

Universitatea Politehnica Bucureşti

Facultatea de Automatică şi Calculatoare

Departamentul de Automatică şi Ingineria Sistemelor

**LUCRARE DE LICENŢĂ**

**Predicția structurii proteinelor folosind**

**rețele neurale recurente**

Absolvent

Iulian Cristian Ghiță

Coordonator

Prof. Dr. Ing. Nicolae Constantin

București, 2018

**Cuprins**

**Abstract**

Tehnologia din ultimele decenii a permis descoperirea formulelor chimice a sute de mii de proteine, dar numai a unei fracțiuni de structuri tridimensionale ale acestor proteine. Aceste structuri sunt esențiale în înțelegerea de mecanisme biologice ca virusurile și anticorpii care îi combat și maladii cauzate de mutații ale unor proteine ca hemoglobina. Pentru a lărgi volumul de structuri dimensionale plecând de la masa de formule chimice deja existente am propus realizarea unor rețele neurale să aproximeze structuri necunoscute folosind numai formule de proteine. Pentru acesata, am utilizat o rețea cu o fereastră alunecătoare variabilă cu o suprapunere, de asemenea variabilă.

**Capitolul 1 - Introducere**

**1.1 Obiectivul propus**

Proteinele sunt substanțele chimice cele mai comune din organism și sursa majorității bolilor și a tratamentelor împotriva lor. Structura lor unică le predispune analizei computaționale. Acestea sunt compuse din lanțuri discrete de subunități chimice numite aminoacizi. Acești aminoacizi sunt într-un număr limitat, în jur de douăzeci în funcție de domeniul în care sunt analizați, ceea ce permite o codificare simplă a lor în alfabetul latin. Această codificare, deși cea mai comună, nu cuprinde diferențele chimice dintre acești aminoacizi, pentru aceasta există alte codificări care transformă fiecare aminoacid într-un vector de numere care reprezintă valorile proprietăților fizio-chimice care îl caracterizează.

Relația dintre formula chimică a unei proteine și proprietățile sale nu este evidentă, fiind posibile mai multe proteine cu aceeași funcție în organism dar cu formule chimice diferite. Cazul a două proteine cu aceeași formulă dar cu funcții diferite nu există, ceea ce sugerează o relație cauzală de la structura chimică la proprietăți. Funcția proteinei este determinată cel mai mult de structura sa tridimensională. Lanțurile de aminoacizi sunt flexibile, nu stau drepte ci se aranjează în conformații stabile din punct de vedere electro-chimic, stabilitatea fiind realizată de legături slabe de hidrogen cu alte părți ale lanțului proteic. Deși pe parcursul interacției lor cu alte substanțe chimice structura lor tridimensională se poate schimba, toate proteinele au o conformație de bază, stabilă, care poate fi reprezentată prin coordonate carteziene.

Procesul de identificare a conformației unei proteine folosește metode experimentale, precum cristalografia, care poate fi costisitor și îndelungar, în special când sunt considerate miile de proteine care sunt descoperite în fiecare an. Formulele lor chimice pot fi descoperite mult mai ușor ceea ce rezultă în mari baze de date cu proteine descrise chimic și baze de date mici, care conțin doar un subset al acestor proteine cu descrierea tridimensională.

Rețelele neurale au potențialul de a aproxima funcții și mapări neliniare, plecănd de la exemple de corespondențe cunoscute, fără a cunoaște în prealabil relațiile dintre acestea. Folosind datele din exemplele date, rețeaua poate apoi primi un set de date nou și să genereze un set de date care să aproximeze corespondența reala a acestuia. În cazul proteinelor cele două seturi de date sunt formulele chimice și structurile tridimensionale corespondente acestora. Folosind date din domeniul virologiei și ale structurii hemoglobinei, două cazuri în care structurile proteice au un rol important, am dezvoltat rețele neurale care să predicteze structura folosind numai formula chimică.

**1.2 Structura documentului**

În primul capitol am descris starea cercetării actuale în predicția structurii proteinelor folosind metode statistice și rețele neurale.

Al doilea capitol prezintă câteva noțiuni teoretice despre compoziția și proprietățile proteinelor și despre rețele neurale. Se continuă cu descrierea aminoacizilor, categorizarea acestora, modul în care aceștia formează legături între ei și se aranjează în forme tridimensionale. Urmează noțiuni legate de cele două domenii din care se folosesc date: virusurile și hemoglobina. Sunt discutate mecanismele acestora și rolul structurii proteinelor în funcționarea lor. Ultima parte a capitolului intră în teoria rețelelor neurale. Sunt discutate arhitecturile principale, funcții de activare, preprocesarea datelor și algoritmi de antrenare.

Al treailea capitol discută arhitectura sistemului implementat. Începe cu formatele fișierelor PDB și FASTA, în care datele sunt ținute.

**1.3 Starea tehnologiei**

În ultimii ani, medicina moleculară și biotehnologia au fost transformate prin descoperiri din studiile genomului uman. Înțelegerea mecanismelor bolii la nivel celular a accelerat descoperirea și dezvoltarea de noi agenți terapeutici. În ultimii ani, s-a înregistrat o creștere substanțială a numărului de peptide și proteine terapeutice care ajung pe piață. Aproximativ 20 de miliarde de dolari sunt cheltuiți anual pentru cercetare și dezvoltare biofarmaceutică [1]. O mare parte din aceste cheltuieli au dus la aproximativ 2500 de medicamente pe bază de biotehnologie aflate în prezent în curs de dezvoltare, 900 în studii preclinice și peste 1600 de studii clinice [2].

Dificultatea extremă a predicției proprietăților proteinelor și a efectelor lor clinice plecând de la structura lor prezintă o provocare pentru dezvoltarea de noi proteine importante din punct de vedere farmaceutic. Ca rezultat, dezvoltatorii de medicamente pe bază de proteine se orientează către algoritmi computaționali care asociază secvențele de aminoacizi cu proprietățile dorite. În acest sens, tehnicile de învățare a rețelelor neurale joacă un rol tot mai important [3][4].

Studiul relației cantitative structură-funcție (Quantitative structure–activity relationship – QSAR) caută să asocieze structura chimică a proteinelor cu funcția lor, cu asumpția că există corelație între acestea. QSAR a atras interes științific, în special în industria farmaceutică pentru descoperirea de droguri. În plus față de avansarea cunoștințelor fundamentale despre QSAR, cercetările au încurajat aplicarea lor într-o gamă mai largă de discipline, inclusiv analize biologice și chimice de rutină. QSAR s-a maturizat semnificativ în ultimele decenii, pentru a include descriptori, modele și algoritmi de inteligență artificială. Un avantaj al rețelelor neurale față de metodele pe bază de regresie statistică folosite anterior este abilitatea lor de a modela relații neliniare între structuri chimice și proprietăți fizio-chimice[5][6].

De exemplu, în [7], QSAR pentru antigenul proteinei non-structurale NS3 a virusului hepatita C a fost modelată folosind rețele neurale. Separat s-a arătat că se pot proiecta teste mai eficiente pentru identificarea de anticorpi împotriva hepatitei C dacă se cunosc funcțiile proteinelor, inclusiv a NS3, de la mai multe tulpini de virus [8]. Rezultatele au găsit asocierile dintre proprietățile antigenice și structura proteinei NS3. [9] a studiat metabolizarea medicamentelor de către izomorfismse ale cytocromei CYP450 (o clasă de enzime) Folosind o rețea neurala a clasificat un set de 580 de substanțe chimice metabolizate de către șapte izomorfisme ale CYP450, folosind diferite codificări ale structurii substanțelor care descriau structura acestora.

Din perspectiva teoriei inteligenței artificiale, predicția structurii secundare a proteinelor poate fi privită ca o problemă de clasificare, în care apartenența fiecărui aminoacid la una din trei sau patru clase este determinată de trăsăturile secveței. Primele încercări de predicție a structurii secundare au folosit analiză statistică [10][11][12], iar [13]apoi metode de clustering[14][15]. Aceste metode a atins o rată de succes de 60%. Metode folosind rețele neurale au atins rate de succes de peste 70% [16][17][18]. Când se studiază predicția structurii proteinelor este importantă alegerea corectă a schemei de codificare a aminoacizilor. O codificare îmbunătățită realizată cu algoritmul PSIPRED [19] a crescut acuratețea predicției până la 78.3%. O altă abordare este aplicarea rețelelor neuronale recurente, în care legăturile dintre neuroni formează un cicluri [20].

Structura terțiara (structura 3D a unei proteine) s-a predictat în literatură prin organizarea proteinelor în clase în funcție de structură și prin punerea unei noi proteine într-o clasă deja existentă pornind de la secvența ei chimică și din înrudirea evoluționară a acesteia cu alte proteine. Baza de date SCOP conține proteine organizate în familii, subfamilii, folduri și clase pe baza structurii 3D [21]. Apartenența unei proteine la clase presupune compararea cu alte proteine folosind alinieri, o metodă cu o clasă de complexitate mare și ineficientă cănd dimensiunea datelor crește, ceea ce a dus la cercetări pentru metode care nu implică alinieri [22][23].

**Capitolul 2 - Aspecte Teoretice ale Biologiei**

**2.1 Proteine**

**2.1.1 Ce sunt proteinele?**

Proteinele sunt polimeri (lanțuri) lungi de aminoacizi, care constituie cea mai mare parte (în afară de apă) a celulelor, și natural, a organismelor multicelulare. Unele proteine au activitate catalitică (permit sau inhibă realizarea reacțiilor chimice) și funcționează ca enzime; altele servesc drept elemente structurale, receptori de semnal sau transportatori care transportă sau elimină substanțe specifice din celule. Proteinele sunt probabil cele mai versatile dintre toate biomoleculele. Bacteria Escherichia coli este formată din mai mult de 3.000 de tipuri de proteine diferite, iar un om dintr-un număr de la 25.000 până la 35.000 de tipuri de proteine[24] (pg 96).

Proteinele, apar într-o mare varietate; mii de diferite tipuri, variind de la peptide relativ mici (formate dintr-un număr mai mic de 100 de aminoacizi) până la polimeri mari cu greutăți moleculare de ordinul milioanelor (în conformitate cu [25]), pot fi găsite într-o singură celulă. Mai mult decât atât, proteinele prezintă o enormă diversitate a funcției biologice și sunt cele mai importante produse organismului, acestea fiind expresia directa a codului genetic. Expresia codului genetic se refera la translația codului din ADN (acid dezoxiribonucleic) în ARN (acid ribonucleic) și apoi transcrierea în lanțurile de aminoacizi corespunzători.

Toate proteinele, de la cele mai vechi linii de bacterii până la cele mai complexe forme ale vieții, sunt construite din același set omniprezent de 20 de aminoacizi, legați prin legături covalente în secvențe liniare caracteristice. Deoarece fiecare dintre acești aminoacizi are o catenă laterală cu proprietăți chimice distincte, acest grup de 20 de molecule poate fi considerat ca fiind alfabetul în care limbajul structurii proteice este scris[24](pg 104-106).

Ceea ce este cel mai remarcabil este faptul că celulele pot produce proteine ​​cu proprietăți și funcții complet diferite prin îmbinarea acelorași 20 de aminoacizi în mai multe combinații de secvențe. Construite din aceste blocuri constructive, organismele pot crea produse atât de diverse precum enzime, hormoni, anticorpi, transportatori, fibre musculare, proteina lentilelor din ochi, pene, pânze de păianjen, cornuri de rinocer, antibiotice, otrăvuri și numeroase alte substanțe având activități biologice distincte.

Dimensiunea relativ mare a proteinelor reflectă funcțiile lor. Funcția unei enzime, de exemplu, necesită o structură stabilă care conține o zonă de contact suficient de mare pentru a-și lega substratul și a cataliza o reacție. Dimensiunea proteinei are însă limite impuse de doi factori: capacitatea genetică de codificare a acizilor nucleici și precizia procesului de biosinteză a proteinelor. Folosirea mai multor copii ale uneia sau a câtorva proteine ​​pentru a face o structură mare de închidere (capsid) este importantă pentru viruși deoarece această strategie conservă materialul genetic. Conform teoriei expresiei genelor, există o corespondență liniară între secvența unei gene în acidul nucleic și secvența de aminoacizi a proteinei pentru care codifică. Acizii nucleici ai virusurilor sunt mult prea mici pentru a codifica informațiile necesare pentru o capsulă de proteine ​​fabricată dintr-o singură polipeptidă. Prin utilizarea mai multor copii de polipeptide mai mici, este nevoie de un acid nucleic mai scurt pentru codificarea subunităților capsidice și acest acid nucleic poate fi folosit în mod eficient și repetat. Celulele utilizează de asemenea complexe de polipeptide în mușchi, cilia, citoschelet și alte structuri. Este mai eficient să se facă mai multe copii ale unei mici polipeptide decât o singură copie a unei proteine ​​foarte mari. Majoritatea proteinelor cu o greutate moleculară mai mare de 100.000 au subunități multiple, identice sau diferite. Al doilea factor care limitează dimensiunea proteinelor este frecvența erorilor în timpul biosintezei proteinelor. Frecvența de eroare este scăzută (aproximativ o greșeală la 10000 de reziduuri de aminoacizi adăugate), dar chiar și această rată scăzută are ca rezultat o probabilitate mare de proteine ​​deteriorate dacă proteina este foarte mare. Probabilitatea de a încorpora un aminoacid greșit într-o proteină mare este mai înaltă decât pentru o proteină mică.

**2.1.2 Aminoacizi**

Proteinele sunt polimeri de aminoacizi, fiecare reziduu acid fiind legat de vecinul său printr-o legătură covalentă. (Termenul "reziduu" reflectă pierderea elementelor de apă atunci când un aminoacid este legat de un altul.) Proteinele pot fi defalcate (hidrolizate, separate) în aminoacizii lor constituenți printr-o varietate de metode, și primele studii ale proteinelor s-au concentrat asupra aminoacizilor liberi derivați din acestea. Douăzeci de aminoacizii diferiți se regăsesc frecvent în proteine. Primul care a fost descoperit a fost asparagina, în 1806. Ultimul din cei douăzeci rămași, treonina, nu a fost identificat până în 1938. Toți aminoacizii au nume triviale sau comune, în unele cazuri derivate din sursa din care acestea au fost izolate pentru prima dată. Asparagina a fost întâi descoperită în sparanghel, și glutamatul în glutenul de grâu; tirozina a fost mai întâi izolată din brânză (denumirea sa derivă din grecescul tiros, "brânză"); și glicina (glykos greacă, "Dulce") a fost numit astfel din cauza gustului său dulce[24](pg 75).

Toți 20 dintre aminoacizii obișnuiți au o grupare carboxil și o grupare amidică (numită și grupare amino) legate la același atom de carbon (numit carbonul α) (Figura 2.1). Ei diferă unii de ceilalți prin lanțurile lor laterale sau grupuri R, care variază în structură, mărime și sarcină electrică. În plus față de acești 20 de aminoacizi există altii mai puțin frecventi. Unii sunt reziduuri modificate după ce proteina a fost sintetizată; altii sunt aminoacizi prezenți în organisme vii, dar nu ca și constituenți ai proteinelor. Aminoacizii au toți o structura de bază similara (cu excepția prolinei). Radicalul R atașat atomului de carbon este diferit pentu fiecare aminoacid (Figura 2.1).

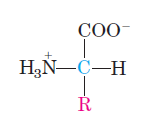


Figura 2.1 Atomul în albastru este cunoscut sub denumirea de carbon α, R este radicalul ([24] pg 76)

Cunoașterea proprietăților chimice ale aminoacizilor comuni este esențială pentru înțelegerea biochimiei. Subiectul poate fi simplificat prin gruparea aminoacizilor în cinci clase principale bazate pe proprietățile radicalilor lor R (în special polaritatea acestora, sau tendința de a interacționa cu apa la pH (aciditate) biologic (în jurul valorii pH 7,0). Polaritatea grupurilor R variază foarte mult, de la nepolar și hidrofob (insolubil în apă) până la foarte mult polar și hidrofil (solubil în apă). Structurile celor 20 de aminoacizi comuni sunt prezentate în Anexa A, în Figura A.1. Aceste grupuri sunt:

* Grupul R non-aromatic, alifatic: alanină, valină, leucină și izoleucină; Grupurile R din această clasă de aminoacizi sunt nepolare și hidrofobe;
* Grupul R aromatic: Fenilalanina, tirozina și triptofanul, cu lanțurile lor aromatice laterale, sunt relativ nepolare (hidrofobe);
* Grupul R polar fără sarcină: serină, treonină, cisteină, asparagină și glutamină; Grupurile R ale acestor aminoacizi sunt mai solubile în apă sau mai hidrofile decât cele ale aminoacizilor nepolari, deoarece conțin grupări funcționale care formează legături de hidrogen cu apa;
* Grupul R cu sarcină pozitivă: lizina, arginina, histidina; Acestea au grupurile R cele mai hidrofile și au cea mai mare sarcină electrică la pH 7,0;
* Grupul R cu sarcină negativă: aspartat, glutamat; au sarcina electrică negativă la pH 7,0.

Două molecule de aminoacizi pot fi legate în mod covalent printr-o legătură amidică substituită, denumită legătură peptidică pentru a se obține o dipeptidă. O astfel de legătură este formată prin eliminarea elementelor de apă (deshidratare) de la gruparea α-carboxil a unui aminoacid și grupul α-amino la altuia. Formarea legăturii peptidice este un exemplu de reacție de condensare, clasă comună de reacții în celulele vii. Sub condiții biochimie standard, echilibrul pentru reacția prezentată în Figura 2.2 favorizează aminoacizii față de dipeptidă. Pentru a face reacția termodinamică mai favorabilă, gruparea carboxil trebuie să fie modificată chimic astfel încât gruparea hidroxil să poate fi eliminată. Într-o peptidă sau o proteină, capătul liber care conține gruparea NH3+ se numește terminusul amino, iar omologul său în ADN-ul care a generat proteina se numește terminusul 5’, și capătul care conține gruparea COO-, terminusul carboxil, iar omologul în ADN terminusul 3’.

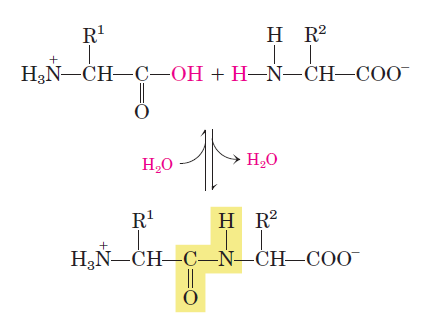


Figura 2.2 Sus, doi aminoacizi individuali, jos, aceeași doi aminoacizi legați într-o dipeptidă; un numar nelimitat de aminoacizi pot fi legați astfel. ([24] pg 85)

**2.1.3 Structura tridimensională**

Aranjamentul spațial al atomilor dintr-o proteină este cunoscut ca și conformația sa. Posibilele conformații ale unei proteine includ orice stare structurală care poate fi realizată fără a rupe legăturile covalente. O schimbare în conformație ar putea să apară, de exemplu, prin rotația în jurul oricărei legături peptidice. Dintre numeroasele conformații care sunt teoretic posibile într-o proteină, una sau mai multe în general predomină în condiții biologice. Conformațiile existente într-un anumit set de condiții sunt de obicei cele care sunt termodinamic cele mai stabile, având cea mai mică energie liberă Gibbs (G) ([24] pg 117). Numai legăturile peptidice se pot roti: legaturile Cα-C și Cα-N. Legăturile C-N (cănd C nu este carbonul α) nu se pot roti. Astfel, fiecarărui aminoacid i se poate asocia un plan în care carbonul α, gruparea amidică și gruparea carboxil sunt conținute (Figura 2.3).

