



UNIVERSITATEA „VASILE ALECSANDRI” DIN  
BACĂU

Facultatea de Științe

Str. Calea Mărășești, nr. 157, Bacău, 600115  
Tel. ++40-234-542411, tel./ fax ++40-234-571012  
[www.ub.ro](http://www.ub.ro); e-mail: [stiinte@ub.ro](mailto:stiinte@ub.ro)



## PROGRAMUL DE STUDII INFORMATICĂ-IFR

# LUCRARE DE LICENȚĂ

**Coordonator științific:**

Lector univ. dr.

Tomozei Cosmin

**Absolvent:**

Chelariu Andreea-Corina

**Bacău  
2025**



UNIVERSITATEA „VASILE ALECSANDRI” DIN  
BACĂU

Facultatea de Științe

Str. Calea Mărășești, nr. 157, Bacău, 600115  
Tel. ++40-234-542411, tel./ fax ++40-234-571012

[www.ub.ro](http://www.ub.ro); e-mail: [stiinte@ub.ro](mailto:stiinte@ub.ro)



## PROGRAMUL DE STUDII INFORMATICĂ-IFR

# LUCRARE DE LICENȚĂ

**Coordonator științific:**

Lector univ. dr.

Tomozei Cosmin

**Absolvent:**

Chelariu Andreea-Corina

**Bacău  
2025**



UNIVERSITATEA „VASILE ALECSANDRI” DIN  
BACĂU

Facultatea de Științe

Str. Calea Mărășești, nr. 157, Bacău, 600115  
Tel. ++40-234-542411, tel./ fax ++40-234-571012  
[www.ub.ro](http://www.ub.ro); e-mail: [stiinte@ub.ro](mailto:stiinte@ub.ro)



## PROGRAMUL DE STUDII INFORMATICĂ-IFR

# Rețele neuronale LSTM pentru predicție preț acțiuni.

**Coordonator științific:**  
**Lector univ. dr.**

Tomozei Cosmin

**Absolvent:**

Chelariu Andreea-Corina

**Bacău**  
**2025**

## CUPRINS

Obiectivul Aplicatiei.....	4
Introducere in preziceri de pret de actiuni .....	6
Implementare.....	8
Tipuri de retele -motivarea alegerii retelei.....	8
Librarii externe.....	9
Ordonarea datelor de intrare .....	12
LSTM.....	13
Normalizarea datelor .....	16
Initializarea greutatilor.....	17
Vectori de antrenament.....	17
Structura aplicatiei – design software .....	19
Testare si rezultate .....	38
Concluzii si directii de imbunatatiri .....	38
Bibliografie .....	40

## OBIECTIVUL APLICATIEI

În ultimii ani, inteligența artificială (AI) a devenit un subiect tot mai predominant în cercetarea științifică, reflectând interesul crescând al comunității academice pentru această tehnologie emergentă. Conform unui studiu realizat de CSIRO în noiembrie 2022[3], procentul lucrărilor peer-reviewed dedicate AI a crescut considerabil, ajungând la 5.7% din totalul publicațiilor științifice până în septembrie 2022, comparativ cu 3.1% în 2017 și doar 1.2% în 2000. Această tendință a fost evidentă și în 2020, înainte ca domeniul cercetării să se orienteze predominant spre COVID-19, când cinci dintre cele șapte cele mai influente lucrări de pe Google Scholar, conform metricilor de citare, au avut ca temă inteligența artificială, evidențiind astfel impactul și relevanța acestui domeniu în peisajul academic contemporan.

Câștigătorii granturilor ERC au fost invitați să anticipeze dezvoltarea AI în procesul științific până în 2030, evidențiind aplicațiile cheie și progresele semnificative. Majoritatea respondenților au concluzionat că AI va avea un rol determinant, fie ca instrument de sprijin, fie ca factor esențial în transformarea metodologiilor de cercetare. În anumite cazuri, AI nu doar că va optimiza procesul, ci va accelera, revoluționa și modifica fundamental structura investigării științifice. [2]

Unul dintre cele mai frecvente domenii de aplicare ale AI este analiza și procesarea datelor, unde tehnologia contribuie semnificativ la accelerarea cuantificării și vizualizării seturilor de date complexe și de mari dimensiuni. AI poate identifica corelații și modele subtile, imposibil de detectat prin cercetarea tradițională bazată pe ipoteze prestabilite.[2].

Conform unui studiu al OECD din 2024 [4], principalul risc al inteligenței artificiale este concentrarea excesivă a puterii tehnologice și economice în mâinile unui număr redus de companii, ceea ce poate limita competiția și accesul la inovație. Acest fenomen poate accentua inegalitățile economice, marginalizând firmele mai mici și instituțiile academice, și poate influența procesele politice prin subordonarea societății față de interesele marilor furnizori AI. În absența unor reglementări eficiente, AI ar putea deveni un instrument de control, afectând transparența decizională și reducând posibilitatea unor alegeri democratice informate.

Studiu al IMF din 2025 [5], analizează cum Inteligența Artificială va restructura piața muncii în perioada 2025-2030, conducând la transformarea sau înlocuirea a **39%** dintre competențele actuale ale angajaților. Se estimează că **59%** din forța de muncă va avea nevoie de recalificare,

iar **11%** dintre lucrători vor întâmpina dificultăți în obținerea pregătirii necesare, ceea ce le va reduce semnificativ șansele de angajare și mobilitate profesională.

Un studiu din 2025 al IMF[6] : Lucrătorii cu venituri ridicate sunt mai expuși la automatizarea prin AI, aproximativ 60% dintre ei având ocupații unde AI poate prelua multe sarcini, comparativ cu doar 15% dintre cei cu venituri mici. De asemenea, cei cu venituri mari obțin o parte mai mică din veniturile lor din salarii, dețin active financiare semnificative și investesc preponderent în active riscante, dar cu randament ridicat, precum acțiuni ale companiilor.

Inteligența artificială îmbunătățește domeniul financiar prin detectarea rapidă a fraudei, evaluarea precisă a riscurilor, optimizarea strategiilor de investiții, automatizarea proceselor de audit și cercetare fiscală, contribuind la eficiența operațiunilor și fundamentarea deciziilor bazate pe date complexe.

Spre exemplu, Danelfin folosește IA pentru optimizarea strategiilor de investiții printr-un sistem avansat de analiză care evaluează indicatori fundamentali, tehnici și de sentiment, generând **Danelfin AI Score**, un scor ce estimează probabilitatea ca o acțiune să depășească piața. Strategia **Danelfin Best Stocks** selectează acțiuni din cele mai performante cinci sectoare și le echilibrează trimestrial, menținând doar cele cu un scor AI ridicat și un nivel de risc scăzut, iar anual ajustează distribuția portofoliului pentru a maximiza randamentul. Testele retrospective arată o performanță superioară indicelui S&P 500, oferind protecție împotriva scăderilor și o abordare replicabilă pentru investitorii pe termen lung.

