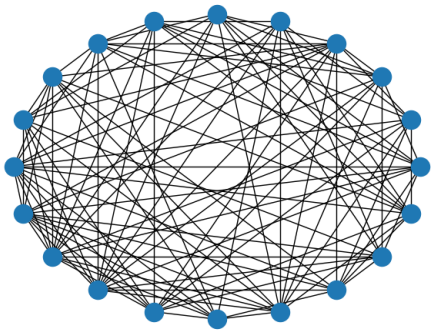


Figura 11: Contraexemple per $N = 20$.

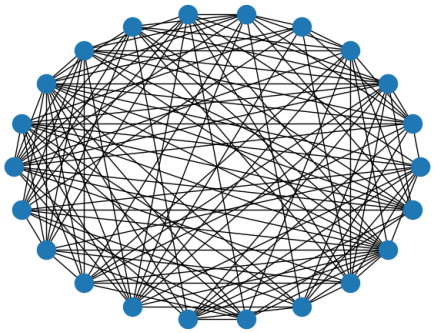


Figura 12: Contraexemple per $N = 22$.

Un cop comprovat que l'ordinador era capaç de trobar grafs que refutessin aquesta conjectura, vam tractar de trobar patrons entre aquests dos grafs, per veure si hi havia alguna relació entre ells. Per fer-ho vam representar aquests grafs mitjançant la seva **matriu d'adjacència**.

Aquesta matriu està composta de manera que:

- Les columnes i les files representen els vèrtexs del graf.
- Hi haurà un 1 si els dos vèrtexs que corresponen a aquell element estan connectats per una aresta, i un 0 si no. És a dir,

$$A = (a_{ij}) \quad \text{tal que} \quad a_{ij} = \begin{cases} 1 & \text{si el vèrtex } i \text{ i el vèrtex } j \text{ estan connectats,} \\ 0 & \text{si no.} \end{cases}$$

Es poden trobar les matrius d'adjacència dels grafs G_{20} i G_{22} a l'Apèndix A.

Tractar de buscar patrons entre aquestes dues matrius va resultar una tasca bastant costosa i que no ens va donar els resultats que esperàvem. Així que vam pensar que potser era millor idea tractar de modificar el codi de manera que un cop trobés un contraexemple per $N = 20$, utilitzés aquest com a base per trobar un amb $N = 22$ vèrtexs. Així doncs, vam utilitzar un codi que afegia dos vèrtexs sense cap aresta a G_{20} i aleshores provava d'anar afegint arestes amb aquests dos nous vèrtexs i calculant l'edge-Szeged índex del graf resultant.

Amb aquest nou enfocament, tampoc es van poder obtenir els resultats que ens haguessin agradat. El programa no va ser capaç de trobar un contraexemple per $N = 22$, afegint dos vèrtexs i qualsevol combinació d'arestes a G_{20}.

Aleshores, podem dir que hem assolit el nostre objectiu inicial, el d'utilitzar l'aprenentatge per reforç com a mètode per trobar grafs explícits que refutessin la Conjectura 1. Tot i això, segueix havent-hi marge de millora, ja que continuar la investigació ens podria permetre arribar a trobar una família de contraexemples, fent que els resultats fossin molt més rics i complets.

Comparació amb mètodes anteriors:
Sabem que és pràcticament impossible l'estudi de tots els grafs possibles amb un nombre de vèrtexs concret, i més quan aquest nombre comença a ser mínimament elevat. Les possibles combinacions d'arestes que es poden triar en el graf augmenta considerablement, fins a punts que resulta un treball molt costós per una persona. Aleshores, una possible solució per tractar de trobar algun contraexemple seria fer-ho mitjançant l'estudi de les propietats de diferents famílies de grafs.

Un exemple d'aquesta possible solució, és el que trobem en [Vuk09], on s'estudia una família de grafs que compleixen que:

$$Sz_e(G_n) > 3 \cdot \left\lfloor \frac{n-3}{6} \right\rfloor^6.$$

En l'article Vukičević demostra que aquesta família és un contraexemple de la Conjectura 1 quan $n \to \infty$. De fet, el resultat que es demostra és el següent:

$$\lim_{n\to\infty} \log_n \max\{Sz_e(G_n)\} = 6$$

on G_n són els grafs de la família amb n vèrtexs. Sabent que

$$\lim_{n\to\infty} \log_n Sz_e(K_n) = 4,$$

tenim que per un n prou gran, la conjectura no serà certa.