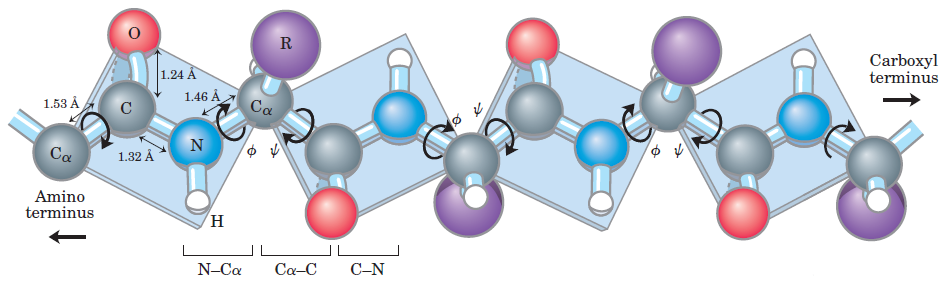


Figura 2.3 O peptidă, unde atomii de carbon sunt gri, cei de azot albaștri, cei de oxigen roșii, cei de hidrogen albi, iar radicalii mov; planurile reprezentate de pătrate vor conține elementele componente ale coloanei vertebrale ale unui aminoacid: carbonul α, gruparea amidică ș cea carboxil; doi aminoacizi legați se vor roti unul față de altul prin legăturile peptidice dintre ei. ([24] pg 119)

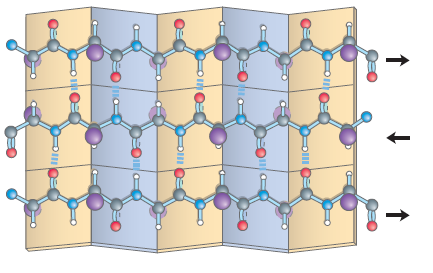
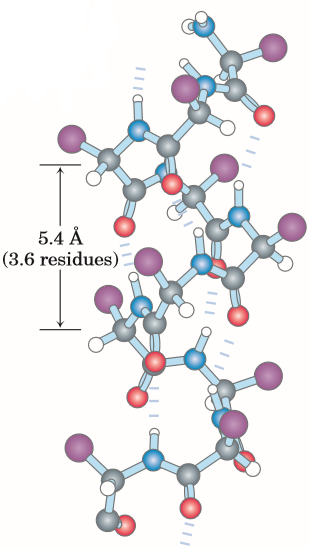
Cunoașterea secvenței aminoacizilor dintr-o proteină poate oferi perspective asupra structurii sale tridimensionale și asupra funcției sale, localizării celulare și evoluției. Cele mai multe dintre aceste vederi sunt derivate prin căutarea asemănărilor cu alte secvențe cunoscute. Mii de secvențe sunt cunoscute și disponibile în baze de date accesibile prin Internet. Modul exact în care secvența de aminoacizi determină structura tridimensională nu este înțeles în detaliu, nici nu putem prezice întotdeauna funcția plecând de la secvență. Cu toate acestea, familiile de proteine ​ care au unele caracteristici structurale sau funcționale partajate pot fi ușor identificate pe baza similitudinilor de secvențe de aminoacizi. Proteinele individuale sunt atribuite familiilor pe baza gradului de similitudine în secvența de aminoacizi. Membrii unei familii sunt de obicei identici în 25% sau mai mult din secvențele lor, iar proteinele din aceste familii au în general cel puțin unele caracteristici comune structurale și funcționale. Unele familii sunt definite, totuși, prin identități care implică doar câteva reziduuri de aminoacizi care sunt critice pentru o anumită funcție.

Fiecare tip de proteine are o secvența unică de aminoacizi, numită structură primară, care este formula sa chimică. Fiind elementul constructiv de bază al unei proteine, secvența de aminoacizi ar trebui să joace un rol fundamental în determinarea structurii tridimensionale a proteinei, și în cele din urmă a funcției sale. Dar este posibilă o anumită flexibilitate a conformației. Se estimează că de la 20% până la 30% din proteinele umane sunt polimorfe, având secvențe de aminoacizi care variază în populația umană. Multe dintre aceste variații în secvență au un efect redus sau niciun efect asupra funcției proteinei. Mai mult, proteinele care execută o funcție în general similară în specii îndepărtat înrudite pot diferi foarte mult în dimensiune și secvență de aminoacizi. Aceasta sugerează seturi diferite de informații conținute în fiecare nivel al structurii. În timp ce structura primară poate indica filogenia (înrudirea evoluționara a diferite proteine), structurile secundare și terțiare indică funcția proteinei ([26] pg 105). Dar deși secvența de aminoacizi în unele regiuni poate varia considerabil fără sa afecteaze funcția biologică, majoritatea proteinelor conțin regiuni care sunt esențiale pentru funcționare și a căror secvență este, prin urmare, conservată și aceste regiuni conservate ar putea indica structurile superioare. Fracțiunea din secvența globală care este critică variază de la proteină la proteină, complicând sarcina de a relaționa secvența cu structura tridimensională și structura cu funcția.

Termenul de structură secundară se referă la conformația locală a unei părți ale unei polipeptide. Cercetarea structurii secundare se concentrează pe cele mai comune forme de pliere ale coloanei vertebrale ale polipeptidei. Există mai multe tipuri de structuri secundare stabile care se produc pe scară largă în proteine. Cele mai proeminente sunt helixul α (Figura 2.6 a), care prezinta o spirală cu radicalii R în exterior, și conformația β (Figura 2.6 b), care are o formă în zig-zag si care poate exista în grupuri paralele numite foi β.

Aproximativ un sfert din reziduurile de aminoacizi din polipeptide se găsesc în conformații helix α, fracția exactă variind foarte mult de la o proteină la alta. Motivul este că helixul α se formează mai ușor decât multe alte conformații posibile, deoarece utilizează în mod optim legăturiile interne de hidrogen. Structura este stabilizată printr-o legătură de hidrogen între atomul de hidrogen atașat la atomul de azot electronegativ al unei legături peptidice și atomul de oxigen carbonil electronegativ al celui de-al patrulea aminoacid de pe partea amino-terminală a acelei legături peptidice (Figura 2.6 a). În cadrul helixului α fiecare legătură peptidică (cu excepția celor apropiate fiecărui capăt al helixului) participă la astfel de legături de hidrogen. Fiecare rotire succesivă a helixului este menținută cu trei până la patru legături de hidrogen adiacente. Toate legăturile de hidrogen combinate conferă întregii structuri elicoidale o stabilitate considerabilă.

Conformația β este a doua cea mai comună conformație. În proteinele globulare, care au o structură pliată compactă, aproape o treime din reziduurile de aminoacizi sunt în întorsături sau în bucle unde lanțul polipeptidic își inversează direcția. Acestea sunt elementele de legătură care leagă secvețele succesive de helix α sau conformaţie β. Foarte frecvente sunt întorsăturile β care conectează capetele a două segmente adiacente ale unei foi β antiparalele. Structura este o întorsătură de 180o, care conține patru reziduuri de aminoacizi, oxigenul carbonil al primului reziduu formează o legătură de hidrogen cu hidrogenul grupării amino din al patrulea. Grupările peptidice ale celor două resturi centrale nu participă la nici o legătură interdependentă de hidrogen. Întorsăturile β se găsesc adesea în apropierea suprafeței unei proteine, unde grupurile peptidice centrale ale celor două reziduuri de aminoacizii din mijlocul întorsaturii se pot lega prin legături de hidrogen cu apă.



1. (b)

Figura 2.6 În (a), conformație helix α, o spirală in jurul unei axe imaginare; în (b), foi β care traversează in zig-zag înainte și înapoi o foaie îndoită imaginară.

Aranjamentul global tridimensional al tuturor atomilor dintr-o proteină este denumit structură terțiară. În timp ce termenul structură secundară se referă la aranjamentul spațial al rreziduurilor de aminoacizi care sunt adiacenti în structura primară, structura terțiară se referă la aspecte pe scară largă ale secvenței de aminoacizi. Aminoacizi care sunt depărtați în secvență pot interacționa în cadrul unei proteinei pliate. Locația curbelor în lanțul polipeptidic, direcția și unghiul acestora este determinat de numărul și localizarea anumitor reziduuri producătoare de îndoire, cum ar fi Pro, Thr, Ser și Gly. Segmentele care interacționează de pe o polipeptidă sunt ținute în pozițiile lor terțiare caracteristice prin diferite tipuri de legături slabe (și uneori prin legături covalente cum ar fi legături disulfurice) între segmente ([24] pg 127).

Unele proteine conțin două sau mai multe lanțuri polipeptidice separate, care pot fi identice sau diferite. Aranjamentul acestor subunități complexele tridimensionale este numit structură cuaternară.

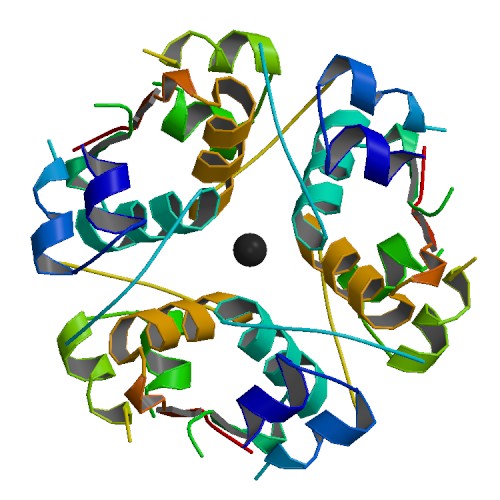


Figura 2.7 Structura terțiară a insulinei [27]

**2.1.4 Funcția proteinelor**

Cunoașterea structurii tridimensionale a unei proteine este o parte importantă a înțelegerii modului în care funcționează proteina. Cele mai multe dintre interacțiunile efectuate de proteine între ele sau cu alte molecule sunt de scurtă durată, dar constituie baza unor procese fiziologice complexe, cum ar fi transportul oxigenului, funcția imunitară și contracția musculară. Funcțiile multor proteine implică legarea reversibilă a altor molecule. O moleculă legată reversibil de o proteină se numește ligand. Un ligand poate fi orice fel de moleculă, inclusiv o altă proteină. Natura tranzitorie a interacțiunilor proteinelor este esențială pentru viață, permițând unui organism să răspundă rapid și reversibil la schimbarea circumstanțelor de mediu și metabolice. Un ligand se leagă la un situs de pe proteină numit situs de legare, care este complementar ligandului în mărime, formă, încărcare, cu caracter hidrofob sau hidrofil. Interacțiunea este specifică: proteina poate să discrimineze între mii de molecule diferite în mediul său și să se lege selectiv doar de una sau mai multe molecule. O proteină dată poate avea situsuri de legare separate pentru mai mulți liganzi diferiți.

**2.2 Virusuri**

**2.2.1 Ciclul de reproducere al virusurilor**

Virusurile sunt agenți care se reproduc prin infectarea celulelor. Acestea constau dintr-o moleculă de ADN sau ARN, o capsulă din proteine care protejează conținutul și proteine pe suprafață al căror rol este identificarea și atașarea de organismul gazdă și injectarea acidului nucleic viral.

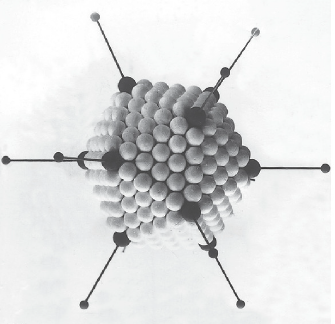


Figura 2.8 Schemă a unui virus; antenele sunt proteinele de identificare si atașare ([28] pg 23)

Ciclurile de reproducere ale virusilor variază între specii, dar constau în câteva etape esențiale comune. Prima etapă este aceea a atașamentului, când virusul se atașează de potențiala celula gazdă. Interacțiunea este specifică fiecarei specii virale și constă în legarea proteinei de atașare a virusului la receptorul țintă de pe suprafața celulei. Contactul inițial dintre un virus și celula gazdă este unul dinamic, reversibil, care implică adesea interacțiuni electrostatice slabe. Cu toate acestea, contactele devin repede mult mai puternice, cu interacțiuni mai stabile, care sunt, în unele cazuri, ireversibile. Faza de atașare determină specificitatea virusului pentru un anumit tip de specie celulară sau gazdă. După atașarea la suprafața celulei, virusul trebuie să efectueze intrarea pentru a se putea replica într-un proces numit penetrare sau intrare. Odată ajuns în interiorul celulei, genomul virusului trebuie să devină disponibil. Acest lucru se realizează prin pierderea a mai multor sau a tuturor proteinelor care alcătuiesc virusul într-un proces numit “uncoating”. Pentru unii viruși, fazele de intrare și “uncoating” sunt combinate într-un singur proces. În mod tipic, aceste prime trei faze nu necesită cheltuieli de energie sub formă de hidroliză ATP ([28] pg 7).

După ce genomul de virus a intrat în celula gazdă, acesta este utilizat în faza de biosinteză când au loc operațiile uzuale ale celulei gazdă: replicarea genomului, transcripția ARNm (ARN mesager) și translația ARNm-ului în proteină. Procesul de translație preluat de virus folosește ribozomii furnizați de celula gazdă și această cerință determină ca virușii să existe numai ca paraziți intracelulari. Genomurile nou-sintetizate pot fi apoi folosite ca șabloane pentru alte runde de replicare și ca șabloane pentru transcripția mai multor molecule mARN de virus într-un proces de amplificare care crește randamentul virusului din celulele infectate. Atunci când genomuri noi sunt produse, ele interacționeaza cu proteinele virale recent sintetizate pentru a forma virusuri descendente într-un proces numit asamblare. În cele din urmă, particulele trebuie să părăsească celula într-o fază de eliberare, după care caută noi celule gazdă pentru a începe din nou procesul. Particulele produse în interiorul celulei pot necesita o prelucrare ulterioară pentru a deveni infecțioase și această fază de maturare poate să apară înainte sau după eliberare.

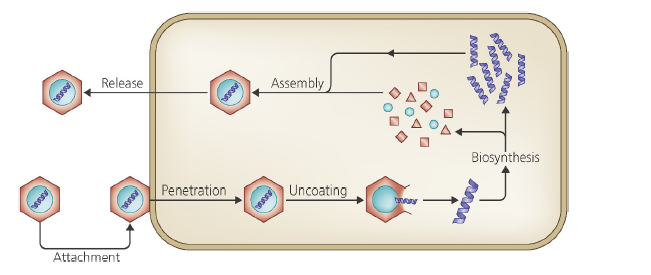


Figura 2.9 Ciclul de reproducere a virusurilor; dreptunghiul bej este interiorul celulei gazdă ([28] pg 7)

**2.2.2 Sistemul de infecție**

Sistemele de infecție ai virusurilor constau de obicei în proteine de membrană pe care se află receptori. Acești receptori identifică prin contact dacă o celula este o gazdă potrivită și în caz afirmativ, se atașează de aceasta pentru ca, pe urmă, proteine specializate din membrană să penetreze peretele celular și să insereze ARN-ul virusului. Anticorpii sunt substanțe care se atașeaza de receptorii virusurilor, blocând astfel abilitatea lor de a se atașa de celule. Dar forma proteinelor din mebrana virusurilor pot bloca anticorpii din a ajunge la receptori, ca în exemplul virusului HIV-1. În acest scop, anticorpii trebuie realizați ținând cont atât de formulele chimice cât și de structurile proteinelor care constituie virusurile. Receptorii virali sunt molecule obișnuite de pe suprafața celulelor care au fost adoptate de viruși pentru a servi ca molecule de atașare și de intrare. Virusurile se leagă de receptorii virali pentru a introduce genomul în gazdă.

Stratul exterior al multor virusuri animale este un înveliș lipidic, în care proteinele de atașare sunt încorporate. Acești viruși livrează materialul genetic și alte conținuturi în celulă prin fuziunea membranelor, fie la suprafața celulară sau după ce virusul a fost luat în celulă într-un vezicul endocist. Plicul lipidic și proteinele încorporate devin parte a membranei celulare. Acest înveliș lipidic complică și mai mult acțiunea anticorpilor.

Atunci când o particulă de virus se află în apropierea unei celule, ceva trebuie să inițieze procesul de infectare. Acest eveniment este legarea proteinelor de atașare de pe suprafața particulei de virus cu componentele suprafeței celulare (receptori). Prin legarea în acest fel este împiedicată difuzia liberă a particulei, crescând astfel șansele ca intrarea virusului să aibe succes. Legarea proteinelor de atașare la receptori declanșează adesea schimbări conformaționale în particulă care sunt necesare pentru intrarea sau eliberarea genomului in celulă. În aceste condiții, atașamentul reprezintă un angajament ireversibil al particulelor virale in interacțiunea lor cu acea celulă; dacă evenimentele de intrare nu reușesc după această modificare sau dacă celula nu este o gazdă viabilă pentru virus, particula nu va putea să se detașeze și să caute o altă celulă gazdă mai favorabilă.

În exemplul virusului HIV-1 (cel mai greu de combătut virus la ora actuală), acesta infectează celulele T CD4+ sau macrofage CD4+. Inițial, virusul folosește o proteină celulară (ciclofilină A) care este încorporată în particula HIV-1 în timpul asamblării pentru a lega un receptor cu afinitate scăzută, numit heparan. Această interacțiune crește șansa de a intra în contact cu receptorul primar, CD4, care este o moleculă mai puțin abundentă. Dacă virusul nu găsește o moleculă primară de receptor, acesta se va disocia complet de celulă și procesul va începe din nou. Procesul de căutare constă în acela că virusul se rotește de-a lungul suprafeței celulare, disocind în mod repetat și reasociind cu receptori de afinitate scăzută, până când vine în contact cu moleculele CD4. Legarea CD4 cu gp120 este o interacțiune cu afinitate mare, dar numai cu această asociere, virusul și straturile bilaterale ale lipidelor celulare sunt prea îndepărtate pentru ca fuziunea să aibă loc. De aceea, legarea la CD4 determină ca gp120 să sufere modificări conformaționale care expun un alt situs care este specific pentru unul dintre co-receptorii, CCR-5 sau CXCR4. Deoarece aceste molecule de co-receptor cuprind 7 regiuni transmembranare cu doar bucle mici pe suprafața extracelulară, legarea la gp120 atrage componentele mai aproape.

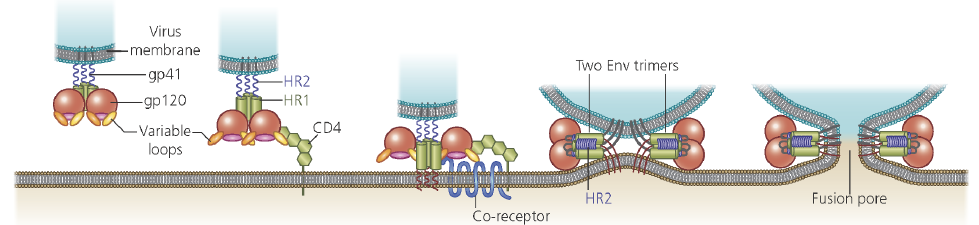


Figura 2.10 Infectarea unei celule T de către virusul HIV-1; complexul de proteine de suprafață gp41 – gp120 al virusului HIV-1 se atașeaza de receptorul CD4 al celulei T; proteina gp120 își schimbă conformația pentru a se lega și de co-receptorii celulei T; după aceasta, virusul este legat de celulă și ARN-ul viral poate fi introdus ([28] pg 76)

**2.2.3 Anticorpi**

Anticorpii sunt proteine care pot circula liber în ser sau pot fi atașate de suprafețele celulelor imunitare. Rolul anticorpilor este fie de a se lega de proteinele de suprafață ale virusurilor, neutralizându-i, fie de a identifica și a se lega de suprafețele celulelor deja infectate. Fiecare tip de anticorp se poate lega numai de câteva proteine virale (numite și antigeni).