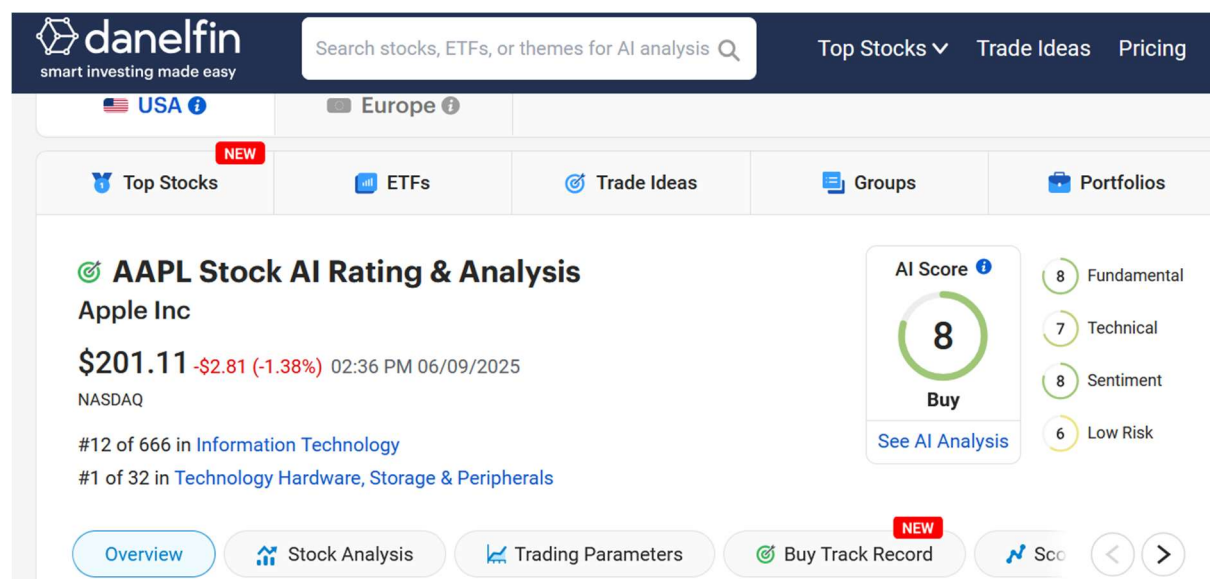
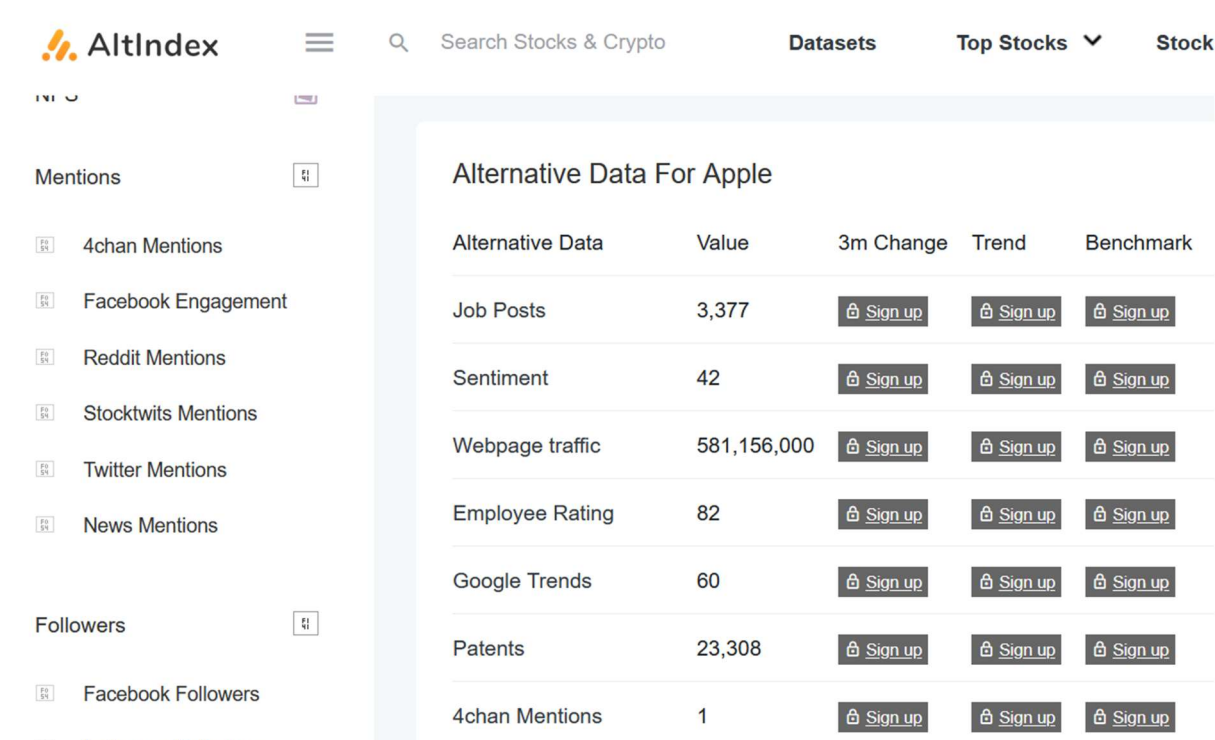


FIGURA 1 - DANELFIN

AltIndex e o alta firma care oferă tot pe web indexi alternativi si apoi cu o inteligenta artificiala oferă un scor de probabilitate sa performeze mai bine decât piața.



Alternative Data For Apple				
Alternative Data	Value	3m Change	Trend	Benchmark
Job Posts	3,377	<a href="#">Sign up</a>	<a href="#">Sign up</a>	<a href="#">Sign up</a>
Sentiment	42	<a href="#">Sign up</a>	<a href="#">Sign up</a>	<a href="#">Sign up</a>
Webpage traffic	581,156,000	<a href="#">Sign up</a>	<a href="#">Sign up</a>	<a href="#">Sign up</a>
Employee Rating	82	<a href="#">Sign up</a>	<a href="#">Sign up</a>	<a href="#">Sign up</a>
Google Trends	60	<a href="#">Sign up</a>	<a href="#">Sign up</a>	<a href="#">Sign up</a>
Patents	23,308	<a href="#">Sign up</a>	<a href="#">Sign up</a>	<a href="#">Sign up</a>
4chan Mentions	1	<a href="#">Sign up</a>	<a href="#">Sign up</a>	<a href="#">Sign up</a>

Comparativ cu Danelfin, există un număr mai mare de funcționalități care sunt restricționate și disponibile doar prin intermediul unui abonament plătit.