Podem comparar el que suposaria buscar un contraexemple concret intentant utilitzar l'estudi d'aquesta família de grafs en comparació amb el mètode d'aprenentatge per reforç. Per fer-ho

definim la funció següent:

$$f(n) = \frac{1}{2}n(n-1)^3 - 3 \cdot \left\lfloor \frac{n-3}{6} \right\rfloor^6$$

ja que volem trobar que $Sz_e(G_n) > Sz_e(K_n)$, aleshores si veiem que

$$Sz_e(G_n) > 3 \cdot \left\lfloor \frac{n-3}{6} \right\rfloor^6 > Sz_e(K_n) = \frac{1}{2}n(n-1)^3 \Longrightarrow 3 \cdot \left\lfloor \frac{n-3}{6} \right\rfloor^6 > \frac{1}{2}n(n-1)^3 \Longrightarrow f(n) < 0$$

voldrà dir que hem trobat un contraexemple.

Grafiant aquesta funció obtenim el següent:

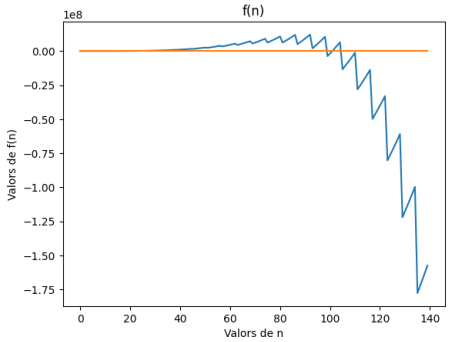


Figura 13: $f(n)$ per $n \in (0, 140)$.

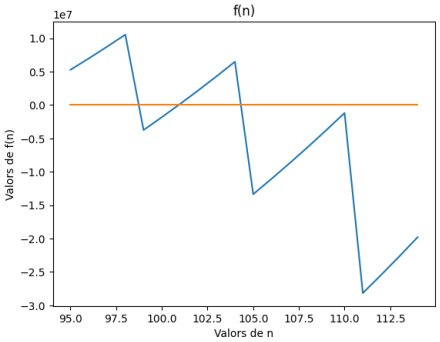


Figura 14: $f(n)$ per $n \in (95, 115)$.

Veiem, que només podem assegurar que $f(n) < 0$ quan aproximadament $n > 100$. Per tant, aquest procediment segurament ens permetria trobar un contraexemple després de realitzar una sèrie de manipulacions més, però d'una manera molt més complicada i costosa, ja que estaríem treballant amb grafs d'ordre $N > 100$.

Com ja hem vist, mitjançant l'aprenentatge per reforç l'ordinador ha sigut capaç de trobar un contraexemple per $N = 20$, millorant aproximadament en 80 vèrtexs l'ordre dels anteriors exemples coneguts.

4.5 Conclusions

Indubtablement, l'aprenentatge per reforç es mostra com una eina amb molt potencial, especialment quan es combina amb les xarxes neuronals. El nostre estudi en aquest àmbit ens ha permès veure com entenent les bases matemàtiques i aplicant-les a una situació específica hem pogut obtenir resultats satisfactoris, aparentment massa complicats de trobar sense l'ús d'aquestes eines.

Els contraexemples trobats, per molt que han sigut innovadors i un avenç en l'estudi de les característiques d'aquests grafs, són simplement l'inici d'una recerca que podria anar més enllà. Hem intentat sense èxit trobar patrons o relacions entre els diferents grafs que refutaven la Conjectura 1. Però, és fàcil pensar que fent un estudi més extens, per exemple, analitzant les propietats de les matrius d'adjacència, podríem arribar a definir una família de grafs que siguin contraexemples de la conjectura. Això, donaria un valor afegit als resultats trobats, aportant més informació a l'estudi de l'edge-Szeged índex.