Imunoglobulina (IgG) este cel mai comun anticorp. Aceasta conține două regiuni Fa și Fb (Figura 2.11) care conțin regiuni variabile ce se pot lega de un antigen specific. Atunci când imunoglobulina se leagă de o bacterie sau virus invadator, aceasta activează anumite leucocite, cum ar fi macrofagele, pentru a înghiți și distruge invadatorul și, de asemenea, activează alte părți ale răspunsului imunitar. O altă regiune a anticorpului Fc, are rolul de a semnala și de a se lega de receptori de pe suprafața celulelor macrofage, pentru ca acestea din urmă să elimine virusul sau celula infectată printr-un proces numit fagocitoză. În alte cuvinte, regiunea Fc a imunoglobulinei marcheaza agenții patogeni pentru eliminare de către celulele imunitare specializate.

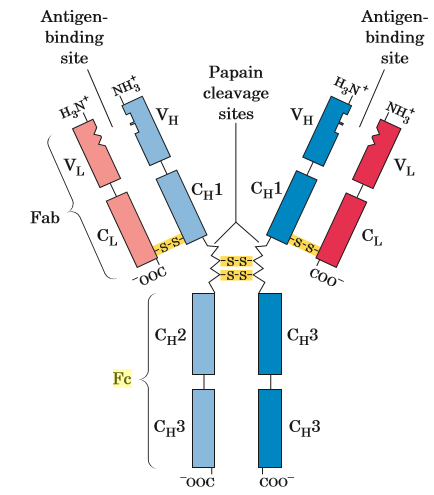


Figura 2.11 Schema imunoglobulinei; secțiunile Fa și Fb de sus a rolul de a se lega de proteinele de suprafața ale virusurilor sau ale celulelor infectate, în timp ce secțiunea Fc de jos are rolul de a semnala și a se lega de o celulă macrofagă pentru a elimina agentul viral ([24] pg 179)

Neutralizarea reprezintă scăderea infectivitatii care se produce atunci când anticorpul se leagă la un antigen de pe particula de virus. Nu toți anticorpii care se leagă la o particulă de virus sunt capabili să-i neutralizeze infecțiozitatea. Neutralizarea este un fenomen specific tulpinii virusului (variație a virusului). Literatura timpurie a presupus că anticorpul de neutralizare acționează exclusiv prin împiedicarea atașării virusului la receptori de pe suprafața celulei. Anticorpii la anumite tulpini neutralizează prin blocarea atașării unui la receptorii celulei sale țintă, dar s-a demonstrat că anticorpii neutralizanți la alte tulpini nu inhibă atașarea virusului. În cazul virusului polio, virusul neutralizat se atașează de celule, este preluat într-o veziculă endocitară, dar nu este capabil să realizeze procesul de ”uncoating”. Într-un al doilea exemplu, un virus de influenza se atașează de celulă, este endocitozat, dar fuziunea membranelor virale și celulare nu are loc. Teoria actuală este că există tot atâtea mecanisme de neutralizare cât sunt procese pe care un virus trebuie să le suporte înainte ca genomul viral să poată intra într-o celulă și să fie exprimat ([28] pg 231). Mecanismul de neutralizare este specific anticorpului, astfel încât un virus poate fi neutralizat într-o varietate de moduri diferite, iar neutralizarea este determinată în mare măsură de epitopul la care se leagă anticorpul.

Deoarece anticorpii identifică virusurile numai prin antigenul său specific, care este o proteină, structura acesteia este esențială pentru întelegerea interacției cu sistemul imunitar deoarece zona Fab a anticorpului se va lega de antigen numai dacă structurile tridimensionale ale acestora se potrivesc.

**2.3 Globulina**

**2.3.1 Structura și funția globulinelor**

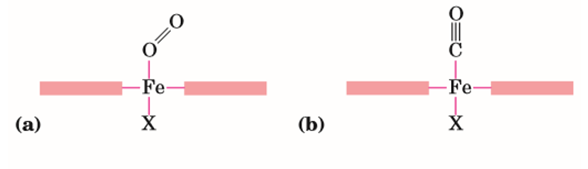
Myoglobina și hemoglobina sunt cele mai studiate și cele mai bine înțelese proteine. Acestea au fost primele proteine ​​pentru care au fost determinate structurile tridimensionale, iar înțelegerea noastră actuală a mioglobinei și a hemoglobinei rezultă din activitatea a mii de biochimiști de-a lungul mai multor decenii.

Oxigenul este slab solubil în soluții apoase și nu poate fi transportat în țesuturi în cantitate suficientă dacă este pur și simplu dizolvat în serul de sânge. Difuzia oxigenului prin țesuturi este de asemenea ineficientă la distanțe mai mari de câțiva milimetri. Evoluția animalelor mari, multicelulare depinde de evoluția proteinelor care ar putea transporta și stoca oxigenul. Organismele multicelulare exploatează proprietățile metalelor, cel mai frecvent fier, pentru transportul de oxigen. Cu toate acestea, fierul liber rezultă în formarea de compuși de oxigen foarte reactivi cum ar fi radicalii hidroxilici care pot deteriora ADN-ul și alte macromolecule. Fierul (mai exact, Fe2+) utilizat în celule este legat în forme care îl izolează și îl fac mai puțin reactiv. În organismele multicelulare - în special cele în care fierul trebuie transportat pe distanțe mari – acesta este adesea încorporat într-un grup protetic legat de proteine ​​numit hemă, iar acesta la randul lui este încorporat în hemoglobină sau myoglobină. Oxigenul trebuie nu numai absorbit și transportat, dar și eliberat, iar globulinele asistă în aceasta acţiune.

În moleculele heme libere (heme care nu sunt legate la proteine), reacția oxigenului la unul dintre cele două legături covalente libere ale fierului (perpendicular pe planul hemei, deasupra și dedesubt) poate avea ca rezultat conversia ireversibilă a Fe2+ la Fe3+. În proteinele care conțin hemă, această reacție este împiedicată prin izolarea hemei adânc în structura proteinei în care accesul la cele două legături de coordonate libere este limitat (Figura 2.13). Una dintre aceste două legături este ocupată de un atom de azot al unui aminoacid histidină din lanțul lateral al proteinei de hlobulină. Celălalt este locul de legare pentru oxigenul molecular (O2).

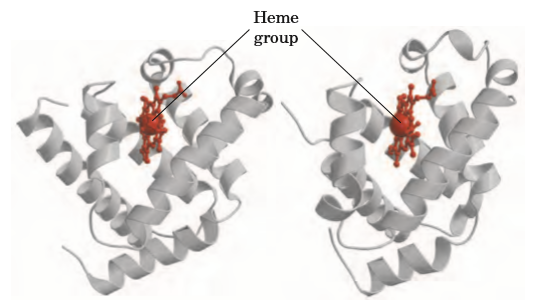
Interacțiunea dintre hemă și liganzii de care aceasta se leagă este puternic afectată de structura proteinelor și este adesea însoțită de modificări conformaționale ale acestora (structura tridimensionala). De exemplu, puterea cu care se leagă hema de liganzi diferiți este modificată atunci când hema este o componentă a mioglobinei. Monoxidul de carbon se leagă de moleculele heme libere de peste 20.000 de ori mai ușor decât O2, dar se leagă numai de aproximativ 200 de ori mai bine atunci când hema este încorporată în mioglobină. Aceasta se datorează faptului ca molecula de oxigen se atașează diagonal pe hemă, fiind stabilizata de un capat al globulinei, în timp ce monoxidul de carbon se atașează perpendicular pe hemă, ceea ce este îngreunat de globulină (Figura 2.12).

În plus față de transportul aproape total al oxigenului necesar celulelor de la plămâni la țesuturi, hemoglobina transportă două produse finale ale respirației celulare: hidrogen H+ și dioxid de carbon CO2, de la țesuturi la plămâni și la rinichi, unde sunt excretate.



**Figura 2.12** Hema este în roșu, molecula de oxigen in portocaliu și secțiuni ale globulinei în gri. Figura (a) vedere paralelă cu hema, unde oxigenul s-a atașat de aceasta; figura (b) același unghi cu o moleculă de monoxid de carbon atașată. ([24] pg 162)

Hemoglobina este aproximativ sferică, cu un diametru de aproape 5,5 nm. Este o proteină tetramerică care conține patru grupări proteice heme, una asociată cu fiecare lanț polipeptidic. Aceasta conține două tipuri de globulină, două lanțuri alfa (141 de reziduuri fiecare) și două lanțuri beta (146 de reziduuri fiecare). Deși mai puțin de jumătate din reziduurile de aminoacizi din secvențele polipeptidice ale subunităților alfa și beta sunt identice, structurile tridimensionale ale celor două tipuri de subunități sunt foarte asemănătoare. Mai mult, structurile lor sunt foarte asemănătoare cu cele ale myoglobinei (Figuta 2.13), chiar dacă secvențele de aminoacizi ale celor trei polipeptide sunt identice în numai 27 de poziții.



**Figura 2.13** În stânga o polipeptidă de myoglobină, în dreapta, o subunitate α a hemoglobinei ([24] pg 163)

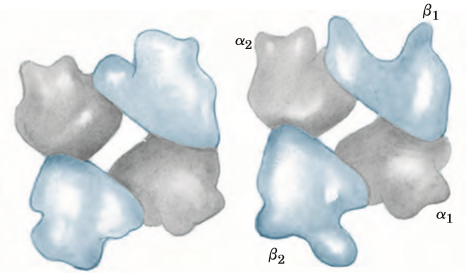
**2.3.2 Siclemia**

Importanța deosebită a secvenței de aminoacizi în determinarea structurilor secundare, terțiare și cuaternare ale proteinelor globulare și, prin urmare, a funcțiilor lor biologice, este demonstrată prin boala ereditara a siclemiei. Aproape 500 de variante genetice ale hemoglobinei sunt cunoscute în populația umană; dar numai câteva sunt rare. Cele mai multe variații constau în diferențe într-un singur reziduu de aminoacid. Efectele asupra structurii și funcției hemoglobinei sunt adesea minore, dar uneori pot fi grave.

Siclemia este o boală genetică în care un individ a moștenit alela pentru hemoglobina siclemica (numită și hemoglobină S) de la ambii părinți. Eritrocitele acestor indivizi sunt mai puține și de asemenea anormale. În plus față de un număr neobișnuit de mare de celule imature, sângele conține multe eritrocite lungi, subțiri, în formă de semilună, care arată ca lama unui secere. Când hemoglobina din celulele secera este deoxigenată, ea devine insolubilă și formează polimeri care se agregă în fibre tubulare. Hemoglobina normală (hemoglobina A) rămâne solubilă la deoxigenare. Fibrele insolubile ale hemoglobinei S deoxigenate sunt responsabile pentru forma de seceră deformată a eritrocitelor, iar proporția celulelor bolnave crește mult pe măsură ce sângele este deoxigenat.

Eritrocitele acestor persoane cu siclemie sunt mai puține și, de asemenea, anormale. În plus față de un număr neobișnuit de mare de celule imature, sângele conține multe eritrocite lungi, subțiri, în formă de semilună, care arată ca lama unei secere. Proprietățile modificate ale hemoglobinei S (siclemie) rezultă dintr-o substituție a unui singur aminoacid, un reziduu Valină în loc de unul Glutamat la poziția 6 în cele două lanțuri beta. Valina nu are încărcătură electrică, în timp ce glutamatul are o încărcătură negativă la pH 7,4. Rezultatul este o ușoară modificare a structurii hemoglobinei.

Globulinele sunt remarcabile prin sensibilitatea structurii lor tridimensionale la mutații chimice în lanțul aminoacidic. Înțelegerea acestei sensibilități poate oferi perspective în înțelegerea maldiilor cauzate de mutația acestora.



**Figura 2.14** În stânga o molecula de hemoblogină normala, în dreapta, o molecula S; subunitățile β sunt schimbate din cauza unui singur aminoacid diferit în fiecare ([24] pg 174)

**Capitolul 3 – Aspecte Teoretice ale Rețelelor neurale**

**3.1 Ce sunt rețelele neurale?**

Rețelele neurale sunt algoritmi statistici care pot aproxima orice funcție liniară sau neliniară folosind o rețea de conexiuni care imită funcționarea creierului. doua proprietate fundamentală a rețelelor neuronale este capacitatea lor de a implementa funcții neliniare, permițând o aproximare uniformă a oricărei funcții continue. Proprietatea lor de a aproxima orice funție continuă este fundamentală în studierea sistemelor biologice și de mediu, care pot prezenta răspunsuri variabile chiar și atunci când intrarea este aceeași. Parametrii acestora conțin un vector de intrare (numit strat) în care datele de antrenare sau de testare sunt introduse, straturi intermediare în care fiecare element (neuron) este conectat la fiecare element de intrare și un strat de ieșire conectat la ultimul strat intermediar. Conexiunile conțin ponderi ale căror valori influențează intensitatea conexiunii dintre doi neuroni.

Rețelele neuronale sunt structuri de mapare neliniare, care se dovedesc a fi capabile de aproximare universală și foarte flexibile la procesele de generare a datelor. Prin urmare, ele oferă o mare diversitate în tipul de aplicații în care rețelele neuronale pot fi utilizate, în special atunci când procesele de generare a datelor care stau la baza lor sunt necunoscute. Analiza datelor biologice și de mediu este inerent complexă, seturile de date conțin adesea neliniarități temporale, spațiale și sezoniere și distribuții non-Gaussian. Abilitatea de a prognoza și de a prezice valori ale datelor secvețiale în timp le conferă o poziție importantă în cadrul sistemelor de suport decizional. Rețelele neuronale oferă un motor de inferențe puternic pentru analiza regresiei, care provine din capacitatea rețelelor neuronale de a mapa relații neliniare ([29] pg 12).

Parametrii rețelelor neurale sunt setați prin antrenare folosind diverși algoritmi, unii fiind potriviți pentru anumite cantități și tipuri de date. Antrenarea se face prin învățare prin exemplu, similar și modelat după sistemele biologice, rezultatul fiind ajustărea conexiunilor sinaptice care există între neuronii individuali ([30] pg 390). Aceasta poate fi imparțită în doua tipuri: supervizată și nesupervizată. Antrenarea supervizată constă în a introduce în rețeaua neurală un set de date la intrare și un alt set de date la ieșire (numit set target sau țintă) și în a modifica ponderile astfel încât datele de ieșire generate de rețea pe baza intrărilor să fie cât mai apropiate de datele de ieșire din setul țintă. În antrenarea nesupervizată sunt introduse numai date de intrare și rețeaua își modifică ponderile astfel încât la ieșire să asocieze datele care au similarități între ele.

O rețea constă din elemente de procesare interconectate numite noduri sau neuroni care lucrează împreună pentru a produce o funcție de ieșire. Producția unei rețele neuronale se bazează pe cooperarea funcțională a neuronilor individuali din cadrul rețelei, unde procesarea informațiilor se efectuează în paralel, mai degrabă decât secvențial. Intrările scalare individuale *p1, p2, p3, ..., pn* sunt fiecare ponderate cu elementele corespunzătoare *w1, w2, w3, ..., wn* ale matricei de ponderi *W* la care se adună bias-urile *b1, b2, b3, ... , bn*. Suma intrărilor ponderate formează intrarea netă care este procesată de o funcție de activare *f*, și produce ieșirea scalară *a*:

|  |  |  |
| --- | --- | --- |
|  |  | (3.1) |

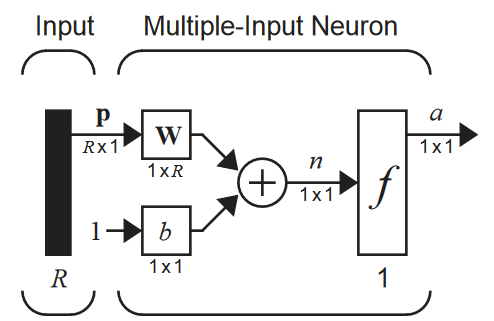


Figura 3.1 Modelul unui neuron; acesta sumează intrările *p* înmulțite cu ponderile *w* și bias-ul *b,* aplică funcția *f* și rezultă ieșirea *a* ([31] pg 43)

Dacă luăm în considerare modelul neuronul biologic, atunci ponderea *w* corespunde rezistenței sinapselor, sumarea reprezintă corpul celular, iar funcția de transfer modelează semnalul axonului. Mai mulți astfel de neuroni plasați în paralel (o serie de neuroni paraleli se numește un strat) formează o rețea neurală. În acest caz, matricea de ponderi este bidimensională *wij* unde *i* este indicele neuronul și *j* este indicele intrării la care acesta este conectat.

|  |  |  |
| --- | --- | --- |
|  |  | (3.2) |

**3.2 Rețeale neurale feedforward**

Cea mai simplă rețea neurală este rețeaua neurală feedward care este reprezentată grafic ca un set de neuroni conectați împreună, în care informațiile se transmit numai în direcția înainte, de la intrări la ieșiri. Într-o reprezentare grafică, în care vârfurile sunt neuronii și marginile sunt conexiunile, graficul unei rețele feedforward este aciclic, urmând nicio cale, urmând conexiunile nu poate duce la punctul de plecare. Neuronii care efectuează calculul final, adică ale căror ieșiri sunt ieșirile rețelei, se numesc neuroni de ieșire; ceilalți neuroni, care efectuează calcule intermediare, sunt numiți neuroni ascunși (Figura 15). Sunt posibile mai multe straturi de neuroni ascunși, fiecare strat fiind conectat la urmatorul.

O rețea feedforward cu *n* intrări, *Nc* neuroni ascunși și *No* neuroni de ieșire calculează funcția neliniară a variabilelor de intrare *n* ca o compoziție ale funcțiilor *Nc*calculate de neuronii ascunși. Trebuie remarcat faptul că rețelele feedforward sunt statice; dacă intrările sunt constante, atunci așa sunt și ieșirile. Timpul necesar pentru calculul funcției fiecărui neuron este, de obicei, neglijabil de mic. Astfel, rețelele neurale feedforward sunt adesea denumite rețele statice, în contrast cu rețelele recurente sau dinamice.

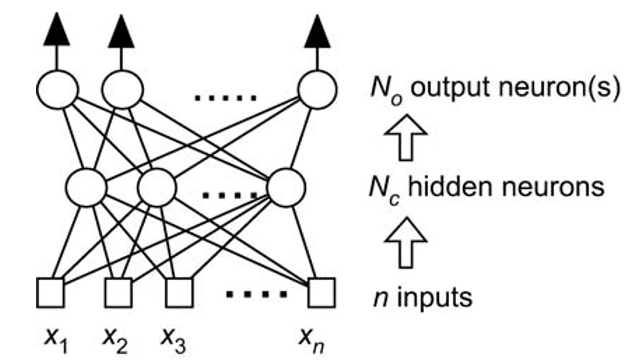


Figura 3.2 O rețea neurală cu *n* intrări, un strat de neuroni ascunși *Nc* și neuroni de ieșire *No* ([32] pg 4)

În stratul ascuns, valoarea de la fiecare neuron de intrare este înmulțită cu o pondere dată, iar valorile ponderate rezultate sunt însumate corespunzător și, opțional cu bias-uri. Suma ponderată este apoi introdusă într-o funcție de activare selectată și rezultatul final este trimis la stratul de ieșire. Determinarea numărului de straturi ascunse este un aspect important pentru modelarea sistemului analizat. Dacă se alege un strat, modelul poate aproxima în mod arbitrar orice funcții care conțin o mapare continuă de la un spațiu al stărilor finit la un altul. Modelele cu două straturi ascunse pot aproxima orice mapare netedă (funcție derivabilă de orice ordin în orice punct al domeniului său) cu precizie arbitrar de mare și cu funcții de activare raționale.