In aceasta lucrare ne propunem sa prezentam o aplicație care sa folosească rețele neuronale pentru a prezice prețul unei acțiuni alese.

Aplicația va prezenta utilizatorului doar opțiunea de alegere a acțiunii după ticker si apoi in spate sa poată face crearea, antrenarea si apoi prezicerea prețului acțiunii.

Scopul este de a pune la dispoziție si celor fără cunoștințe despre inteligenta artificiala sau chiar financiare a unui soft care sa ofere preziceri financiare prin folosirea de inteligenta artificiala care poate fi folosit de pe un calculator personal, ideal si de pe un telefon.

Ne dorim pentru început sa avem antrenarea si in final modelul antrenat pe dispozitivul utilizatorului îmbunătățind protectia datelor.

## INTRODUCERE IN PREZICERI DE PRET DE ACTIUNI

Estimarea viitoarei evoluții a prețurilor acțiunilor reprezintă o provocare considerabilă, având în vedere volatilitatea accentuată a piețelor și impactul unor factori imprevizibili, precum

modificările economice sau evenimentele globale. Cu toate acestea, aplicarea unor metode specifice poate contribui la fundamentarea unor decizii investiționale mai informate.

În analiza financiară se folosesc diverse analize pentru a putea face o decizie informată în investiții financiare.

Analiza tehnică utilizează datele istorice ale prețurilor și volumelor tranzacțiilor pentru a identifica modele și tendințe în evoluția pieței financiare. Această metodă se bazează pe premisa că modificările prețurilor urmează structuri recurente, ceea ce permite formularea de prognoze asupra direcției viitoare a activelor. Printre tehnicile frecvent aplicate în acest domeniu se numără utilizarea indicatorilor tehnici, cum ar fi media mobilă, indicele de forță relativă (RSI) și convergența-divergența mediilor mobile (MACD), care furnizează informații despre momentul și direcția trendurilor. De asemenea, analiza trendurilor și formațiunilor grafice, precum identificarea nivelurilor de suport și rezistență sau recunoașterea tiparelor geometrice, cum ar fi triunghiurile, contribuie la interpretarea evoluției prețurilor. În plus, modelele bazate pe analiza fractală și algoritmi predictivi sunt folosite pentru a extrage regularități din seriile de timp financiare, facilitând astfel anticiparea mișcărilor viitoare ale pieței. Această abordare oferă o bază empirică pentru luarea deciziilor investiționale, bazată pe observația și procesarea statistică a datelor disponibile.

Analiza fundamentală evaluează factori economici, financiari și de piață pentru a determina valoarea intrinsecă a unei acțiuni. Aceasta presupune examinarea indicatorilor financiari ai companiei, precum raportul preț-câștig (*P/E ratio*), rezultatul operațional înainte de dobânzi, taxe, deprecieri și amortizare (*EBITDA*), precum și fluxurile de numerar, care oferă o imagine asupra stabilității și sustenabilității financiare ale entității analizate. În paralel, studiul mediului economic și industrial permite identificarea condițiilor externe ce pot influența performanța companiei, inclusiv analiza sectorului în care aceasta operează și impactul competiției. Totodată, deciziile macroeconomice și politicile financiare adoptate la nivel guvernamental joacă un rol esențial în evoluția pieței, iar cercetările asupra acestora oferă perspective asupra riscurilor și oportunităților asociate investițiilor. Această metodă asigură o abordare cuprinzătoare în procesul de evaluare a activelor financiare.

Metodele bazate pe inteligență artificială și învățare automată utilizează algoritmi avansați pentru identificarea tiparelor complexe și corelațiilor existente în datele financiare. Rețelele neuronale artificiale permit modelarea relațiilor nelineare și optimizarea procesului de predicție prin ajustarea ponderilor interne. Modelele bazate pe regresie și clasificare sunt utilizate pentru



estimarea valorilor viitoare ale activelor financiare și pentru segmentarea datelor pe categorii relevante. Algoritmii de procesare a limbajului natural facilitează analiza știrilor și evaluarea sentimentului pieței, oferind o perspectivă asupra impactului evenimentelor externe asupra comportamentului investițional. Aceste abordări contribuie la creșterea preciziei deciziilor financiare și la îmbunătățirea strategiilor predictive.

În rețelele neuronale, prezicerea prețului acțiunilor este abordată ca o problemă de serie de timp. Modelul trebuie să identifice modul în care prețurile anterioare, alături de factori relevanți precum valoarea acțiunilor concurente sau costul materialelor, influențează evoluția viitoare a prețului.

## IMPLEMENTARE

### TIPURI DE REȚELE -MOTIVAREA ALEGERII REȚELEI

Pentru prezicere de serii de timp sunt mai multe tipuri de rețele care pot prezice serii de timp precum LSTM (Long short term memory), Transformers – Modele mai avansate, cum ar fi GPT și BERT, pot fi ajustate pentru analiza seriilor de timp, oferind capacitate superioară de înțelegere a dependențelor complexe.

Din rețele neuronale recurente (RNN) LSTM este utilizat în predicții bazate pe date secvențiale, cum ar fi prețurile acțiunilor, traducere automată, recunoașterea vorbirii și analiza sentimentului.

Transformers sunt antrenați pe platforme precum AWS, Google Cloud sau Azure, ceea ce adaugă costuri suplimentare, au nevoie de GPU pentru paralelizare și de aici nu se pretează pentru scopul acestei aplicații.

Alegerea pentru început e de a folosi LSTM și de a extinde cu alte metode în timp dacă e necesar.

### MOTIVAREA ALEGERII LIMBAJULUI DE PROGRAMARE

Implementarea logicii LSTM (Long Short-Term Memory) în C++ și a interfeței grafice în C# într-o aplicație reprezintă o alegere arhitecturală bine fundamentată, care combină eficiența performanțială cu flexibilitatea dezvoltării interfeței utilizatorului.

C++ este un limbaj de programare extrem de performant, care permite un control detaliat asupra resurselor hardware și al gestionării memoriei, fapt ce îl face ideal pentru implementarea unor

algoritmi complexi de învățare automată, precum LSTM. Rețelele neuronale de tip LSTM necesită un volum semnificativ de calcule numerice și o gestionare optimizată a resurselor pentru a putea procesa rapid volume mari de date, iar C++ excelează în această privință datorită vitezei de execuție și a capacității de a valorifica procesarea paralelă sau utilizarea GPU-urilor. Astfel, alegerea C++ pentru logica de procesare a modelului garantează o performanță ridicată în timpul antrenamentului și al inferenței modelului.