Hem pogut estudiar i aprofundir en una de les aplicacions de l'aprenentatge per reforç, que és la possibilitat de donar resposta en aspectes de les matemàtiques pures i més concretament

en la teoria de grafs. Es poden extrapolar els resultats trobats a altres aplicacions de l'aprenentatge per reforç, de manera que utilitzant les mateixes bases i principis que hem explorat, es permet generar resultats molt prometedors en altres àrees. Algunes d'aquestes aplicacions amb perspectives notables per al futur inclouen la intel·ligència artificial, la robòtica i diverses branques de la salut.

Apèndix A

Matrius d'adjacència

Pel graf G_{20} presentat en la secció 4.4 que refutava la Conjectura 1 amb $N = 20$, tenim la matriu d'adjacència A_{20}:

$$A_{20} = \begin{pmatrix}
0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0
\end{pmatrix}$$

En el cas del graf G_{22} per $N = 22$, pot ser representat per la matriu d'adjacència:

$$A_{22} = \begin{pmatrix}
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

Bibliografia

[Bah21] Pragati Baheti. *The Essential Guide to Neural Network Architectures*. 2021. URL: https://www.v7labs.com/blog/neural-network-architectures-guide.

[Bel57] Richard Bellman. "A Markovian Decision Process". A: *Indiana University Mathematics Journal* 6 (1957), pàg. 679 - 684. ISSN: 0022-2518.

[Cat11] Universitat Politècnica de Catalunya. *Teoria de Grafs - Notes de Classe*. Set. de 2011.

[GA08] Ivan Gutman i Ali Ashrafi. "The Edge Version of the Szeged Index". A: *Croatica Chemica Acta* 81.2 (2008). ISSN: 0011-1643.

[GBC16] Ian Goodfellow, Yoshua Bengio i Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[Han18] Xintian Han. "A Mathematical Introduction to Reinforcement Learning". A: 2018. URL: https://api.semanticscholar.org/CorpusID:51988118.

[Les+93] Moshe Leshno et al. "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function". A: *Neural Networks* 6.6 (1993), pàg. 861 - 867. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(05)80131-5. URL: https://www.sciencedirect.com/science/article/pii/S0893608005801315.

[Min61] Marvin Minsky. "Steps toward Artificial Intelligence". A: *Proceedings of the IRE* 49.1 (1961), pàg. 8 - 30. DOI: 10.1109/JRPROC.1961.287775.

[Noo17] Alireza Nooraiepour. *Schematic of a DNN with 3 hidden layers*. ResearchGate. 2017. URL: https://www.researchgate.net/figure/Schematic-of-a-DNN-with-3-hidden-layers_fig3_317054093.

[Pun] Alex Punnen. *The Maths behind Neural Networks*. URL: http://alexcpn.github.io/html/NN/ml/8_backpropogation_full/.

[RA18] Richard S. Sutton i Andrew G. Barto. *Reinforcement Learning: An Introduction (2nd ed.)* MIT Press, 2018.

[RA98] Richard S. Sutton i Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[SMR22] Sydney Mathematical Research Institute (SMRI). *Machine Learning for the Working Mathematician*. 2022. URL: https://sites.google.com/view/mlwm-seminar-2022.

[Vuk09] Damir Vukičević. "Note on the graphs with the greatest edge-Szeged index". A: *Communications in Mathematical and in Computer Chemistry* (2009). ISSN: 0340 - 6253. URL: https://match.pmf.kg.ac.rs/electronic_versions/Match61/n3/match61n3_673-681.pdf.

[Wag] Adam Zsolt Wagner. *zawagner22/cross-entropy-for-combinatorics*. URL: https://github.com/zawagner22/cross-entropy-for-combinatorics.

[Wag21] Adam Zsolt Wagner. "Constructions in combinatorics via neural networks". A: (2021). DOI: 10.48550/ARXIV.2104.14516. URL: https://arxiv.org/abs/2104.14516.

[Win21] Wintempla. *SoftMax*. 2021. URL: http://sintesis.ugto.mx/WintemplaWeb/01Neural%20Lab/03Classification/04SoftMax/index.htm.

[WL12] Shouzhong Wang i Bolian Liu. "A Method of Calculating the Edge-Szeged Index of Hexagonal Chain". A: *MATCH - Communications in Mathematical and in Computer Chemistry* 68 (gen. de 2012).

[Zha24] S. Zhao. *Mathematical Foundations of Reinforcement Learning*. Springer Nature Press i Tsinghua University Press, 2024.