**3.3 Rețeale neurale recurente**

Rețelele neurale recurente sunt rețele ale căror graf de conexiuni prezintă cicluri, în care există cel puțin o cale care pleacă de la un neuron și se întoarce la același neuron. Deoarece ieșirea unui neuron nu poate fi o funcție a proprii sale valori curente, aceasta trebuie întârziată. Fiecare conexiune a unei rețele recurente are atașata o întârziere, care poate fi zero. Pentru ca rețeaua să fie cauzală, fiecare conexiune care formează un ciclu trebuie sa aibă o întârziere mai mare decât zero.

Orice rețea neurală recurentă are o formă canonică, formată dintr-o rețea neurală feedforward, blocuri de întârziere și conexiuni, care are un set de ecuații in formă discretă, unde *x(k)* este vectorul de stare la momentul *k*, *u(k)* este vectorul de intrări externe, *g(k)* este vectorul de ieșiri, iar A,B,C,D sunt matrici. Ecuațiile pentru un sistem liniar sunt:

|  |  |  |
| --- | --- | --- |
|  |  | (3.3) |

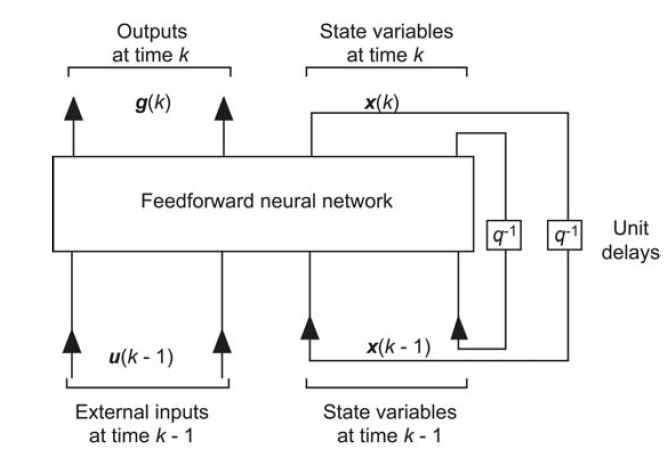


Figura 3.3 Forma canonică a unei rețele neurale recurente ([32] pg 268)

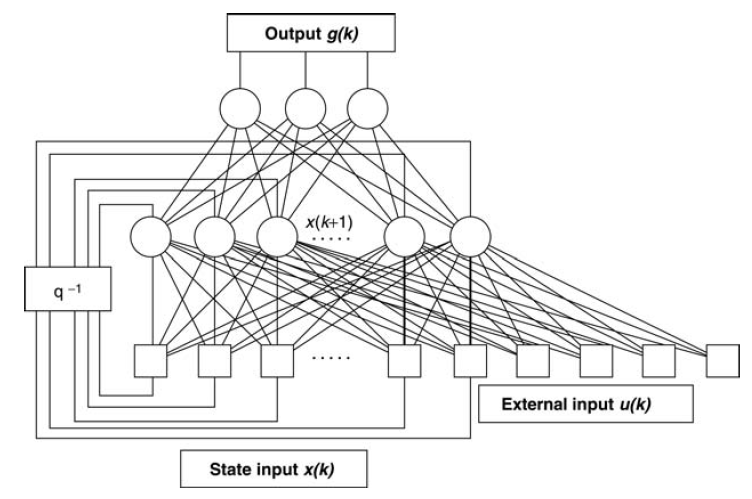


Figura 3.4 Rețea Elman ([32] pg 273)

Principalul tip de rețea recurentă se numește rețea Elman (Figura 3.4). Acesta este o rețea cu un singur strat ascuns, unde fiecare neuron este conectat la întrările externe și ieșirile sunt conectate cu o întârziere de un pas la fiecare neuron din stratul asuns, numiți unități contextuale. Ieșirea este o funcție neliniară care depinde de starea stratului asuns la acel moment și de starea sa la momentul anterior.

**3.4 Funcții de activare**

Funcția de activare este definită în spațiul de intrare N-dimensional, numit și spațiul parametrilor. Majoritatea arhitecturilor de rețea încep prin calcularea sumei ponderate a intrărilor (suma netă totală) și introducerea acesteia ca parametru în funcția de activare pentru a produce o ieșire. Un număr mare de funcții alternative de activare au fost propuse și exploatate în eforturile moderne de cercetare.

Funcțiile de activare cu un interval limitat sunt deseori numite funcții de squashing (Ecuația 3.4). Ieșirea acestei este binară în funcție de faptul dacă intrarea respectă un prag specificat, T. Dacă intrarea netă totală este mai mică de 0, atunci ieșirea neuronului este 0, altfel este 1.

|  |  |  |
| --- | --- | --- |
|  |  | (3.4) |

Funcția pur liniară nu modifică în nici un fel datele primite, astfel, *a* = *n*, unde *n* este suma ponderată a intrărilor.

Funcțiile de activare neliniare pot să ofere noi posibilități de modelare pentru o gamă mai largă de aplicații. Aceste funcții sunt folosite în mod frecvent în rețelele cu backpropagation, în mare parte deoarece este ușor de diferențiat ([31] pg 62). Funcțiile de transfer log-sigmoid primesc intrări și generează ieșiri între 0 și 1, deoarece intrarea netă a neuronului trece de la infinit negativ la infinit pozitiv și este exprimată astfel:

|  |  |  |
| --- | --- | --- |
|  |  | (3.4) |

Tangenta hiperbolică este similară log-sigmoidului dar poate prezenta o dinamică de învățare diferită în timpul fazei de antrenament. Scopul funcției sigmoide este de a genera un grad de neliniaritate între intrarea și ieșirea neuronilor. Modelele care utilizează funcțiile de activare hiperbolice au adesea caracteristici îmbunătățite de învățare generalizate și produc modele cu o precizie îmbunătățită (Ecuația 3.5).

|  |  |  |
| --- | --- | --- |
|  |  | (3.5) |

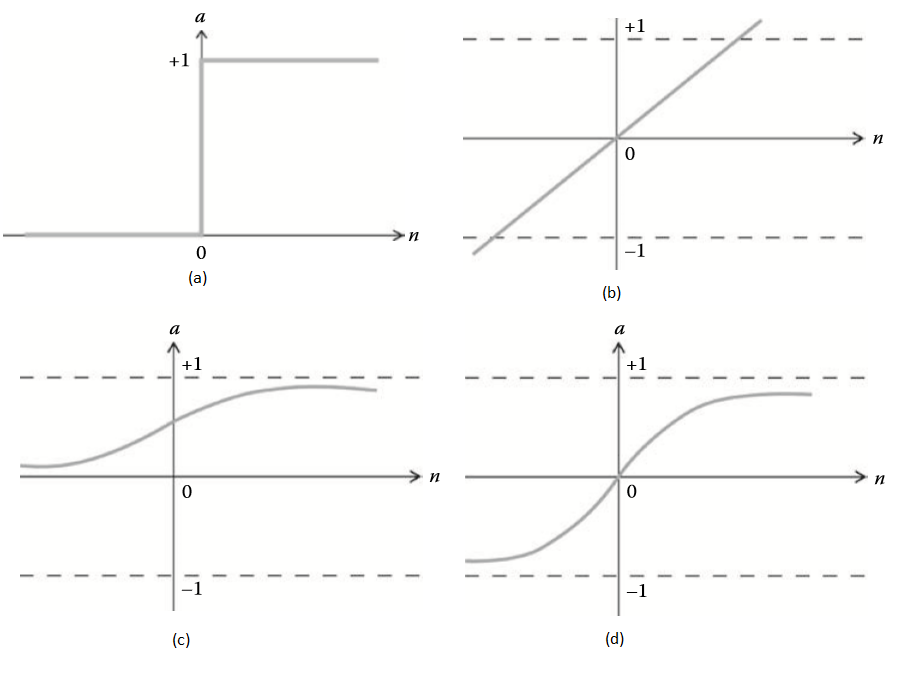


Figura 3.5 (a) – funcție treaptă (squashing); (b) – funcție rampă; (c) - funcție log-sigmoid; (d) – tangentă hiperbolică ([29] pg 9)

**3.5 Preprocesarea datelor**

Înainte de antrenament, variabilele de intrare trebuie să fie normalizate și centrate; dacă intrările au ordine de mărime foarte diferite, cele mai mici nu vor fi luate în considerare în timpul antrenamentului. Acest lucru este deosebit de important în seturile de date biologice și de mediu, în care valori lipsă, redundante, distribuții anormale și date inerent zgomotoase sunt comune și nu excepțiile. Una dintre cele mai frecvente greșeli atunci când se preproceseaza date brute este de a renunța pur și simplu la observațiile privind valorile variabilelor lipsă, chiar dacă este doar o variabilele care lipsește. La analizarea rețelelor neuronale, algoritmii de învățare sunt afectați de datele lipsă, deoarece se bazează în mare măsură pe aceste date pentru a învăța relațiile dintre datele de intrare și ieșire ale sistemelor [18]. Scopul final al preprocesării datelor este de a manipula datele neprocesate într-o formă cu care o rețea neuronală poate fi suficient de bine instruită. Alegerile făcute în acest stadiu de dezvoltare sunt esențiale pentru realizarea și implementarea unei rețele neuronale. Cea mai elementară operație de preprocesare este scalarea datelor, prin formula:

|  |  |  |
| --- | --- | --- |
|  |  | (3.6) |

Unde *zij* este un element din matricea scalată a setului de date, unde coloanele reprezintă seturi diferite de date, iar rândurile dimensiunea unui set dedate, egal cu numărul de intrări a rețelei folosite. Datele trebuie del puțin scalate în intervalul utilizat de neuronii de intrare din rețeaua neuronală. Acesta este în mod caracteristic în intervalul de la -1 la 1 sau de la zero la 1. De asemenea, trebuie stabilit un interval de intrare obligatoriu pentru rețea. Acest mijloc de normalizare va scala datele de intrare într-un interval adecvat, dar nu le va spori uniformitatea.

Problemele de date lipsă și eventualele valori limită sunt inerente cercetării științifice biologice și de mediu. Există numeroase motive pentru care pot lipsi datele, inclusiv disfuncționalitatea instrumentului și raportarea incorectă. Pentru a ști cum se gestionezează datele lipsă, este avantajos să se știe de ce sunt absente. Practicanții consideră de obicei trei mecanisme generale de "lipsă", în care datele lipsă pot fi clasificate după modelul lor: lipsesc complet la întâmplare, lipsesc la întâmplare sau lipsesc în mod regulat. Manipularea necorespunzătoare a valorilor lipsă de date va modifica analiza deoarece, până când se demonstrează altfel, trebuie să se presupună că acele cazuri care lipsesc diferă în mod semnificativ din punct de vedere analitic, de la cazurile în care sunt prezente valori. Există numeroase opțiuni cu care să se preproceseze valorile lipsă. O abordare frecvent utilizată este pur și simplu ignorarea datelor lipsă, ceea ce duce la o eficiență limitată și la prejudecăți în timpul procedurii de modelare. O a doua abordare este imputarea datelor, care variază de la simplă (înlocuind valorile lipsă cu zerouri sau cu media vectorului) cu mecanisme de imputare multiple mai complexe (de exemplu, metoda lanțului Markov Monte Carlo [MCMC]). O abordare alternativă dezvoltată de [33] nu a încercat să reconstruiască valorile lipsă, ci a evaluat mai degrabă impactul lipsei asupra rezultatelor. Aici, autorii au încorporat incertitudinea datorată lipsei într-o funcție adecvată de risc.

Proiectarea modelului poate necesita o reducere a dimensiunii vectorului de intrare *x*. Acest lucru este deosebit de important atunci când numărul variabilelor este prea mare pentru a fi procesat convenabil sau atunci când se presupune că acestea nu sunt mutual independente. În acest din urmă caz, reducerea acestora simplifică proiectarea modelului. Testarea este, așadar, mult mai mare decât respectarea variabilității datelor și este mai puțin sensibilă la suprasolicitare din cauza supra-parametrizării. Având în vedere complexitatea seturilor de date biologice și de mediu, este necesară selectarea adecvată a elementelor de intrare pentru a justifica crearea unor cadre robuste pentru dezvoltarea rețelei neuronale. Scopul definit al selecției caracteristicilor este de a selecta în mod corespunzător un subset de *k* variabile *S*, dintr-un set inițial candidat *C*, care cuprinde setul tuturor intrărilor de model potențiale. Selecția optimă a subseturilor are ca rezultat reducerea dimensiunii, antrenarea mai ușoară și mai eficientă, estimări mai bune în cazul seturilor de date subdimensionate, metode de procesare mai dezvoltate și performanțe mai bune [34]. Algoritmii pentru selectarea caracteristicilor pot fi caracterizați în după funcțiile lor de evaluare. Aceștia găsesc subsetul de caracteristici cu o evaluare optimă și îl folosesc ca set de date finale pe care se realizează antrenarea. Clasificatorul rezultat este apoi trecut printr-o evaluare finală utilizând un set de date noi de testare independent de căutarea primară. Cea mai simplă operație de selectare este eliminarea datelor duplicate.

Generalizabilitatea modelului este un aspect important al dezvoltării rețelei neuronale și o înțelegere completă a acestui concept este imperativă pentru selectarea adecvată a subsetului de date. Generalizarea se referă la capacitatea ieșirilor modelului de a aproxima valorile țintă folosind intrări care nu se află în setul de antrenament. Într-un sens practic, generalizarea bună nu este întotdeauna posibilă și necesită informații satisfăcătoare de intrare corelate cu ieșiri țintă.

Pentru a evalua în mod eficient dacă rețeaua și-a atins scopul de generalizabilitate, setul de date este împărțit în subseturi de antrenare, testare și validare. În practica de modelare a rețelelor neuronale este esențial să se obțină un echilibru bun în alocarea setului de date de intrare, împărțit adesea între antrenare (75%), testare (15%) și validare (10%) [35]. Setul de date de antrenament este utilizat pentru reglarea parametrilor modelului prin calculul gradientului și actualizarea ponderilor și bias-urilor rețelei. Setul de validare este folosit în evaluarea modelării unde eroarea din setul de validare este supravegheată în timpul fazei de antrenare. Suprapotrivirea este un obstacol de semnificație fundamentală în timpul antrenamentului, cu implicații semnificative în aplicarea rețelelor neuronale. În mod ideal, erorile de validare și de antrenare se diminuează pe tot parcursul fazei inițiale a antrenamentului. Atunci când rețeaua începe să suprapotrivească datele, eroarea de validare începe să se amplifice. Atunci când eroarea de validare crește pentru un anumit număr de iterații, antrenamentul este oprit și ponderile și bias-urile la minimul erorii de validare sunt returnate (Figura 3.6). În cele din urmă, setul de date de testare este utilizat după procesul de antrenare pentru a formula o evaluare finală a modelului și cât de acceptabilă este generalizarea.

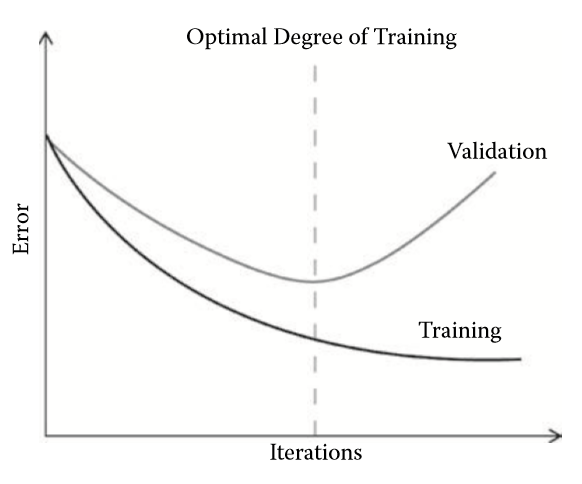


Figura 3.6 Eroarea în funcție de pasul de iterație de antrenare pentru seturile de antrenare și validare; validarea indică eroarea minimă ([29] pg 68)

**3.6 Algoritmi de antrenare**

**3.6.1 Calcularea erorii**

Pentru o problemă de clasificare, cea mai populară metodă de a găsi un set optim de ponderi este de a găsi matricea W care minimizează eroarea medie pătrată (mean squared error - MSE). Avantajul acestei funcții de eroare de eroare este că derivata sa în funcție de fiecare pondere există în întreg domeniul, fiind astfel posibilă aplicarea metodei gradientului. Dezanvantajul acesteia este că tinde să exagereze diferențele, rezultând in clasificări greșite. Dacă sunt *N* exemple în setul de antrenare X, fiecare un vector de dimensiune M, cu ieșiri țintă corespunzătoare *Dk* , un vector tot de dimensiune M, atunci MSE, este dată de:

|  |  |  |
| --- | --- | --- |
|  |  | (3.7) |

**3.4.2 Regula perceptronului**

Cel mai simplu algoritm de antrenare este regula de antrenare a perceptronului care poate fi aplicată seturilor de date liniar separabile. Aceasta constă în a modifica la fiecare pas de antrenare *k* valorile ponderilor *wij* :

|  |  |  |
| --- | --- | --- |
|  |  | (3.8) |

Unde *wij* este ponderea de neuronul *j* la intrarea *i*, *r* rste pasul de antrenare, este valoarea țintă a ieșirii *j*, este valoarea intrării *i* din setul de antrenare *k*, iar este ieșirea *j* din setul de antrenare *k*, cu expresia:

|  |  |  |
| --- | --- | --- |
|  |  | (3.9) |

Dacă ieșirea țintă este echivalentă cu ieșirea calculată, = , atunci ponderea nodului de ieșire *j* rămâne neschimbată. Procesul de învățare se termină atunci când toți vectorii de ponderi *wij* rămân neschimbați în timpul unei secvențe de antrenare succesive. Regula perceptronului este garantată să conveargă la o soluție într-un număr infinit de pași, atâta timp cât există o soluție. Dacă învățarea perceptronului este rulată pe un set de date de instruire neseparabil liniar, algoritmul nu va funcționa corect.

**3.4.3 Regula gradientului**

Atunci când se analizează perceptronii multistrat, suprafața de eroare este o funcție neliniară a matricei de ponderi *W*. Algoritmii care aproximează folosind coborârea pe gradient sunt folosiți pentru a antrena rețelele cu mai multe straturi. Pentru a antrena folosind algoritmul cu coborâre pe gradient, gradientul funcției cost cu privire la fiecare greutate *wij* a rețelei trebuie să fie calculat. Acest proces oferă informații despre modul în care fiecare schimbare în greutate va afecta eroarea globală.

|  |  |  |
| --- | --- | --- |
|  |  | (3.10) |

Unde *E* este funcția de cost. Cea mai des utilizată funcție cost este eroarea medie pătratică medie (MSE) cu formula:

|  |  |  |
| --- | --- | --- |
|  |  | (3.11) |

Unde este ieșirea rețelei neurale și depinde de ponderile din *W*, iar este ieșirea țintă. va indica panta cea mai abruptă a funcției cost în raport cu ponderile *W*; calea către minimul local va fi negativul acestei valori, cu care se actualizează *W* la pasul *k + 1* :

|  |  |  |
| --- | --- | --- |
|  |  | (3.11) |

Rata de antrenare *r* > 0 reglează dimensiunea pasului la fiecare ciclu de antrenament. Dacă rata de antrenare este selectată corect, metoda va converge la un minim local *E(w)* pentru un *r* suficient de mic, cu condiția ca gradientul să fie nenul. Valoarea lui *r* poate avea un efect substanțial asupra progresiei algoritmului, cu o prea mică valoare ducând la o prelungire nenecesară a algoritmului, iar o prea mare valoare ducând la o posibilă oscilare și o convergență nereușită la un minim.