Pe de altă parte, C# este un limbaj care oferă un mediu de dezvoltare excelent pentru aplicațiile cu interfață grafică, datorită integrării sale strânse cu platformele .NET și a bibliotecilor specifice pentru crearea de interfețe vizuale sofisticate și interactive, cum ar fi Windows Forms sau WPF (Windows Presentation Foundation). C# permite dezvoltarea rapidă a unor aplicații cu un design ușor de utilizat, fiind un limbaj de programare accesibil, eficient și productiv pentru crearea interfețelor de utilizator. Acest aspect contribuie semnificativ la îmbunătățirea experienței utilizatorului final, oferind un mediu de interacțiune intuitiv și plăcut.

Prin separarea logicii de calcul intensiv din C++ de interfața grafică din C#, se asigură un design modular, care facilitează atât dezvoltarea, cât și întreținerea aplicației pe termen lung. Acest tip de arhitectură permite ca cele două componente – logica de procesare și interfața grafică – să poată fi dezvoltate, testate și optimizate independent, având astfel un impact minim asupra performanței generale a aplicației.

În ceea ce privește interoperabilitatea între cele două limbaje, există soluții tehnice, cum ar fi utilizarea unor biblioteci intermediare sau a wrapper-elor, care permit comunicarea eficientă între C++ și C#, fără a compromite performanța. Prin urmare, logica intensivă de calcul, implementată în C++, poate fi accesată din aplicația scrisă în C# pentru a oferi utilizatorului final o interfață grafică fluidă și interactivă, fără a afecta viteza de procesare a datelor.

## LIBRARII EXTERNE

ScottPlot reprezintă o bibliotecă de vizualizare a datelor destinată aplicațiilor dezvoltate în C# și .NET, care oferă funcționalități avansate pentru crearea și personalizarea graficelor. Aceasta permite generarea rapidă de grafice pentru diverse tipuri de date și este optimizată pentru a oferi performanțe ridicate, chiar și în cazul unor seturi mari de date.

Biblioteca se distinge prin integrarea sa ușoară cu platforma .NET, oferind un API simplu și eficient pentru crearea și manipularea graficelor. ScottPlot susține o gamă largă de tipuri de

grafice, inclusiv grafice liniare, de dispersie, histograme și grafice polar, permițând utilizatorilor să aleagă tipul de reprezentare adecvat necesităților aplicației. De asemenea, biblioteca oferă opțiuni detaliate de personalizare a graficelor, incluzând modificarea axelor, a culorilor, a etichetelor și a fonturilor, asigurând o flexibilitate considerabilă în procesul de vizualizare a datelor.

Un alt aspect relevant al ScottPlot este suportul pentru interactivitate. Utilizatorii pot interacționa direct cu graficele prin zoom, glisare și selecție a datelor, facilitând astfel analiza vizuală a acestora în timp real. De asemenea, ScottPlot permite actualizarea dinamică a graficelor, fiind adecvat pentru aplicații care necesită vizualizarea datelor în timp real, precum monitorizarea proceselor sau aplicațiile de analiza datelor provenite din senzori.

Biblioteca este, de asemenea, cunoscută pentru performanța sa în manipularea datelor, fiind capabilă să genereze grafice într-un interval de timp scurt, chiar și pentru seturi mari de date. Acest aspect este esențial pentru aplicațiile care necesită vizualizarea rapidă și eficientă a datelor.

ScottPlot beneficiază de o documentație detaliată, care facilitează integrarea sa în proiectele existente, iar fiind o bibliotecă open-source, aceasta permite comunității să contribuie și să îmbunătățească continuu funcționalitățile oferite. În plus, biblioteca este compatibilă cu diferite tipuri de aplicații dezvoltate în .NET, inclusiv cele care utilizează Windows Forms și WPF, facilitând integrarea acestora în interfețele grafice ale aplicațiilor.

În concluzie, ScottPlot este o opțiune eficientă pentru implementarea de funcționalități de vizualizare a datelor în aplicațiile C#, oferind performanță, interactivitate și flexibilitate în personalizarea graficelor.

**Alpha Vantage** este o platformă ce oferă o interfață de programare a aplicației (API - Application Programming Interface) care facilitează accesul la date financiare și economice. Serviciile sale sunt destinate cu precădere dezvoltatorilor, cercetătorilor, analiștilor cantitativi și instituțiilor academice, care necesită volume mari de date istorice și în timp real pentru analize, modelare predictivă și dezvoltare de aplicații.

Deși platforme precum Yahoo Finance sunt larg utilizate și recunoscute pentru accesibilitatea lor generală, acestea prezintă limitări semnificative din perspectiva integrării programatice, neoferind o interfață de programare a aplicației (API) dedicată. Soluțiile alternative, bazate pe **web scraping** sau parsarea manuală a datelor din interfețele web, introduc riscuri substanțiale

de instabilitate și mentenanță pe termen lung, dat fiind că depind de structura volatilă a paginilor web.

În vederea asigurării robusteții și predictibilității în accesul la date pe viitor, s-a optat pentru o soluție bazată pe API. Procesul de selecție s-a restrâns la evaluarea a două servicii principale: Alpha Vantage și Quandl (acum parte a Nasdaq Data Link), ambele oferind API-uri stabile și bine documentate.

**Alpha Vantage** este o alegere excelentă pentru acces rapid și cost-eficient la date financiare standard, potrivit pentru prototipuri și nevoi de bază. **Quandl (Nasdaq Data Link)**, pe de altă parte, este un furnizor de date mai scump, dar care oferă o diversitate și o calitate superioară a datelor, în special a celor alternative și specializate, ținând un segment de piață mai premium.

Visual Studio constituie o alegere adecvată pentru dezvoltarea aplicațiilor care utilizează atât C++, cât și C#, datorită facilității sale de utilizare și a capacității de a gestiona ambele limbaje într-un singur mediu de dezvoltare integrat. Această caracteristică permite dezvoltatorilor să lucreze într-un flux de lucru unificat, fără a fi necesar să comute între diferite medii de dezvoltare.

Prin suportul oferit pentru ambele limbaje, Visual Studio facilitează gestionarea proiectelor complexe ce implică atât componente de performanță înaltă, implementate în C++, cât și dezvoltarea interfeței grafice în C#. De asemenea, Visual Studio oferă funcționalități avansate, cum ar fi completarea automată a codului, depanarea integrată și instrumente de gestionare a proiectului, care simplifică procesul de dezvoltare, indiferent de limbajul utilizat.

Astfel, utilizarea Visual Studio contribuie la eficientizarea procesului de dezvoltare a aplicațiilor care integrează C++ și C#, oferind un mediu unificat și optimizat pentru realizarea acestora.