**3.4.4 Algoritmul Levenberg-Marquardt**

Algoritmul Levenberg-Marquardt constă în actualizarea parametrilor, la iterația *k*, prin:

|  |  |  |
| --- | --- | --- |
|  |  | (3.12) |

Unde este un coeficient pozitiv, este vectorul erorilor la momentul *k*:

|  |  |  |
| --- | --- | --- |
|  |  | (3.12) |

Și *Jk* este Jacobianul erorilor la momentul *k*:

|  |  |  |
| --- | --- | --- |
|  |  | (3.13) |

# 3.4.5 Algoritmul BFGS (Broyden–Fletcher–Goldfarb–Shanno)

Algoritmul BFGS constă în actualizarea parametrilor, la iterația *k*, prin:

|  |  |  |
| --- | --- | --- |
|  |  | (3.14) |

Unde *μi* este pozitiv, *M0* este matricea dientitate și unde *Mi* o aproximare, calculată iterativ, a inversei matricei hessiene; acesta din urmă este calculată, la fiecare iterație, de către:

|  |  |  |
| --- | --- | --- |
|  |  | (3.15) |
|  |  | (3.16) |
|  |  | (3.17) |

**Capitolul 4 - Arhitectura sistemului**

**4.1 formatul PDB**

Protein Data Bank (PDB) este o arhivă care conține structuri tridimensionale de macromolecule biologice determinate experimental care servește o comunitate globală de cercetători, educatori și studenți. Datele conținute în arhivă includ coordonatele atomice, structura cristalografică, factori și date experimentale RMN (rezonanta magnetica nucleara). Pe lângă coordonate, fiecare intrare include și nume de molecule, informații despre structurile primare și secundare, referințe pentru baze de date de secvențe, acolo unde este cazul, precum și informații despre asamblarea liganda și a biologica, detalii privind colectarea datelor și citările bibliografice.

Înregistrările ATOM prezintă coordonatele atomice pentru aminoacizii și nucleotidele standard. Ei conțin de asemenea factorii de ocupanță și de temperatură pentru fiecare atom. Coordonate pentru substanțe chimice non-polimerice și residuuri non-standard utilizează tipul de înregistrare HETATM. Residuurile HETATM nu sunt întotdeauna conectate cu alte residuuri. Sarcina electrica poate apărea opțional într-o inregistrare. Proteinele sunt codificate în ordinea 5’ -> 3’, unde 5’ este gruparea liberă amino, iar 3’ este gruparea liberă carboxil. Structura unei înregistrări este dată în Anexa B în Tabelul B.1, iar un exemplu de conținut al unui fișier în Figura 4.1:

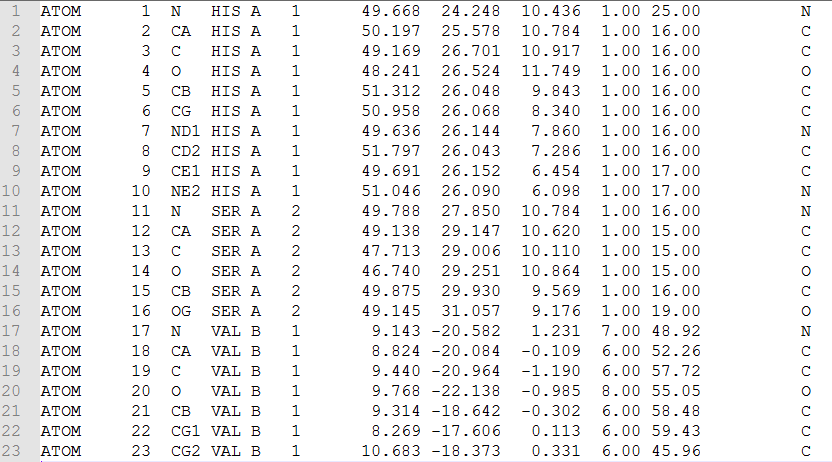


Figura 4.1 Exemplu de conținut al unui fișier .pdb

În exemplul din Figura 4.1 avem trei reziduuri (aminoacizi): histidină (HIS) urmată de serină (SER), urmată de valină (VAL). Atomii din care sunt formate residuurile sunt în ultima coloană, în a doua coloană sunt identificatorii fiecarui atom în proteină, iar în a treia coloană sunt scrise codificat pozițiile acestora în aminoacid. Coloana patru conține numele aminoacizilor compuși din atomi. Următoarea coloană conține identificatorul lanțului de aminoacizi; o proteină poate fi compusă din mai multe lanțuri de aminoacizi, iar acestea sunt notate de obicei în ordine alfabetică. Aici avem doua lanțuri: A și B. Următoarea coloana este identificatorul fiecarui residuu individual intr-un lant de aminoacizi. Urmeaza trei coloane care conțin coordonatele, iar apoi, ocupanța și factorul de temeratură.

Formatul mmCIF (fișier .cif) este similar, dar conține date in plus. Cele necesare pentru determinarea formei proteinei sunt aceleasi. Exemplu în Figura 4.2:

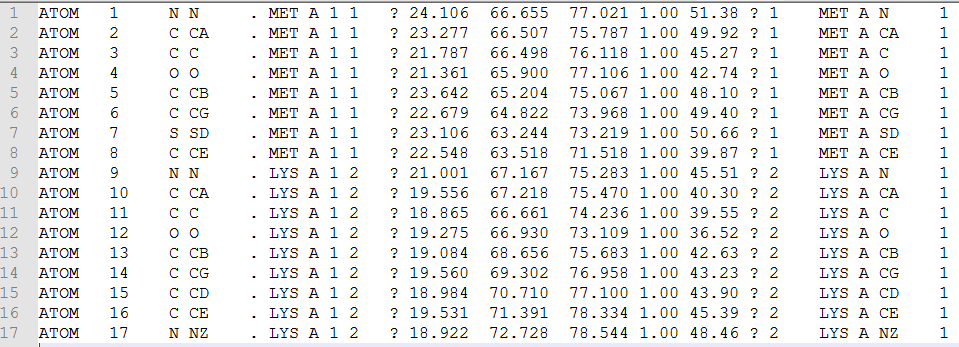


Figura 4.2 Exemplu de conținut fișier .cif

**4.2 Formatul Fasta**

Formatul FASTA este un format bazat pe text pentru a reprezenta fie secvențe de nucleotide (ADN sau ARN), fie secvențe peptidice, în care nucleotidele sau aminoacizii sunt reprezentați folosind coduri de o singură literă. Formatul permite, de asemenea, ca secvențele să fie precedate de numele acestora și de comentarii. Formatul provine din pachetul software FASTA, dar a devenit un standard în domeniul bioinformaticii. Un fișier poate conține mai multe lanțuri de proteine sau nucleotide. Acestea încep cu numele și descrierea după simbolul „>”. Începând cu rândul următor este scrisă secveța proteinei, în litere. Codificarea proteinelor este standardizată și detaliată în Anexa B în Tabelul B.2.

**4.3 Utilizarea datelor de către rețeaua neurală**

Rețeaua neurală are un număr de intrări care poate fi setat, unde fiecare intrare conține un aminoacid. Numărul de ieșiri este același cu numărul de intrări, unde fiecare ieșire conține trei valori care reprezintă cele trei coordonate carteziene ale aminoacidului, relativ la aminoacidul precedent din lanț. Rețeaua de intrare folosește o fereastră alunecătoare care parcurge proteina pentru antrenare. De aceea, ordinea aminoacizilor în proteină afectează starea rețelei. Fereastra se poate suprapune parțial cu fereastra de la pasul precedent și această suprapunerese măsoară în numărul de aminoacizi incluși (Figura 4.3). Pasul cu care fereastra avansează este egal cu *p* = *f* – *c*, unde *p* este dimensiunea pasului, *f* este dimensiunea ferestrei, iar *c* este dimensiunea suprapunerii.

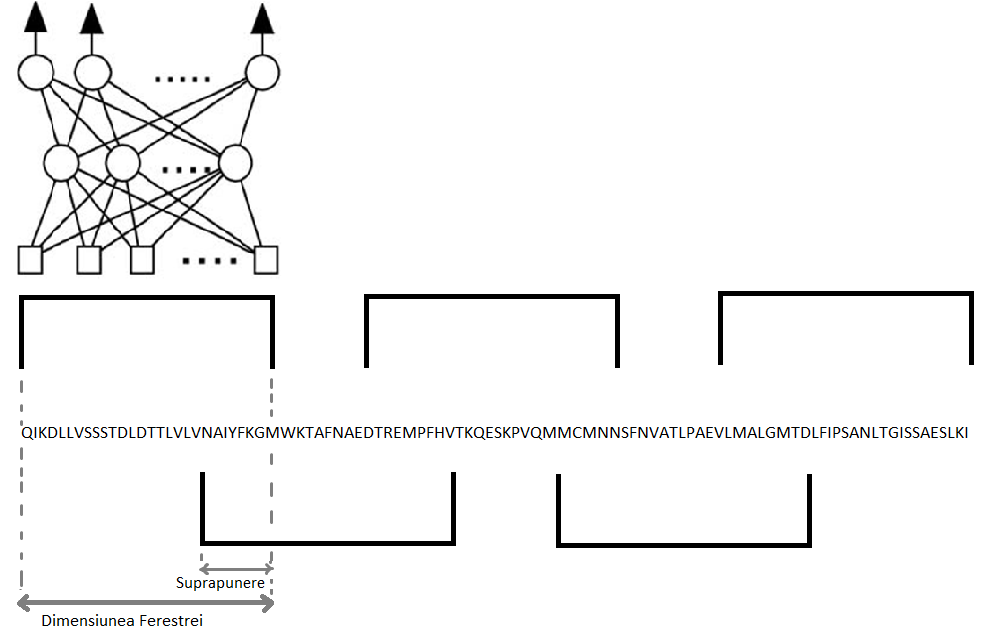


Figura 4.3 Rețeaua neurală cu fereastra aferentă alunecă de-a lungul proteinei lăsând o suprapunere.

**4.4 Realizarea interfeței utilizatorului**

Interfața grafică utilizatorului a fost realizată cu ajutorul utilitarului Guide (Graphical User Interface Development Environment) din Matlab. Acesta permite crearea de interfețe care conțin elemente precum câmpuri text, butoane, liste, grafice și checkbox-uri, pe de-o parte prin drag-and-drop pentru aranjarea elemtelor în interfață și pe de altă parte prin scrierea de funcții pentru controlarea comportamentului elementelor. Un program care conține o interfață Guide este compus din cel puțin două fișiere: un fișier binar .fig care conține aranjamentul geometric al elementelor și un fișier text .m care conține funcțiile Matlab care controlează toate acțiunile executate de elementele interfeței. Am denumit fișierele interfeței AnnGen.fig și AnnGen.m, abreviere pentru “artificial neural network generator”.

Elementele, numite și widet-uri, sunt componentele care formează o interfață. În continuare sunt prezentați pașii pentru a realiza un buton “Browse” și un câmp de text editabil, utilizate în multiple locuri în interfață. Rostul acestor butoane este de a deschide un Explorer Windows pentru a selecta un fișier sau un director din memorie și a scrie și salva calea acestuia în câmpul text din stânga fiecărui buton.

Guide oferă o interfață a sa proprie de editare (la care ne vom referi de acum ca editor) pentru crearea și plasarea elementelor într-o grilă. Diferitele tipuri de elemente sunt aranjate în partea stângă a editorului, elementul pentru buton fiind primul (Anexa A, Figura A.2.).

Proprietățile elementului pot fi schimbate prin deschiderea ferestrei ”Inspector” prin dublu-click pe elementul care urmează a fi modificat în editor. Fereastra ”Inspector” va avea câmpuri diferite pentru fiecare tip de element pentru care este deschisă. Pentru butoanele ”Browse” am modificat numele care apare pe buton prin introducerea în câmpul ”String” a textului ”Browse”. Pentru câmpul editabil, valoarea din ”String” setează textul din câmp care apare la inițializarea interfeței, în acest caz este ”Folder Path”. Fiecare element are asociat o variabilă, a cărei nume se modifică din câmpul ”Tag”. Pentru primul buton care apare în proiectul de față, în parte din stânga sus, numele a fost setat ca “fastaBrowsePushButton”, iar câmpul editabil din stânga ca “fastaFolderPathText” (Figura 4.4).

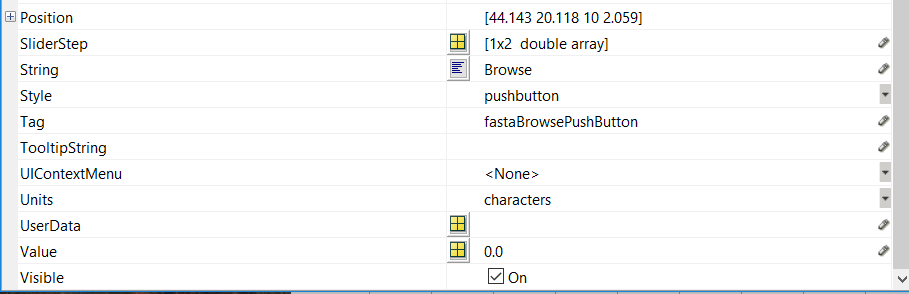


Figura 4.4 Fereastra Inspector pentru un element de tip buton

Fiecare element are un număr de funcții asociate în fișierul .m. Funcția comună tuturor este cea de “Callback” care este apelată atunci când elementul este utilizat (apăsat în cazul butoanelor, se apasă Enter sau se deselectează câmpul în cazul câmpurilor editabile). O altă funcție care apare la elemente precum câmp editabil sau listă este “CreateFcn” care se execută la inițializarea interfeței. Numele acestor funcții depinde de Tag-ul elementului asociat. Numele funcției callback a butonului în cauză este, de exemplu, “fastaBrowsePushButton\_Callback”, iar a câmpului editabil “fastaFolderPathText\_Callback”. Antetul funcțiilor sunt generate automat la crearea unui element. Acestea conțin un parametru de interes, numit “handles”.

handles este singura variabilă globală a interfeței. Orice variabilă utilizată într-o funcție din fișierul AnnGen.m este locală funcției respective și expiră la încheierea acesteia. Variabila “handles” este o structură care poate conține orice tip de variabilă ca o proprietate a sa. Pentru butonul “fastaBrowsePushButton”, calea către directorul selectat este obținută din apelarea funcției guidata și este salvată global în proprietatea handles.fastaFolder.

O funcție poate apela alte funcții. O funcție care este apelată de o alta trebuie să returneze variabila globală handle. fastaBrowsePushButton\_Callback apelează fastaFolderPathText\_Callback (Figura 4.5).

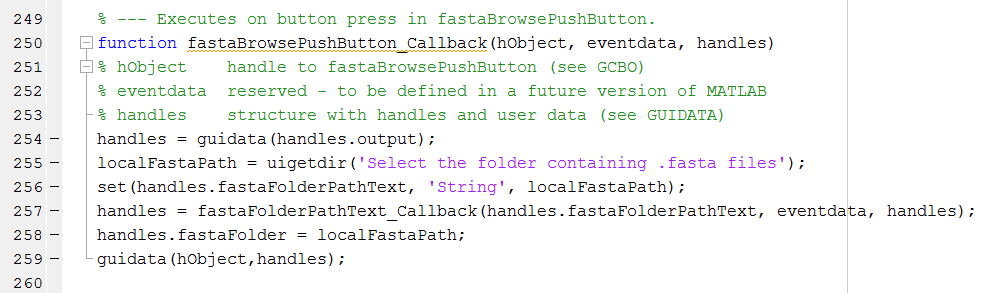


Figura 4.5 Callback-ul butonului fastaBrowsePushButton

fastaFolderPathText\_Callback, deoarece execuția sa poate fi îndelungată, începe cu un set de comenzi care transformă cursorul în regim de așteptare (clepsidră sau cerc, depinde de sistemul de operare) atâta timp cât cursorul este deasupra interefeței. Funcția extrage detaliile tuturor fișierelor care se termină în “.fasta” din fișier, convertește în celule și salvează numele lor într-un vector în handles.fastaFiles și va apela fastaListbox\_Callback, care populează lista de mai jos cu numele fișierelor găsite. Dacă calea directorului este invalidă sau nu conține fișiere .fasta, o fereastră de eroare va apărea, indicând acest lucru (Figura 4.6).

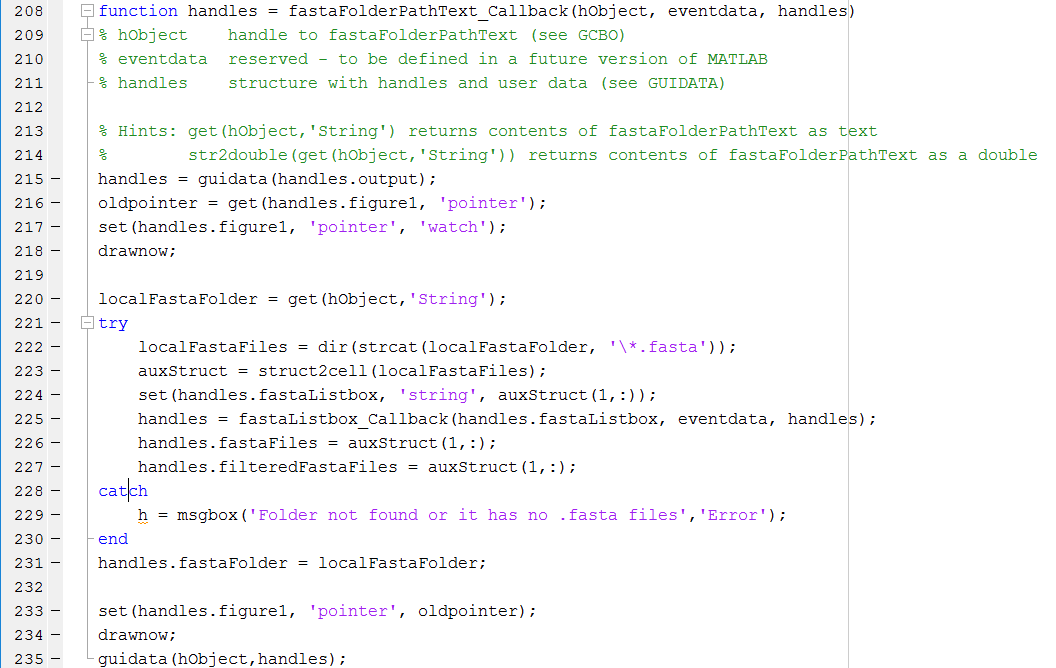


Figura 4.6 Callback-ul câmpului editabil fastaFolderPathText

**4.5 Antrenarea unei noi rețele neurale**

În partea intitulată ”Create Neural Network Parameters” a interfeței sunt setați parametrii pentru antrenarea unei noi rețele neurale astfel: directorul care conține datele de antrenare în partea de sus, dimensiunea ferestrei, numărul de neuroni ascunși, funcția de antrenare, resursele folosite, tirul rețelei, numele rețelei și împărțirea setului de date în antrenare, validare și testare (Anexa A, Figura A.4).

Platforma folosește funcții din toolboxurile Machine Learning și Bioinformatics din Matlab. Funcția care realizează procesarea datelor, crearea și antrenarea a unei noi rețele este generateNeuralNetwork (Anexa C Figura C.1). Structura rețelei este construită de funcția createMultiOutputAnn care poate fi feedforward sau recurentă (rețeaua recurentă se realizează prin funcția din toolbox-ul Machine Learning layercnet). Aceasta are un strat ascuns principal care conține neuronii care fac computația și câte un strat pentru fiecare ieșire. Straturile de ieșire nu fac nimic, ci sunt necesare într-o rețea cu mai culte ieșiri în Matlab. Numărul straturilor și conexiunile dintre acestea sunt factorii care determină arhitectura. Straturile corespunzătoare ieșirilor sunt setate în vectorul ann.layerConnect cu valoarea 0. Aceste straturi sunt conectate la stratul ascuns prin setarea matricii ann.layerConnect(i, 1), care creează o conexiune de la stratul 1 la stratul *i* (Figura 4.7).

Dimensiunea intrării și a ieșirii (lungimea ferestrei) este setată automat când sunt introduse primele date de antrenare. Funcția de activare a stratului ascuns este tangent-sigmoidă, iar a fiecărui strat de ieșire este liniară pentru a nu afecta ieșirile. learnFcn atașată fiecărei ponderi din matricea de ponderi layerWeights definește algoritmul de antrenare care va fi folosit pentru a actualiza ponderea de pe acea legătură. Matlab oferă doisprezece algoritmi de antrenare dintre care am folosit algoritmii Levenberg-Marquardt și BFGS Quasi-Newton. Diagrama rezultatului pentru o rețea feedforward apare în Figura 4.8, iar pentru una recurentă în Figura 4.9.

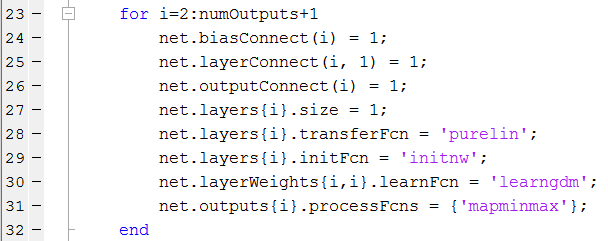


Figura 4.7 Codul din createMultiOutputAnn care realizează conexiunile la ieșirile multiple

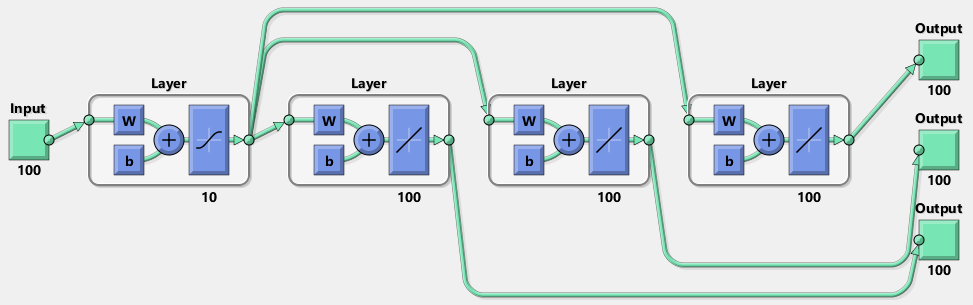


Figura 4.8 O rețea feedforward cu o intrare și trei ieșiri; numărul de sub intrare și ieșiri este dimensiunea unui set de date (fereastra) și este determinat de datele de antrenare folosite.

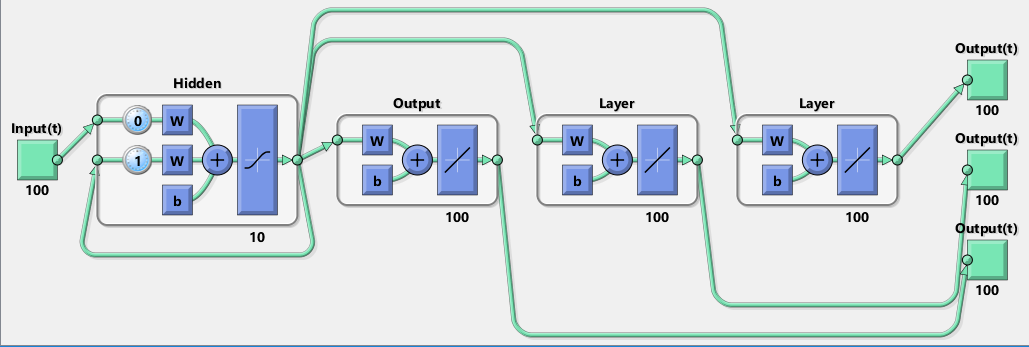


Figura 4.9 O rețea recurentă cu aceleași intrări și ieșiri ca rețeaua feedforward precedentă.

4.6 Procesarea datelor

Coordonatele trebuie extrase din fiecare fișier individual .cif într-un for, prin citire rând cu rând și adăugarea datelor rândurilor care încep cu ”ATOM” într-o matrice de celule, numită atomData. Aceasta va avea dimensiunea *n* x 22, *n* fiind numărul de intrări din setul de date care va fi utilizat (care reprezintă atomi individuali din coloana aminoacizilor). Matricea este verificată dacă conține aminoacizi și nu altceva prin funcția ismember și un vector care conține denumirile tuturor aminoacizilor admisibili. Dacă una sau mai multe linii din matrice conțin altceva decât ce este în vector, se trece la următoarea iterație a for-ului.

Un fișier care conține o proteină va avea acea proteină împărțită în lanțuri, reprezentate de litere în ordine alfabetică. Deoarece sfârșitul unui lanț nu coincide cu începutul altui lanț, acestea nu pot fi concatenate și introduse în rețeaua neurală, ci mai întâi trebuie separate pentru a produce rezultate care reflectă realitatea. Fiecare lanț este apoi separat în subdiviziuni de lungimea ferestrei rețelei și procesate de funcția formatPdbData (Anexa C, Figura C.2). Aceasta extrage numai datele necesare: reziduurile într-o matrice residueData și coordonatele într-un vector de celule cu trei elemente în care fiecare element este o matrice care conține o coordonată X, Y și Z. Datele sunt transformate din formatul liniar al matricii de intrare atomData într-un format matriceal cu numărul de linii fiind egal cu dimensiunea ferestrei. Datele vor fi inserate în această nouă matrice din stânga-sus în jos, umplând coloană cu coloană până când toate date vor fi inserate. Rezultatul va fi o matrice de dimensiune *f* x *m*, unde

*f* este dimensiunea ferestrei,

*m* este parte întreagă din *n*/*f*,

*n* este lungimea lanțului din atomData introdus, care conține reziduurile

Reziduurile sunt de asemenea convertite în format numeric prin funcțiile aa2intCustom și aminolookup. Se va mai returna și un vector compus din trei matrici asemănătoare care conțin coordonatele. Elementele rămase pentru a completa matricile de ieșire au valoarea NaN.

Rezultatele sunt concatenate la matricile totalResidueData și totalCoordinateData. Se vor concatena lanțuri procesate ale fișierului până se ajunge la o dimensiune acceptabilă pentru a le folosi la antrenare. Este impractic a se concatena datele unui fișier întreg sau a tuturor fișierelor și a le folosi pentru antrenare deoarece execuția durează prea mult și are tendința de a bloca calculatorul. Așa că acceași rețea este antrenată repetat cu seturi diferite de date. Un set de antrenare va arăta astfel, începănd cu matricea care conține reziduurile în format numeric:

totalResidueData =

[5 1 13 18 5 1 13 1

5 1 13 18 5 1 13 1

5 8 13 18 5 7 13 1

5 8 13 NaN 5 7 13 1

1 8 5 NaN 1 7 5 1

1 8 5 NaN 1 7 5 1

1 8 5 NaN 1 7 5 NaN

1 8 5 NaN 1 7 5 NaN

1 13 18 NaN 1 13 1 NaN]

Figura 4.10 Un set de antrenare de reziduuri

În continuare este trecută prima celulă (care este coordonata X) a vectorului de celule de coordonate. Această celulă conține o matrice de aceeași dimensiune ca matricea reziduurilor și fiecare element (*i*, *j*) al matricii reprezintă coordonata X a atomului constituent al unui reziduu de la poiția (*i*, *j*) în matricea de reziduuri. Toate cele trei matrici sunt trecute în Anexa C, Figura C3. Când rețeaua este antrenată o fereastră este deschisă care arată progresul, eroarea și alte informații (Anexa A, Figura A.3).

totalCoordinateData = {

[12.900 12.247 -27.567 -15.062 12.900 12.247 -27.567 12.247;

12.900 12.247 -27.567 -15.062 12.900 12.247 -27.567 12.247 ;

12.900 -19.868 -27.567 -15.062 12.900 3.797 12.247 12.247 ;

12.900 -19.868 -27.567 NaN 12.900 3.797 12.247 12.247 ;

12.247 -19.868 12.900 NaN 12.247 3.797 12.247 12.247 ;

12.247 -19.868 12.900 NaN 12.247 3.797 12.247 12.247 ;

12.247 -19.868 12.900 NaN 12.247 3.797 12.247 NaN ;

12.247 -19.868 12.900 NaN 12.247 3.797 12.247 NaN ;

12.247 -27.567 -15.062 NaN 12.247 -27.567 12.247 NaN ]

; [...]; [...]

}

**Figura 4.11** Prima celulă a setuluid e antrenare cu coordonate

Rețeaua este apoi pusă într-o structură și salvată într-un fișier:

tempAnnStorage = struct('ANN', ann, 'TR', tr, 'Codification', codification, 'NetworkName', networkName, 'NetworkType', networkType, 'Crossover', crossover, 'TrainingFunction', trainingFunction, 'PlotData', []);

...

save(handles.ANNFile, 'ANNStorage');

**Figura 4.12** Cod din generateFeedforwardNetwork și createNewANNPushButton\_Callback pentru salvarea rețelei

**4.7 Utilizarea unei rețele deja antrenate**

În partea intitulată ”Use Neural Network Parameters” din interfață, jumătatea de sus are de a face cu fișierele de intrare .fasta. Lista de fișiere poate fi filtrată cu text introdus în câmpul ”Search” și conținutul unui fișier individual poate fi vizualizat selectând acel fișier și apăsând butonul ”Details”. Codul folosit poate fi regăsit în Anexa 2.1. Aceată listă este de asemenea folosită atunci când o rețea neurală existentă este utilizată. În panoul de jos denumit ”Artificial Neural Networks” se poate încărca o rețea neurală deja antrenată dintr-un fișier .mat din memoria calculatorului prin câmpul și butonul indicate de textul ”ANN Storage Filepath”. Un fișier poate conține mai multe rețele care vor fi afișate numerotat în lista de mai jos. Când un număr din listă este selectat rețeaua este încărcată în Matlab și tabelul de lângă se încarcă cu detaliile acesteia. În partea de sus, rețeaua poate fi folosită cu datele de intrare dintr-un fișier .fasta selectat apăsând ”Use ANN on Selected File” sau din toate fișierele filtrate (care apar în listă) apăsând ”Use ANN on All Files”. Calea către un director în care fișierele generate să fie salvate trebuie scrisă în câmpul ”Output Folder” (Anexa A, Figura A.5).

Utilizarea rețelei se face în funcția useANN (Anexa C, Figura C.4). Rețeaua primește un fișier .fasta ca intrare, în care fiecare aminoacid este reprezentat de o literă. Un exemplu, cu numele ”3GYN.fasta”, este prezentat în Anexa A, Figura A.5. Acesta este tratat ca un vector unidimensional. În funcție de dimensiunea ferestrei și a suprapunerii rețelei utilizate, care sunt memorate împreună cu aceasta, vectorul de date de intrare este segmentat în intervale de dimensiunea ferestrei *f*, iar intervalele sunt plasate la distanța *f* – *p* unul de celălalt, unde *p* este dimensiunea suprapunerii. În zonele cu suprapuneri, rețeaua va da două rezultate pentru fiecare aminoacid, așa că rezultatul final va fi media aritmetică a fiecărei perechi de coordonate.

În formatul PDB, aminoacizii sunt reprezentați diferit decât în .fasta. Un aminoacid este o secvență de rânduri, preprezentând atomi, de lungime variabilă pentru fiecare tip (de exemplu, Valina este compusă din șapte atomi, deci șapte rânduri în fișierul .cif). Pentru a realiza compatibilitate între cele două formate, aminoacizii din vectorul de intrare sunt duplicați de un număr corespunzător tipului lor, înainte de a fi convertiți în format numeric și trimiși la rețea. Valorile duplicării sunt stocate într-un dicționar din funcția formatFastaData, care construiește conținutul fișierului .cif corespunzător datelor de intrare, care urmează a fi apoi completat cu coordonatele generate de rețea (Anexa C.6). În Figura 4.13 este exemplificată procesarea unei secvențe scurte .fasta pentru a fi utilizată. Rezultatul din fișierul .cif este prezentat în Anexa A, Figura A.6.

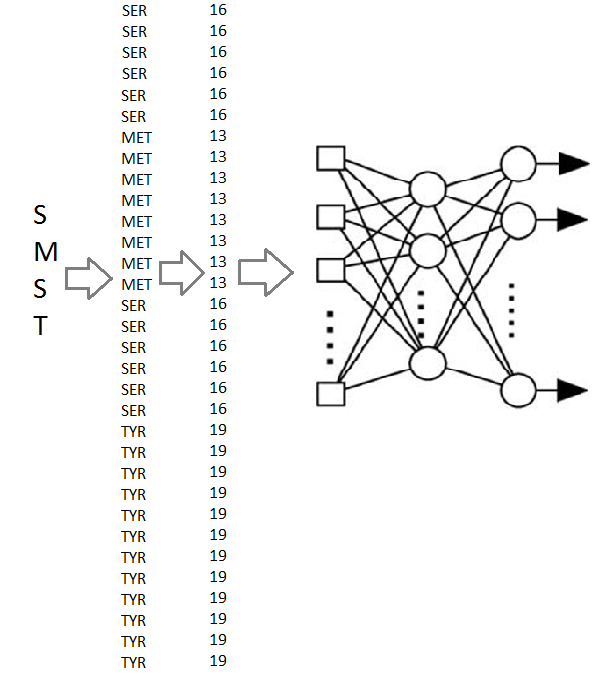


Figura 4.13 Secvența fasta SMST, duplicată conform atomilor din coloanele vertebrale ale fiecărui aminoacid și convertiți în format numeric înainte de a fi trimiși rețelei neurale

5. Rezultate

5.1 Datele folosite

Pentru antrenare am folosit două seturi de date obținute din baza de date RCSB PDB, <https://www.rcsb.org/> [36]. Primul este un set de glicoproteine de înveliș a diverse specii de virusuri. Al doilea este un set de hemoglobine cu mutații.

5.2 Eroarea rețelei

Resursele calculatorului folosit au fost un procesor Intel I6 6700HQ și un card de grafica Nvidia Geforce GTX 960M. Am rulat rețele feedforward și recurente cu dimensiuni ale ferestrei de 100, 120 și 150 și cu suprapuneri de 0, 30 și 50, folosind algoritmii Levenberg-Marquardt, BFGS Quasi-Newton și algoritmul Scaled Conjugate Gradient Backpropagation, care este un algoritm cu coborâre pe gradient specific utilizării cardului de grafică. Am testat apoi rețelele pe fișierele .fasta corespunzătoare setului de antrenare și fișiere din afara setului. Am comparat rezultatele obținute folosind eroarea medie pătratică și le-am trecut într-un tabel.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | Tip Rețea | Algoritm | Fereastră | Suprapunere | Eroarea | | 1 | Feedforward | Levenberg-Marquardt | 100 | 0 |  | | 2 | Feedforward | Levenberg-Marquardt | 100 | 30 |  | | 3 | Feedforward | Levenberg-Marquardt | 100 | 50 |  | | 4 | Feedforward | Levenberg-Marquardt | 120 | 0 |  | | 5 | Feedforward | Levenberg-Marquardt | 120 | 30 |  | | 6 | Feedforward | Levenberg-Marquardt | 120 | 50 |  | | 7 | Feedforward | Levenberg-Marquardt | 150 | 0 |  | | 8 | Feedforward | Levenberg-Marquardt | 150 | 30 |  | | 9 | Feedforward | Levenberg-Marquardt | 150 | 50 |  | | 10 | Feedforward | BFGS Quasi-Newton | 100 | 0 |  | | 11 | Feedforward | BFGS Quasi-Newton | 100 | 30 |  | | 12 | Feedforward | BFGS Quasi-Newton | 100 | 50 |  | | 13 | Feedforward | BFGS Quasi-Newton | 120 | 0 |  | | 14 | Feedforward | BFGS Quasi-Newton | 120 | 30 |  | | 15 | Feedforward | BFGS Quasi-Newton | 120 | 50 |  | | 16 | Feedforward | BFGS Quasi-Newton | 150 | 0 |  | | 17 | Feedforward | BFGS Quasi-Newton | 150 | 30 |  | | 18 | Feedforward | BFGS Quasi-Newton | 150 | 50 |  | | 19 | Feedforward | Procesor Grafică | 100 | 0 |  | | 20 | Feedforward | Procesor Grafică | 100 | 30 |  | | 21 | Feedforward | Procesor Grafică | 100 | 50 |  | | 22 | Feedforward | Procesor Grafică | 120 | 0 |  | | 23 | Feedforward | Procesor Grafică | 120 | 30 |  | | 24 | Feedforward | Procesor Grafică | 120 | 50 |  | | 25 | Feedforward | Procesor Grafică | 150 | 0 |  | | 26 | Feedforward | Procesor Grafică | 150 | 30 |  | | 27 | Feedforward | Procesor Grafică | 150 | 50 |  | | 28 | Recurentă | Levenberg-Marquardt | 100 | 0 |  | | 29 | Recurentă | Levenberg-Marquardt | 100 | 30 |  | | 30 | Recurentă | Levenberg-Marquardt | 100 | 50 |  | | 31 | Recurentă | Levenberg-Marquardt | 120 | 0 |  | | 32 | Recurentă | Levenberg-Marquardt | 120 | 30 |  | | 33 | Recurentă | Levenberg-Marquardt | 120 | 50 |  | | 34 | Recurentă | Levenberg-Marquardt | 150 | 0 |  | | 35 | Recurentă | Levenberg-Marquardt | 150 | 30 |  | | 36 | Recurentă | Levenberg-Marquardt | 150 | 50 |  | | 37 | Recurentă | BFGS Quasi-Newton | 100 | 0 |  | | 38 | Recurentă | BFGS Quasi-Newton | 100 | 30 |  | | 39 | Recurentă | BFGS Quasi-Newton | 100 | 50 |  | | 40 | Recurentă | BFGS Quasi-Newton | 120 | 0 |  | | 41 | Recurentă | BFGS Quasi-Newton | 120 | 30 |  | | 42 | Recurentă | BFGS Quasi-Newton | 120 | 50 |  | | 43 | Recurentă | BFGS Quasi-Newton | 150 | 0 |  | | 44 | Recurentă | BFGS Quasi-Newton | 150 | 30 |  | | 45 | Recurentă | BFGS Quasi-Newton | 150 | 50 |  | | 46 | Recurentă | Procesor Grafică | 100 | 0 |  | | 47 | Recurentă | Procesor Grafică | 100 | 30 |  | | 48 | Recurentă | Procesor Grafică | 100 | 50 |  | | 49 | Recurentă | Procesor Grafică | 120 | 0 |  | | 50 | Recurentă | Procesor Grafică | 120 | 30 |  | | 51 | Recurentă | Procesor Grafică | 120 | 50 |  | | 52 | Recurentă | Procesor Grafică | 150 | 0 |  | | 53 | Recurentă | Procesor Grafică | 150 | 30 |  | | 54 | Recurentă | Procesor Grafică | 150 | 50 |  | |  |
|  |  |  |