Alegerea **Windows Forms** în loc de **WPF** se bazează pe două motive principale: simplitatea interfeței grafice și portabilitatea.

În primul rând, Windows Forms oferă o abordare mai simplă și mai directă pentru dezvoltarea interfețelor grafice, fiind mai ușor de utilizat și mai rapid de implementat comparativ cu WPF, care implică o gestionare mai complexă a resurselor grafice, stilurilor și animațiilor. Această simplitate face ca Windows Forms să fie o alegere adecvată pentru aplicațiile care nu necesită

interfețe grafice avansate sau personalizate, permițând dezvoltarea rapidă și eficientă a funcționalităților vizuale de bază.

În al doilea rând, Windows Forms asigură o compatibilitate extinsă cu diverse versiuni ale sistemului de operare Windows, ceea ce sporește portabilitatea aplicațiilor dezvoltate pe această platformă. În comparație cu WPF, care este compatibilă doar cu versiunile mai recente de Windows, Windows Forms permite o acoperire mai largă a pieței, fiind potrivită pentru aplicațiile destinate unui public divers, incluzând utilizatori care lucrează cu versiuni mai vechi ale sistemului de operare.

Astfel, alegerea Windows Forms este justificată prin necesitatea unei interfețe grafice simple și a unei portabilități extinse pe multiple versiuni ale Windows.

## ORDONAREA DATELOR DE INTRARE

Ne propunem estimarea prețului viitor pe baza valorilor înregistrate în ultimele  $x$  zile, unde parametrul  $x$  este configurabil de către utilizator. În acest context, utilizarea rețelelor neuronale de tip *Long Short-Term Memory* (LSTM) prezintă avantajul de a gestiona date secvențiale, având capacitatea de a învăța din modelele temporale existente în succesiunea valorilor anterioare.

toate cele  $x$  valori sunt integrate simultan într-un singur vector de intrare, furnizat modelului într-o formă agregată. Alternativ, rețeaua poate fi rulată secvențial pe  $x$  valori consecutive, introducând fiecare observație individuală într-un proces iterativ. Ne dorim captarea mai eficientă a dependențelor temporale dintre valorile anterioare. Deoarece LSTM este conceput pentru a reține informația de-a lungul timpului și pentru a actualiza memoria în funcție de noi intrări, furnizarea valorilor în mod progresiv îi permite să identifice mai precis relațiile subtile dintre observații consecutive. Această abordare este deosebit de utilă în situațiile în care există pattern-uri care evoluează treptat și nu pot fi extrase corespunzător dintr-un singur vector static.

Rețelele neuronale sunt proiectate pentru a învăța relațiile dintre caracteristicile de intrare (*features*) și variabila țintă, ajustând parametrii modelului astfel încât să optimizeze procesul de predicție. Pentru aceasta aplicație ne alegem ca valorile precedente să fie folosite ca feature dar să dam posibilitatea să adăugăm și alte acțiuni ca feature.

## LSTM

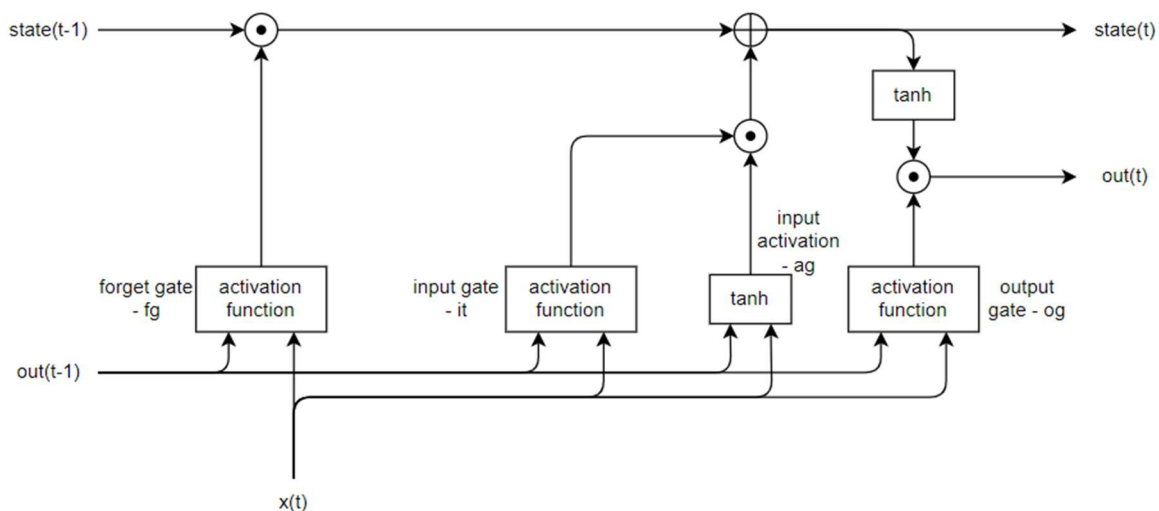


FIGURA 2 - LSTM GRAFIC

In acest document vom folosi structura LSTM cu notațiile de mai sus in care:

$X(t)$  – intrarea la momentul  $t$

$State(t)$  – starea celulei la momentul  $t$

$Out(t)$  – ieșirea

Cu porțile:

Fg – forget gate

Ig – input gate

Ag -input activation gate

⊙ - reprezinta produsul Hadamard pentru matrici

⊕ - reprezinta adunarea matriceala

Functia de activare sunt sigmoid, si tanh cu formulele:

$$\sigma = \frac{1}{1 + e^{-x}}$$

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

Graficele lor generate cu scilab:

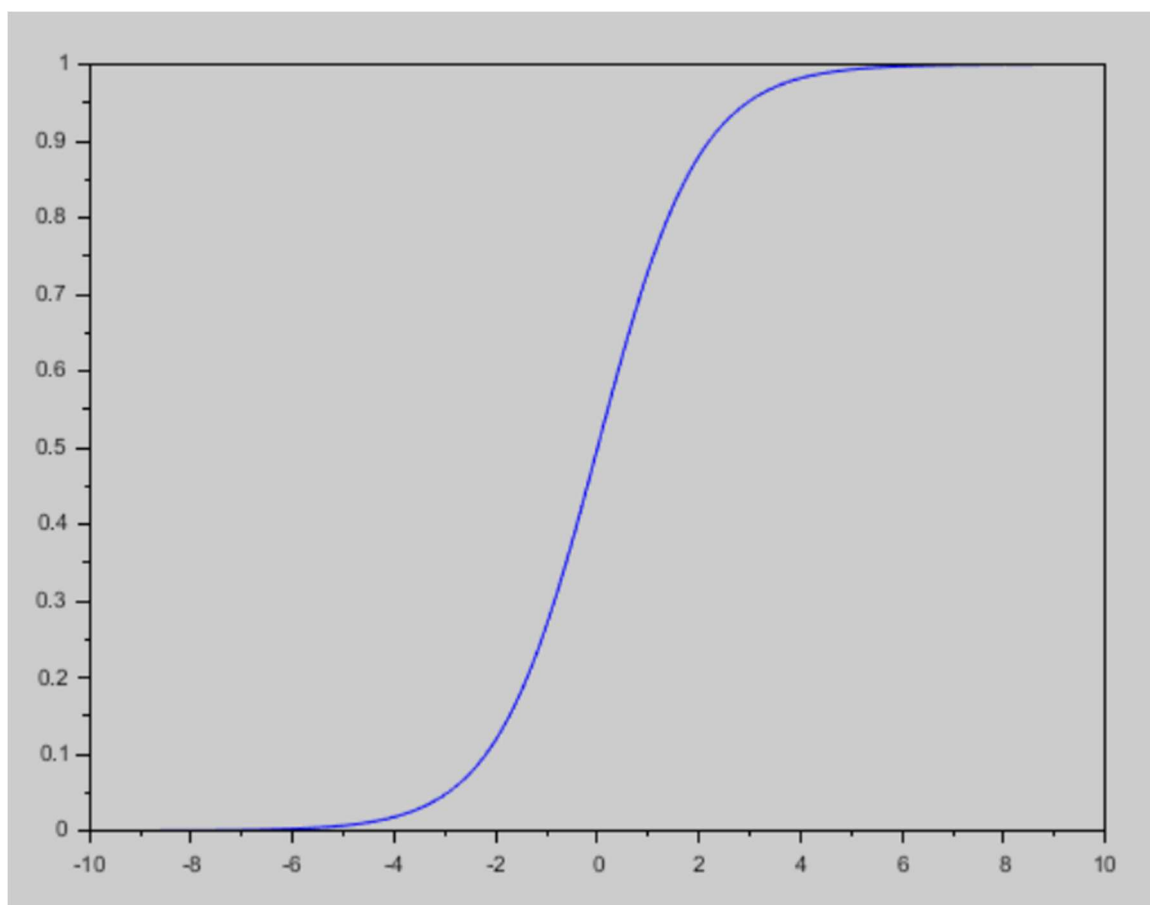


FIGURA 3-SIGMOID

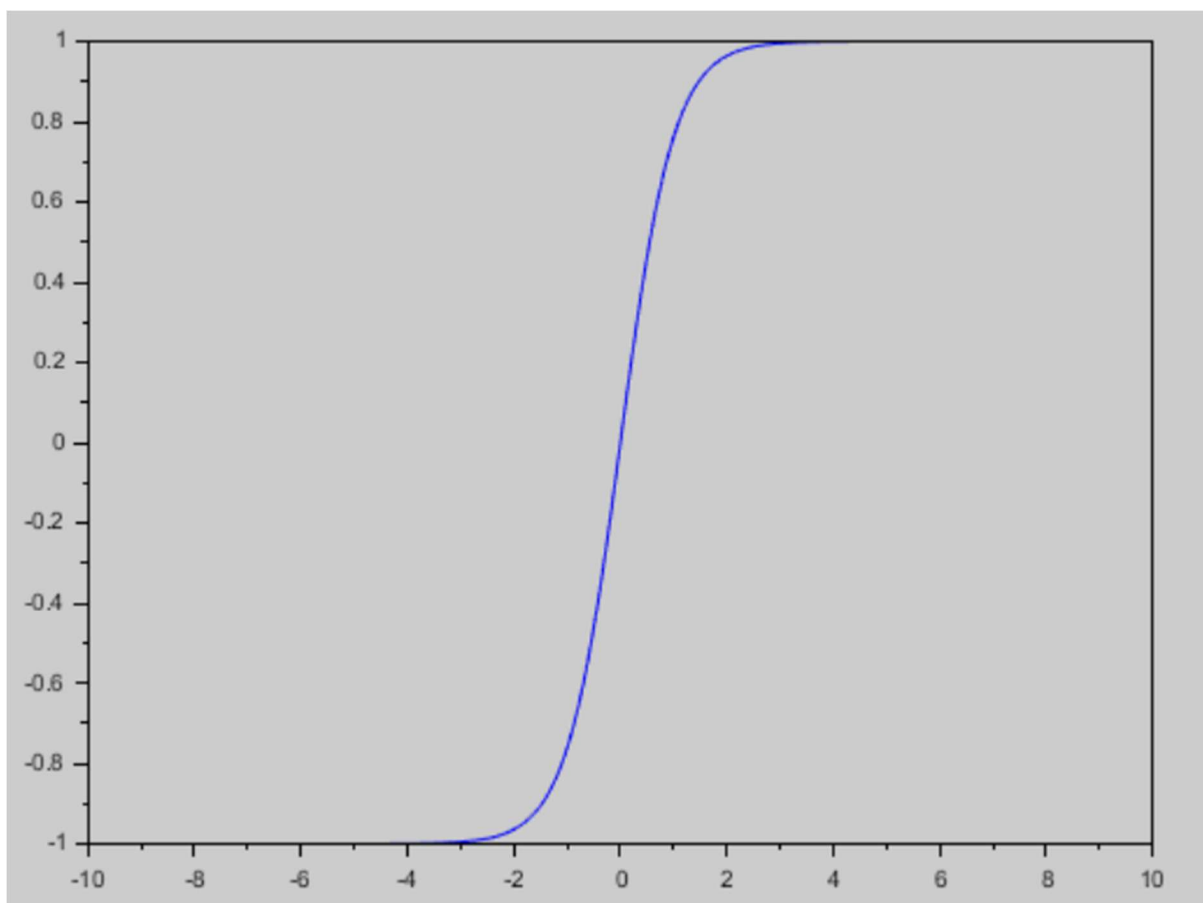


FIGURA 4 - TANH

Formulele pentru calcul al porților folosite conform [8]:

$$fg(t) = \sigma(W_f \cdot x(t) + U_f \cdot out(t-1) + b_f)$$

$$ig(t) = \sigma(W_i \cdot x(t) + U_i \cdot out(t-1) + b_i)$$

$$ag(t) = \sigma(W_a \cdot x(t) + U_a \cdot out(t-1) + b_a)$$

$$og(t) = \sigma(W_o \cdot x(t) + U_o \cdot out(t-1) + b_o)$$

$$state(t) = ag(t) \cdot ig(t) + fg(t) \cdot state(t-1)$$

$$out(t) = \tanh(state(t)) \odot og(t)$$

Acum pentru backpropagation formulele conform aceluiași articol sunt:

$$\delta out(t) = \Delta(t) + U_a^T \delta ag(t+1) + U_i^T \delta ig(t+1) + U_f^T \delta fg(t+1) + U_o^T \delta og(t+1)$$

$$\delta ogp(t) = \delta out(t) \odot \tanh(state(t)) \odot \sigma'(ogp(t))$$

$$\delta state(t) = \delta out(t) \odot og(t) \odot \tanh'(state(t)) + \delta state(t+1) \odot fg(t+1)$$



$$\delta fgp(t) = \delta state(t) \odot state(t-1) \odot \sigma'(fgp(t))$$

$$\delta igp(t) = \delta state(t) \odot ag(t) \odot \sigma'(igp(t))$$

$$\delta agp(t) = \delta state(t) \odot ig(t) \odot \tanh'(agp(t))$$

Aici am postfixat p la numele porții pentru a nota valoarea înainte de funcția de activare, astfel agp e doar adunarea  $W_a * x(t) + U_a * out(t-1) + b_a$ .

Gradienții pentru greutateți sunt calculați după formulele:

$$\delta W_* = \sum_{t=0}^T \langle \delta * (t), x(t) \rangle, \quad *= ag, ig, og, fg$$

$$\delta U_* = \sum_{t=0}^T \langle \delta * (t+1), out(t) \rangle, \quad *= ag, ig, og, fg$$

$$\delta b_* = \sum_{t=0}^T \delta * (t), \quad *= ag, ig, og, fg$$

Prin  $\langle a, b \rangle$  se notează produsul exterior sau tensorial.

Odată calculat gradientul, parametrii modelului sunt actualizați în direcția opusă gradientului pentru a reduce valoarea funcției de cost. Această actualizare se face cu un pas determinat de un factor de învățare, notat cu  $\lambda$ . Formula de actualizare a parametrilor este:

$$W = W - \lambda * \delta W$$

Unde  $W$  reprezintă orice parametru de greutate, iar  $\delta W$  indică gradientul asociat acestuia, conform formulelor de mai sus.

## NORMALIZAREA DATELOR

Normalizarea datelor de intrare influențează viteza de convergență, acuratețea predicțiilor și capacitatea modelului de a generaliza pe date noi. Procesul previne dominanța *features*-urilor cu valori mari, asigurând o distribuție echilibrată a datelor și facilitând propagarea semnalului prin rețea.

Randamentele financiare sunt adesea exprimate în termeni logaritmici pentru a reflecta modificările procentuale într-un mod simetric. Formula utilizată este:

$$r_t = \ln(P_t) - \ln(P_{t-1})$$

unde  $P_t$  reprezintă prețul activului la momentul  $t$ . Această metodă este preferată deoarece randamentele logaritmice sunt aditivabile pe perioade succesive.

Pornind de aici vom folosi o funcție de normalizare

$$f = \ln\left(\frac{P(t)}{P(t-1)}\right)$$

Cum logaritmul subunitar este negativ, această funcție ne permite să avem și sensul creșterii și dimensiunea creșterii de la o valoare la alta.

Pentru a obține valori înapoi vom folosi funcția inversă să îi spunem denormalizare:

$$P_t = e^f P_{t-1}$$

Pentru a obține din o valoare normalizată o valoare înapoi avem nevoie de ultima valoare din șirul de date de intrare.

## INITIALIZAREA GREUTĂȚILOR

Pentru inițializarea greutateților vom folosi propoziția de Xavier Glorot și Yoshua Bengio [1]. Avantajul metodei este de a evita problemele de tip vanishing sau exploding gradient.

Vom inițializa valorile tuturor greutateților cu o distribuție uniformă între

$$\pm \sqrt{\frac{6}{n_{in} + n_{out}}}$$

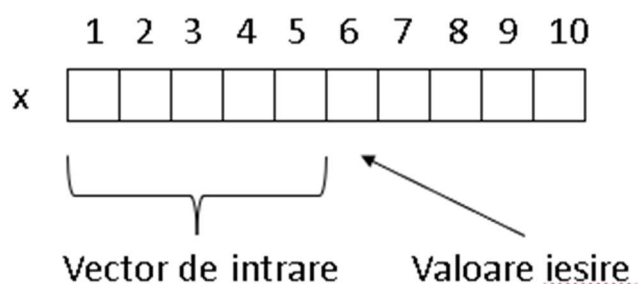
Unde  $n_{in}$  reprezintă numărul de features și  $n_{out}$  reprezintă numărul de ieșiri din rețea.

## VECTORI DE ANTRENAMENT

În construirea vectorilor de antrenament al rețelei ne va trebui să păstrăm o parte din date pentru testare. Astfel din 30% din serie de date de intrare, partea de final, o vom păstra pentru testare și nu o vom introduce în etapa de antrenament.

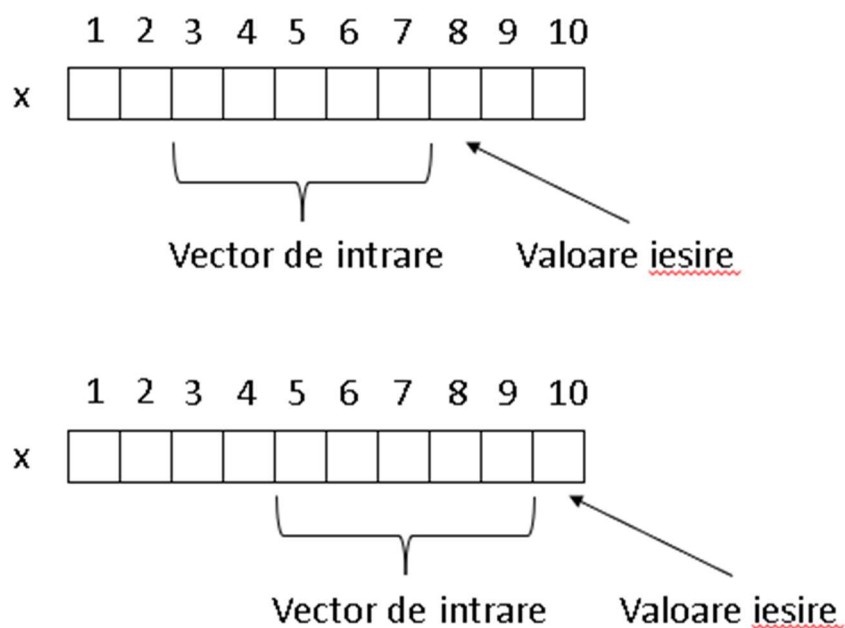
În esență vom dori să folosim ultimele  $n$  valori din istoricul prețului unei acțiuni pentru a prezice valoarea în viitor. Astfel vectorul de antrenament va fi alcătuit din  $n$  valori de intrare și o valoare de ieșire care trebuie să fie estimată.