**Anexe**

**Anexa A**

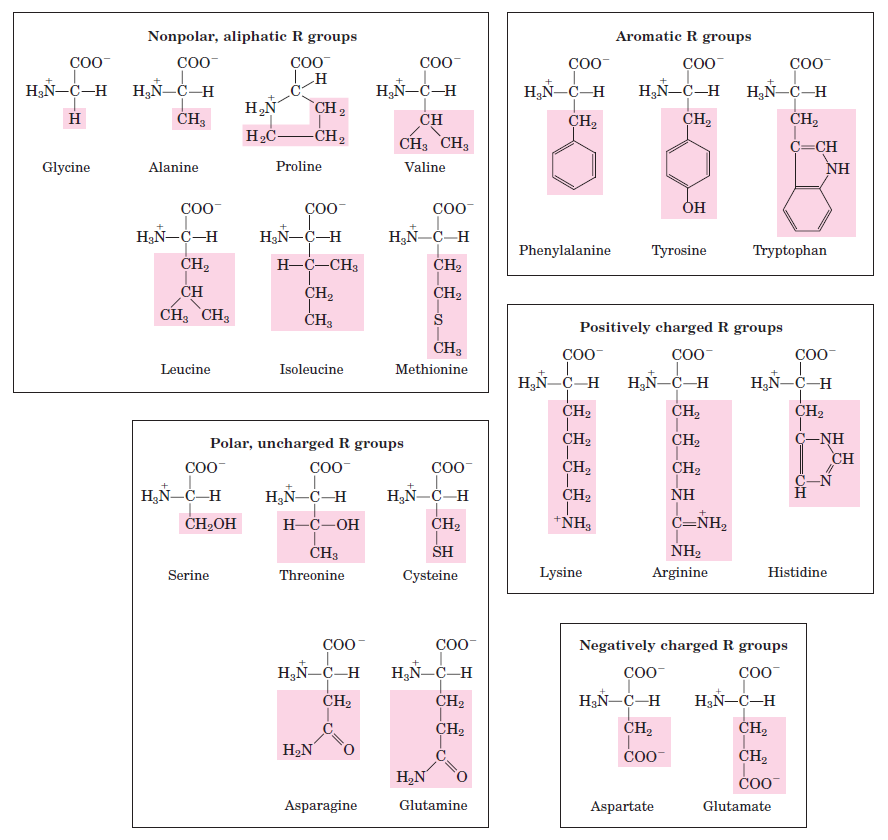


Figura A.1 Cei 20 de aminoacizi aranjați in cinci clase ([24] pg 79)

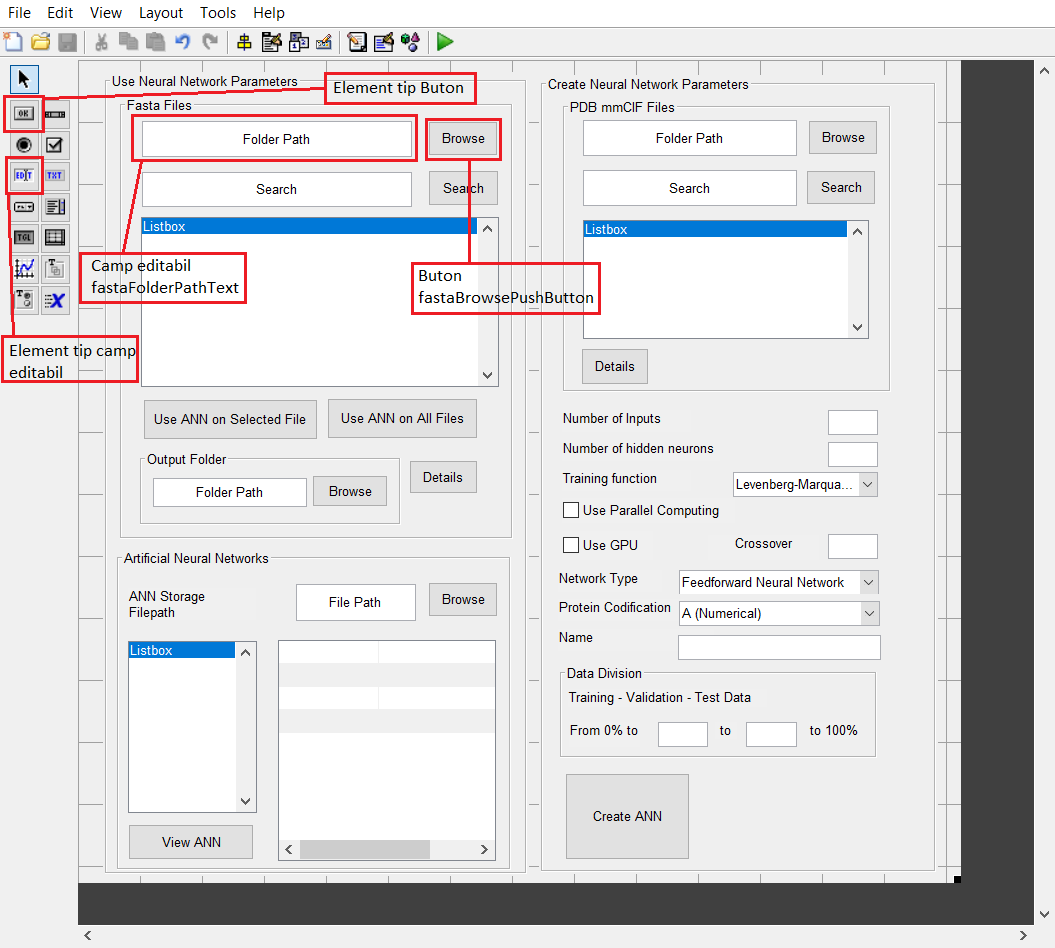


Figura A.2 Interfața de editare a toolboxu-ului Guide

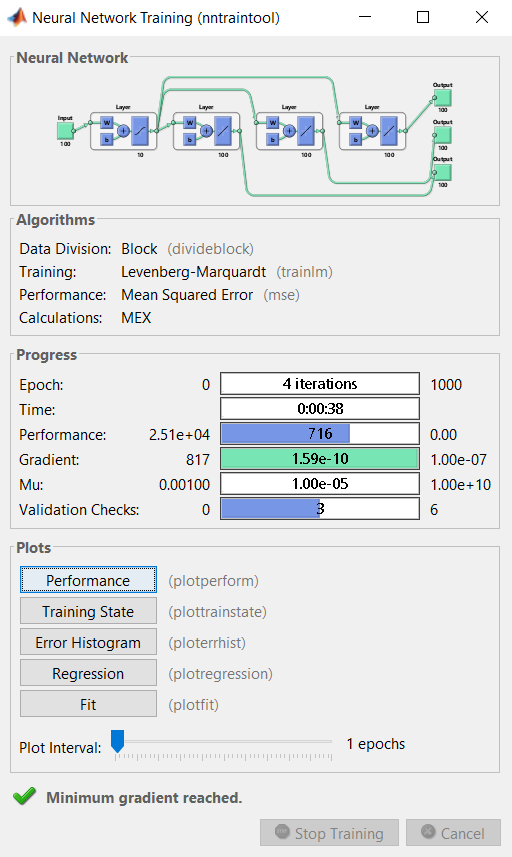


Figura A.3 Fereastra care indică antrenarea unei rețele

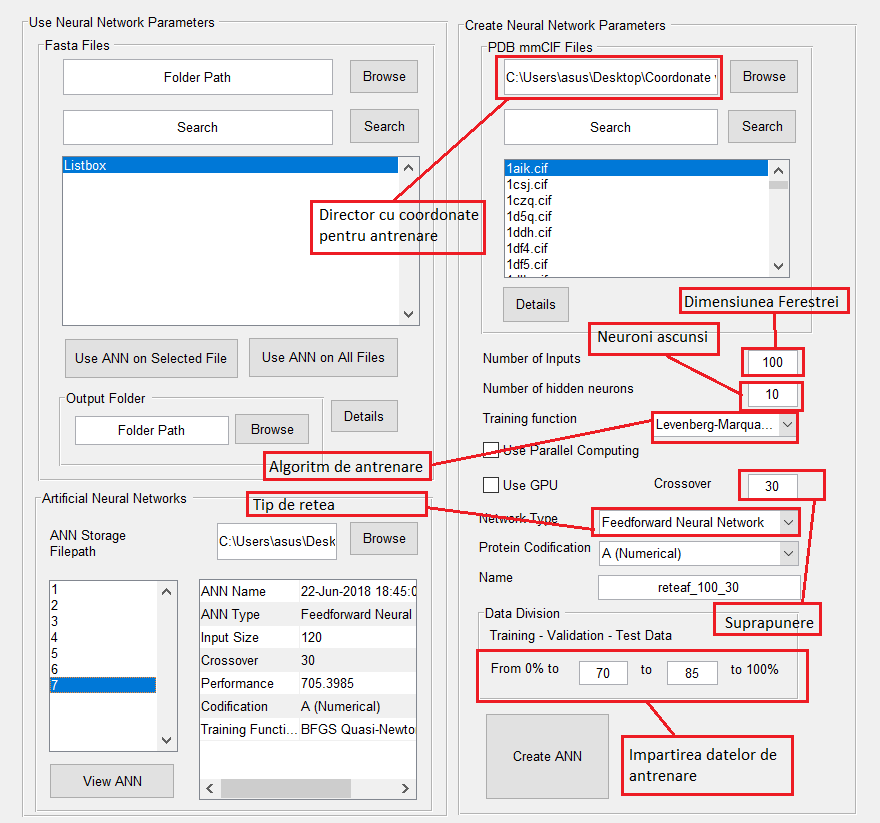
****

Figure A.4 Starea interfeței pentru antrenarea unei noi rețele neurale

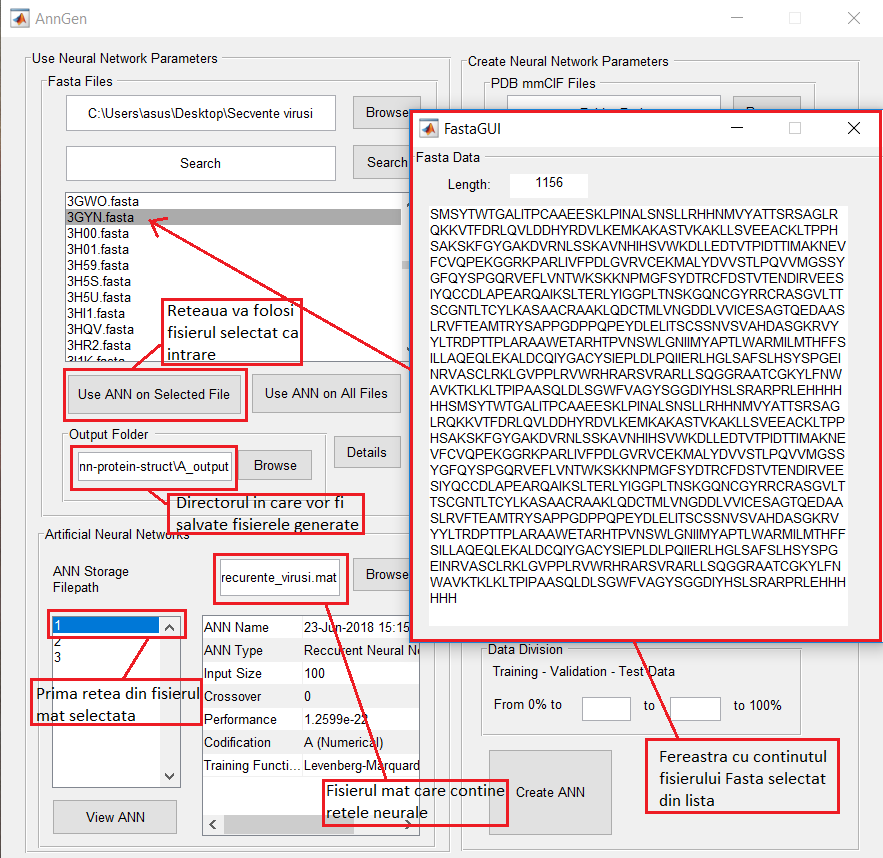
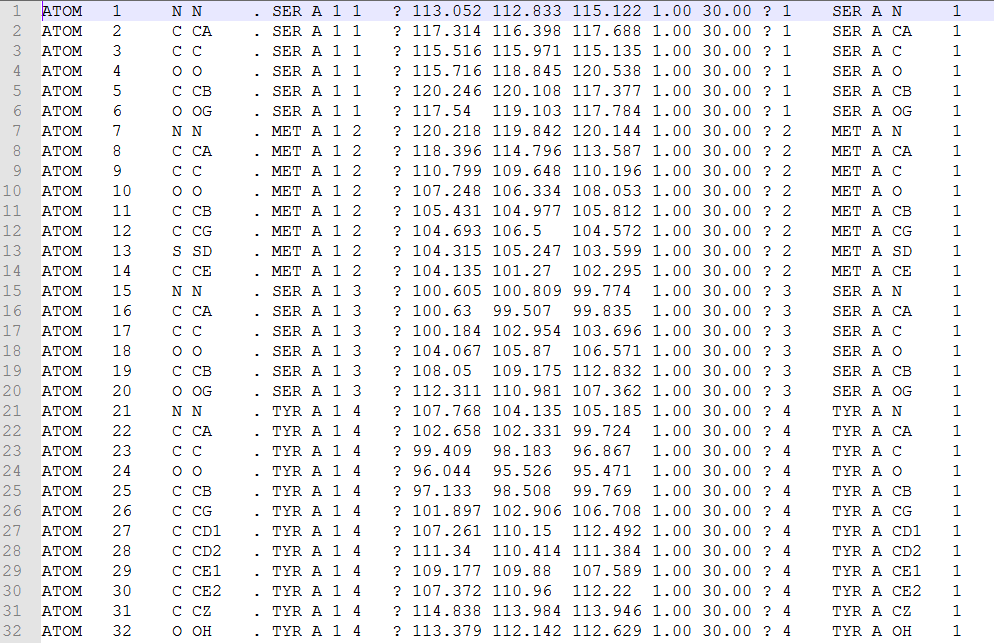


Figura A.5 Interfața pregatită pentru a utiliza o rețea neurală antrenată cu un fișier .fasta ca intrare; Rezultatul va fi un fișier PDB/mmCif cu același nume ca și fișierul .fasta de intrare salvat în directorul Output specificat



**Figura A.6** Secțiune a fișierului 3GYN.cif rezultat în urma utilizării rețelei neurale asupra fișierului 4GYN.fasta.

**Anexa B**

|  |  |  |  |
| --- | --- | --- | --- |
| Coloana | Tipul de date | Câmp | Definiție |
| 1 - 6 | Numele Inregistrarii | "ATOM " |  |
| 7 - 11 | Integer | serial | Numarul serial al atomului |
| 13 - 16 | Atom | name | Numele Atomului |
| 17 | Character | altLoc | Indicator alternativ al locatiei |
| 18 - 20 | Nume Residuu | resName | Numele Residuului |
| 22 | Character | chainID | Identificatorul lantului |
| 23 - 26 | Integer | resSeq | Numarul secventei de residuuri |
| 27 | AChar | iCode | Cod pentru insertia de residuuri |
| 31 - 38 | Real(8.3) | x | Coordonata ortogonala X in Angstromi |
| 39 - 46 | Real(8.3) | y | Coordonata ortogonala Y in Angstromi |
| 47 - 54 | Real(8.3) | z | Coordonata ortogonala Z in Angstromi |
| 55 - 60 | Real(6.2) | occupancy | Ocupanta |
| 61 - 66 | Real(6.2) | tempFactor | Factorul de Temperatura |
| 77 - 78 | String(2) | element | Simbolul Elementului |
| 79 - 80 | String(2) | charge | Sarcina electrica a atomului |

**Tabelul B.1** Structura unei înregistrări în formatul PDB/mmCIF

|  |  |
| --- | --- |
| Cod | Aminoacid |
| A | [Alanin](https://en.wikipedia.org/wiki/Alanine" \o "Alanine)ă |
| B | Acidc Aspartic (D) sau Asparagină (N) |
| C | [Cystein](https://en.wikipedia.org/wiki/Cysteine" \o "Cysteine)ă |
| D | [Acid](https://en.wikipedia.org/wiki/Aspartic_acid" \o "Aspartic acid) Aspartic |
| E | Acid [Glutamic](https://en.wikipedia.org/wiki/Glutamic_acid) |
| F | [Phenylalanin](https://en.wikipedia.org/wiki/Phenylalanine" \o "Phenylalanine)ă |
| G | [Glycin](https://en.wikipedia.org/wiki/Glycine" \o "Glycine)ă |
| H | [Histidin](https://en.wikipedia.org/wiki/Histidine" \o "Histidine)ă |
| I | [Isoleucin](https://en.wikipedia.org/wiki/Isoleucine" \o "Isoleucine)ă |
| J | Leucină (L) sau Isoleucină (I) |
| K | [Lysin](https://en.wikipedia.org/wiki/Lysine" \o "Lysine)ă |
| L | [Leucin](https://en.wikipedia.org/wiki/Leucine" \o "Leucine)ă |
| M | [Methionin](https://en.wikipedia.org/wiki/Methionine" \o "Methionine)ă |
| N | [Asparagin](https://en.wikipedia.org/wiki/Asparagine" \o "Asparagine)ă |
| O | [Pyrrolysin](https://en.wikipedia.org/wiki/Pyrrolysine" \o "Pyrrolysine)ă |
| P | [Prolin](https://en.wikipedia.org/wiki/Proline" \o "Proline)ă |
| Q | [Glutamin](https://en.wikipedia.org/wiki/Glutamine" \o "Glutamine)ă |
| R | [Arginin](https://en.wikipedia.org/wiki/Arginine" \o "Arginine)ă |
| S | [Serin](https://en.wikipedia.org/wiki/Serine" \o "Serine)ă |
| T | [Threonin](https://en.wikipedia.org/wiki/Threonine" \o "Threonine)ă |
| U | [Selenocystein](https://en.wikipedia.org/wiki/Selenocysteine" \o "Selenocysteine)ă |
| V | [Valin](https://en.wikipedia.org/wiki/Valine" \o "Valine)ă |
| W | [Tryptophan](https://en.wikipedia.org/wiki/Tryptophan" \o "Tryptophan) |
| Y | [Tyrosin](https://en.wikipedia.org/wiki/Tyrosine" \o "Tyrosine)ă |
| Z | Acid Glutamic (E) sau Glutamină (Q) |
| X | Orice |
| \* | Stop translație |
| - | Spațiu de lungime nedeterminată |

**Tabelul B.2** Codificarea proteinelor în formatul FASTA

Anexa C

function annStorage = generateNeuralNetwork(networkName, networkType, codification, pdbCifFolder, pdbCifFiles, numInputs, noOfHiddenNeurons, trainingFunction, dataLimitsArgs, aditionalResourcesArgs, crossover, residueIndexes, coordinateIndexes)

%returneaza o structura care contine reteaua neurala si alte date

keys = {'Levenberg-Marquardt','BFGS Quasi-Newton','Resilient Backpropagation','Scaled Conjugate Gradient','Conjugate Gradient with Powell/Beale Restarts','Fletcher-Powell Conjugate Gradient','Polak-Ribiére Conjugate Gradient','One Step Secant','Variable Learning Rate Gradient Descent','Gradient Descent with Momentum','Gradient Descent','Bayesian Regularization'};

values = {'trainlm','trainbfg','trainrp','trainscg','traincgb','traincgf','traincgp','trainoss','traingdx','traingdm','traingd','trainbr'};

trainingFunctionsMap = containers.Map(keys, values);

residueAdmisibleNames = {'ALA','ARG','ASN','ASP','CYS','GLN','GLU','GLY','HIS','ILE','LEU','LYS','MET','PHE','PRO','SER','THR','TRP','TYR','VAL','ASX','GLX','XAA','END','GAP'};

if(strcmp(networkType, 'Feedforward Neural Network'))

reccurent = 0;

end

if(strcmp(networkType, 'Reccurent Neural Network'))

reccurent = 1;

end

ann = createMultiOutputAnn(3, noOfHiddenNeurons, trainingFunctionsMap(trainingFunction), reccurent);

ann.divideParam.trainRatio = dataLimitsArgs(1)/100;

ann.divideParam.valRatio = dataLimitsArgs(2)/100 - dataLimitsArgs(1)/100;

ann.divideParam.testRatio = 1 - dataLimitsArgs(2)/100;

annStorage = struct('ANN', {}, 'TR', {}, 'Codification', [], 'NetworkName', [], 'NetworkType', [], 'Crossover', [], 'TrainingFunction', [], 'PlotData', []);

if(aditionalResourcesArgs(1))

useParallel = 'yes';

poolObj = parpool;

else

useParallel = 'no';

end

if(aditionalResourcesArgs(2))

useGpu = 'yes';

ann.inputs{1}.processFcns = {'mapminmax'};

else

useGpu = 'no';

end

totalResidueData = [];

totalCoordinateData = cell(length(coordinateIndexes), 1);

for i=1:length(pdbCifFiles)

[atomData, hetatomData] = readData(strcat(pdbCifFolder, '\', pdbCifFiles{i}));

dataSize = size(atomData);

%verificare ca atomData contine aminoacizi si nu altceva

membershipArray = ismember(upper(atomData(:,residueIndexes)), residueAdmisibleNames);

if(any(membershipArray == 0) || dataSize(1) > 4000)

continue;

end

chainNames = unique(atomData(:,7));

startOfChain = 1;

%impartire in lanturi

for j=1:length(chainNames)

endOfChain = min(startOfChain + length(atomData(find(strcmp(atomData(:,7), chainNames(j))),:)) - 1, dataSize(1));

[residueData, coordinateData] = formatPdbData(atomData(startOfChain:endOfChain,:), numInputs, residueIndexes, coordinateIndexes, crossover);

totalResidueData = [totalResidueData residueData];

for k=1:length(coordinateIndexes)

totalCoordinateData{k} = [totalCoordinateData{k} coordinateData{k}];

end

startOfChain = endOfChain + 1;

end

totalDataSize = size(totalResidueData);

if(reccurent == 0 && totalDataSize(2) >= 10)

[ann tr] = train(ann, totalResidueData, totalCoordinateData, 'useParallel', useParallel, 'useGPU', useGpu);

totalResidueData = [];

totalCoordinateData = cell(length(coordinateIndexes), 1);

end

if(reccurent == 1)

seqTotalResidueData = con2seq(totalResidueData);

seqtotalCoordinateData = con2seq(totalCoordinateData);

totalResidueData = [];

totalCoordinateData = cell(length(coordinateIndexes), 1);

for j=1:length(seqTotalResidueData)

[ann tr] = train(ann, seqTotalResidueData(j), seqtotalCoordinateData(:,j), 'useParallel', useParallel, 'useGPU', useGpu);

ann.inputConnect(1, 1) = 1;

end

end

end

tempAnnStorage = struct('ANN', ann, 'TR', tr, 'Codification', codification, 'NetworkName', networkName, 'NetworkType', networkType, 'Crossover', crossover, 'TrainingFunction', trainingFunction, 'PlotData', []);

annStorage = [annStorage; tempAnnStorage];

if(aditionalResourcesArgs(1))

delete(poolObj);

end

end

**Figura C.1** Funcția care antrenează o nouă rețea neurală

function [residues, formatedCoordinates] = formatPdbData(data, blockSize, residueIndex, coordinatesIndexes, crossover)

indexInterval = blockSize - crossover;

residues = NaN(blockSize, floor(length(data)/indexInterval));

formatedCoordinates = cell(length(coordinatesIndexes), 1);

for j=1:length(coordinatesIndexes)

formatedCoordinates{j} = NaN(blockSize, floor(length(data)/indexInterval));

end

for i=1:floor(length(data)/indexInterval)

for j=1:length(coordinatesIndexes)

dataBlock = data((i-1)\*indexInterval+1 : min((i-1)\*indexInterval+blockSize, length(data)), coordinatesIndexes(j));

formatedCoordinates{j}(1:length(dataBlock),i) = cellfun(@(c) str2double(c), dataBlock);

end

%aminolookup nu functioneaza pe vector asa ca este plicata pe fiecare element al vectorului

for k=1:min(blockSize, length(data)-(i-1)\*indexInterval)

residues(k,i) = aa2intCustom(aminolookup(data{(i-1)\*indexInterval+k, residueIndex}));

end

end

end

**Figura C.2** Funcția care procesează datele de antrenare cu coordonate

totalCoordinateData = {

[12.900 12.247 -27.567 -15.062 12.900 12.247 -27.567 12.247;

12.900 12.247 -27.567 -15.062 12.900 12.247 -27.567 12.247 ;

12.900 -19.868 -27.567 -15.062 12.900 3.797 12.247 12.247 ;

12.900 -19.868 -27.567 NaN 12.900 3.797 12.247 12.247 ;

12.247 -19.868 12.900 NaN 12.247 3.797 12.247 12.247 ;

12.247 -19.868 12.900 NaN 12.247 3.797 12.247 12.247 ;

12.247 -19.868 12.900 NaN 12.247 3.797 12.247 NaN ;

12.247 -19.868 12.900 NaN 12.247 3.797 12.247 NaN ;

12.247 -27.567 -15.062 NaN 12.247 -27.567 12.247 NaN ]

;

[-18.112 -8.538 -27.567 -15.062 -18.112 -8.538 -27.567 -8.538;

-18.112 -8.538 -27.567 -15.062 -18.112 -8.538 -27.567 -8.538 ;

-18.112 -19.868 -27.567 -15.062 -18.112 -32.83 -8.538 -8.538 ;

-18.112 -19.868 -27.567 NaN -18.112 -32.83 -8.538 -8.538 ;

-8.538 -19.868 -18.112 NaN -8.538 -32.83 -8.538 -8.538 ;

-8.538 -19.868 -18.112 NaN -8.538 -32.83 -8.538 -8.538 ;

-8.538 -19.868 -18.112 NaN -8.538 -32.83 -8.538 NaN ;

-8.538 -19.868 -18.112 NaN -8.538 -32.83 -8.538 NaN ;

-8.538 -27.567 -15.062 NaN -8.538 -27.567 -8.538 NaN ]

;

37.484 12.247 -27.567 -15.062 37.484 12.247 -27.567 12.247;

37.484 12.247 -27.567 -15.062 37.484 12.247 -27.567 12.247 ;

37.484 25.621 -27.567 -15.062 37.484 3.797 12.247 12.247 ;

37.484 25.621 -27.567 NaN 37.484 3.797 12.247 12.247 ;

12.247 25.621 37.484 NaN 12.247 3.797 12.247 12.247 ;

12.247 25.621 37.484 NaN 12.247 3.797 12.247 12.247 ;

12.247 25.621 37.484 NaN 12.247 3.797 12.247 NaN ;

12.247 25.621 37.484 NaN 12.247 3.797 12.247 NaN ;

12.247 -27.567 -15.062 NaN 12.247 -27.567 12.247 NaN ]

}

Figura C.3 Vectorul de celule care conține coordonatele X, Y și Z în trei matrici.

function useANN(ann, fastaData, outputFilepath, blockSize, crossover, residueIndex)

if ~exist('crossover', 'var')

crossover = 0;

end

pdbIndexesOfColumns = [1, 8, 14, 16, 22, 24, 28, 30, 32, 36, 38, 46, 54, 62, 67, 73, 75, 80, 84, 86, 92];

startOfChain = 1;

indexInterval = blockSize - crossover;

formatedData = formatFastaData(fastaData);

chainNames = unique(formatedData(:,7));

fileID = fopen(outputFilepath, 'w');

for i=1:length(chainNames)

chainLength = length(formatedData(find(strcmp(formatedData(:,7), chainNames(i))),:));

endOfChain = startOfChain + chainLength - 1;

residues = NaN(blockSize, 1);

% j este indicele blocului din lantul i: startOfChain -> endOfChain

for j=1:floor(chainLength/(indexInterval))

% k este indicele fiecarui atom din blocul j

for k=1:min(blockSize, chainLength - (j-1)\*blockSize)

residues(k) = aa2intCustom(aminolookup(formatedData{(j-1)\*indexInterval+k, residueIndex}));

end

coordinates = ann(residues);

for k=1:min(blockSize, chainLength - (j-1)\*blockSize)

textLine = char(ones(1,92) \* ' ');

curentLineInFile = startOfChain + (j-1)\*blockSize + k - 1;

formatedData{curentLineInFile, 11} = num2str(round(coordinates(3\*(k-1) + 1), 3));

formatedData{curentLineInFile, 12} = num2str(round(coordinates(3\*(k-1) + 2), 3));

formatedData{curentLineInFile, 13} = num2str(round(coordinates(3\*(k-1) + 3), 3));

for m=1:length(pdbIndexesOfColumns)

textLine(pdbIndexesOfColumns(m):pdbIndexesOfColumns(m) + length(formatedData{curentLineInFile, m}) - 1) = formatedData{curentLineInFile, m};

end

fprintf(fileID, '%s\n', textLine);

end

end

startOfChain = endOfChain + 1;

end

fclose(fileID);

end

**Figura C.4** Codul care aplică o rețea neurală deja antrenată asupra unor fișiere .fasta

function formatedOutput = formatFastaData(fastaData)

%Transforma o serie de secvente Fasta intr-un vector de celule echivalente

%seriei de atomi dintrun fisier PDBx/mmCIF

keys = {'Ala','Arg','Asn','Asp','Cys','Gln','Glu','Gly','His','Ile','Leu','Lys','Met','Phe','Pro','Ser','Thr','Trp','Tyr','Val','Asx','Glx','Xaa'};

values = {{'N', 'C', 'C', 'O', 'C', ; 'N', 'CA', 'C', 'O', 'CB' },

{'N', 'C', 'C', 'O', 'C', 'C', 'C', 'N', 'C', 'N', 'N', ; 'N', 'CA', 'C', 'O', 'CB', 'CG', 'CD', 'NE', 'CZ', 'NH1', 'NH2' },

...

};

%Fiecare reziduu are o structura in formatul PDB care contine atomii

%coloanei sale vertebrale. Corespondenta reziduu-coloana este retinuta

%in dictionarul aminoacidMap

aminoacidMap = containers.Map(keys, values);

formatedOutput = {};

chainSymbol = 65;

atomIndex = 1;

%Pentru fiecare reziduu din fisierul .fasta, un numar de linii este

%scris in fisierul PDB, unic pentru fiecare reziduu

for i=1:length(fastaData)

chainResidueIndex = 1;

for j=1:length(fastaData(i).Sequence)

aminoacidStructure = aminoacidMap(aminolookup(fastaData(i).Sequence(j)));

for k=1:length(aminoacidStructure)

formatedOutput{end+1, 1} = 'ATOM';

formatedOutput{end, 2} = num2str(atomIndex);

formatedOutput{end, 3} = aminoacidStructure{1,k};

formatedOutput{end, 4} = aminoacidStructure{2,k};

formatedOutput{end, 5} = '.';

formatedOutput{end, 6} = upper(aminolookup(fastaData(i).Sequence(j)));

formatedOutput{end, 7} = char(chainSymbol);

formatedOutput{end, 8} = '1';

formatedOutput{end, 9} = num2str(chainResidueIndex);

formatedOutput{end, 10} = '?';

formatedOutput{end, 11} = '0.0';

formatedOutput{end, 12} = '0.0';

formatedOutput{end, 13} = '0.0';

formatedOutput{end, 14} = '1.00';

formatedOutput{end, 15} = '30.00';

formatedOutput{end, 16} = '?';

formatedOutput{end, 17} = num2str(chainResidueIndex);

formatedOutput{end, 18} = formatedOutput{end, 6};

formatedOutput{end, 19} = char(chainSymbol);

formatedOutput{end, 20} = aminoacidStructure{2,k};

formatedOutput{end, 21} = '1';

atomIndex = atomIndex + 1;

end

chainResidueIndex = chainResidueIndex + 1;

end

chainSymbol = chainSymbol + 1;

end

end

**Figura C.5** Funcția formatFastaData procesează datele fișierelor .fasta pentru a fi utilizate de o rețea neurală.

**Bibliografie**

[1] S. Lawrence, “Biotech drug market steadily expands,” *Nat. Biotechnol.*, vol. 23, p. 1466, Dec. 2005.

[2] G. Walsh, “Biopharmaceutical benchmarks 2006,” *Nat. Biotechnol.*, vol. 24, p. 769, Jul. 2006.

[3] J. M. Kwasigroch, D. Gilis, Y. Dehouck, and M. Rooman, “in Protein Structures,” vol. 18, no. 12, pp. 1701–1702, 2002.

[4] K. P. Pitera JW, “Exhaustive mutagenesis in silico: multicoordinate free energy calculations on proteins and peptides.,” *Proteins*, no. 41, pp. 385–397, 2000.

[5] M. Jalali-Heravi and M. Asadollahi-Baboli, *Quantitative structure-activity relationship study of serotonin (5-HT7) receptor inhibitors using modified ant colony algorithm and adaptive neuro-fuzzy interference system (ANFIS)*, vol. 44. 2009.

[6] R. Guha, D. T. Stanton, and P. C. Jurs, “Interpreting Computational Neural Network Quantitative Structure−Activity Relationship Models:  A Detailed Interpretation of the Weights and Biases,” *J. Chem. Inf. Model.*, vol. 45, no. 4, pp. 1109–1121, Jul. 2005.

[7] J. Lara, R. M. Wohlhueter, Z. Dimitrova, and Y. E. Khudyakov, “Artificial neural network for prediction of antigenic activity for a major conformational epitope in the hepatitis C virus NS3 protein,” *Bioinformatics*, vol. 24, no. 17, pp. 1858–1864, 2008.

[8] S. Lin *et al.*, “Design of Novel Conformational and Genotype-Specific Antigens for Improving Sensitivity of Immunoassays for Hepatitis C Virus-Specific Antibodies,” *J. Clin. Microbiol.*, vol. 43, no. 8, pp. 3917–3924, Aug. 2005.

[9] L. Michielan, L. Terfloth, J. Gasteiger, and S. Moro, “Comparison of Multilabel and Single-Label Classification Applied to the Prediction of the Isoform Specificity of Cytochrome P450 Substrates,” *J. Chem. Inf. Model.*, vol. 49, no. 11, pp. 2588–2605, Nov. 2009.

[10] G. D. Chou, P.Y. and Fasman, “Prediction of the secondary structure of proteins from their amino acid sequence.,” *Biol.*, no. 47, pp. 45–148, 1978.

[11] J. Garnier, D. J. Osguthorpe, and B. Robson, “Analysis of the accuracy and implications of simple methods for predicting the secondary structure of globular proteins,” *J. Mol. Biol.*, vol. 120, no. 1, pp. 97–120, 1978.

[12] V. Brendel, P. Bucher, I. R. Nourbakhsh, B. E. Blaisdell, and S. Karlin, “Methods and algorithms for statistical analysis of protein sequences,” *Proc. Natl. Acad. Sci. U. S. A.*, vol. 89, no. 6, p. 2002—2006, Mar. 1992.

[13] P. Gianluca, B. Pierre, F. Pietro, and C. Rita, “Prediction of coordination number and relative solvent accessibility in proteins,” *Proteins Struct. Funct. Bioinforma.*, vol. 47, no. 2, pp. 142–153, Mar. 2002.

[14] K. Nakai, A. Kidera, and M. Kanehisa, “Cluster analysis of amino acid indices for prediction of protein structure and function.,” *Protein Eng*, vol. 2, no. 2, pp. 93–100, 1988.

[15] O. Schueler-Furman and D. Baker, “Conserved residue clustering and protein structure prediction,” *Proteins Struct. Funct. Genet.*, vol. 52, no. 2, pp. 225–235, 2003.

[16] B. Rost and C. Sander, *Rost, B. & Sander, C. Improved prediction of protein secondary structure by use of sequence profiles and neural networks. Proc. Natl. Acad. Sci. USA 90, 7558-7562*, vol. 90. 1993.

[17] N. Qian and T. J. Sejnowski, “Predicting the secondary structure of globular proteins using neural network models.,” *J. Mol. Biol.*, vol. 202, no. 4, pp. 865–84, Aug. 1988.

[18] M. K. Gill, T. Asefa, Y. Kaheil, and M. McKee, “Effect of missing data on performance of learning algorithms for hydrologic predictions: Implications to an imputation technique,” *Water Resour. Res.*, vol. 43, no. 7, pp. 1–12, 2007.

[19] D. T. Jones, “Protein secondary structure prediction based on position-specific scoring matrices,” *J. Mol. Biol.*, vol. 292, pp. 195–202, 1999.

[20] G. Pollastri and A. McLysaght, “Porter: A new, accurate server for protein secondary structure prediction,” *Bioinformatics*, vol. 21, no. 8, pp. 1719–1720, 2005.

[21] T. J. P. Hubbard, B. Ailey, S. E. Brenner, A. G. Murzin, and C. Chothia, “SCOP: A structural classification of proteins database,” *Nucleic Acids Res.*, vol. 27, no. 1, pp. 254–256, 1999.

[22] W. S. Noble, R. Kuang, C. Leslie, and J. Weston, “Identifying remote protein homologs by network propagation,” *FEBS J.*, vol. 272, no. 20, pp. 5119–5128, 2005.

[23] P. R. Kensche, V. Van Noort, B. E. Dutilh, and M. A. Huynen, “Practical and theoretical advances in predicting the function of a protein by its phylogenetic distribution,” *J. R. Soc. Interface*, vol. 5, no. 19, pp. 151–170, 2008.

[24] R. Horton, L. A. Moran, D. Rawn, G. Scrimgeour, and M. Perry, “Principles of Biochemistry,” p. 824, 2011.

[25] U. O. F. Pure, “Atomic Weights of the Elements 1993,” *J. Phys. Chem. Ref. Data*, vol. 24, no. 4, pp. 1561–1576, 1995.

[26] Y. E. Khudyakov, *Medical Protein Engineering*. CRC Press, 2009.

[27] M. P. Fávero-Retto, L. C. Palmieri, T. A. C. B. Souza, F. C. L. Almeida, and L. M. T. R. Lima, “Structural meta-analysis of regular human insulin in pharmaceutical formulations,” *Eur. J. Pharm. Biopharm.*, vol. 85, no. 3, pp. 1112–1121, Nov. 2013.

[28] Nigel J. Dimmock; Andrew J. Easton; Keith N. Leppard, *Introduction to Modern Virology*, Seventh Ed. Chichester: John Wiley & Sons, Lth, 2016.

[29] G. Hanrahan, *Artificial Neural Networks in Biological and Environmental Analysis*. 2011.

[30] G. F. Luger, *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*, vol. 5th. 2005.

[31] M. T. Hagan, H. B. Demuth, and M. H. Beale, “Neural Network Design,” *Bost. Massachusetts PWS*, vol. 2, p. 734, 1995.

[32] G. Deyfus, *Neural Networks*, 1st edn. Berlin/Heidelberg: Springer-Verlag, 2005.

[33] K. Pelckmans, J. De Brabanter, J. A. K. Suykens, and B. De Moor, “Handling missing values in support vector machine classifiers,” *Neural Networks*, vol. 18, no. 5–6, pp. 684–692, 2005.

[34] R. May, H. Maier, G. Dandy, and T. M. K. G. Fernando, *Non-linear variable selection for artificial neural networks using partial mutual information*, vol. 23. 2008.

[35] C. Buiu, M. Putz, and S. Avram, “Learning the Relationship between the Primary Structure of HIV Envelope Glycoproteins and Neutralization Activity of Particular Antibodies by Using Artificial Neural Networks,” *Int. J. Mol. Sci.*, vol. 17, no. 10, p. 1710, Oct. 2016.

[36] “H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, P.E. Bourne. (2000) The Protein Data Bank Nucleic Acids Research, 28: 235-242.”