Daca datele istorice vor fi un vector de valori  $x$  atunci putem sa imaginam construirea unui vector de antrenament in acest fel:



**FIGURA 5 - VECTOR DE ANTRENAMENT**

Pentru un vector de intrare  $x$ , se construiesc mai mulți vectori de antrenament astfel încât să acopere cât mai eficient spațiul datelor disponibile. Pentru a evita supraantrenarea rețelei neuronale, nu se generează toate combinațiile posibile de vectori de antrenament, ci se utilizează o metodă de eșantionare prin salt incremental. Alegerea unui pas determină punctul de start al fiecărui vector de antrenament, reducând redundanța și optimizând procesul de învățare. Ca exemplu pentru un pas egal cu 2, al doilea și al treilea vector de antrenament sunt structurați conform exemplului ilustrat în figura 6.

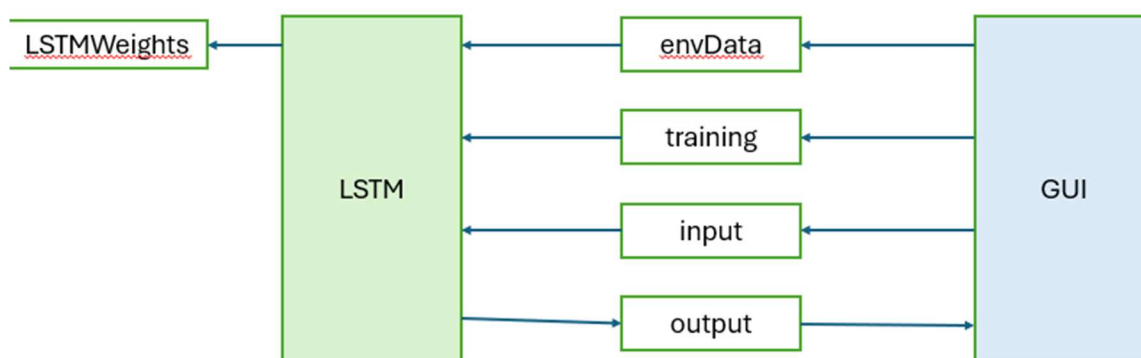


**FIGURA 6 - PAS DE ANTRENAMENT**

Setul de date  $x$  este împărțit astfel încât 70% dintre elemente sunt alocate pentru construirea vectorilor de antrenament, iar 30% sunt utilizate pentru evaluarea performanței modelului. După finalizarea procesului de antrenare pe setul de antrenament, rețeaua neuronală este testată folosind vectorii de test, care nu au fost incluși în etapa de antrenament. Această metodă permite verificarea capacității modelului de a generaliza pe date noi și asigură o evaluare obiectivă a performanței acestuia.

## STRUCTURA APLICATIEI – DESIGN SOFTWARE

Arhitectura aplicației este divizată în două module distincte. Primul modul, dezvoltat în C#, gestionează interfața cu utilizatorul (GUI) și include facilități de preprocesare și pregătire a datelor. Aceste date sunt apoi transmise modulului secundar, care implementează o rețea neuronală recurentă de tip LSTM (Long Short-Term Memory). Această componentă este optimizată specific pentru sarcina de predicție a prețurilor acțiunilor.



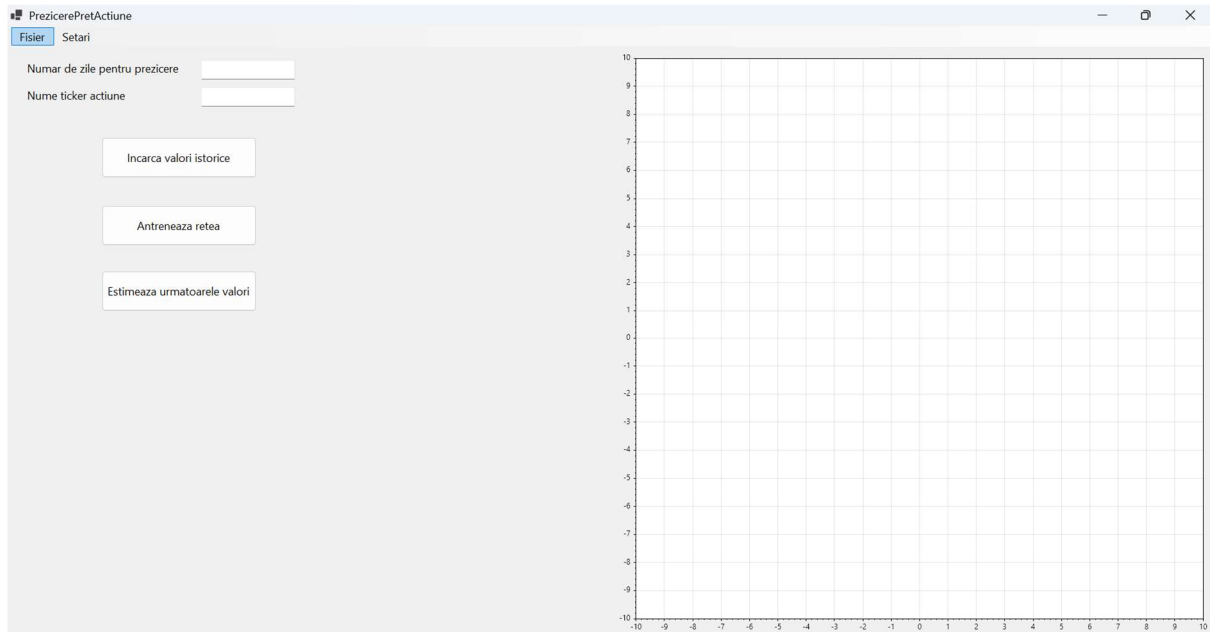
**FIGURA 7 - FLUXUL DE DATE DINTRE MODULE**

Figura 7 ilustrează schematic fluxul de date bidirecțional dintre cele două module ale aplicației. Astfel, modulul de interfață grafică (GUI) este responsabil pentru persistența datelor de configurare în structura `envData`, a seturilor de date de antrenament și a datelor de intrare necesare pentru procesul de estimare, toate acestea fiind ulterior accesate și procesate de modulul LSTM. Reciproc, modulul rețelei neuronale va stoca valorile ponderilor antrenate și rezultatul estimării în directorul `output`, de unde vor fi preluate și utilizate de către modulul principal al aplicației.

## INTERFATA PRINCIPALA

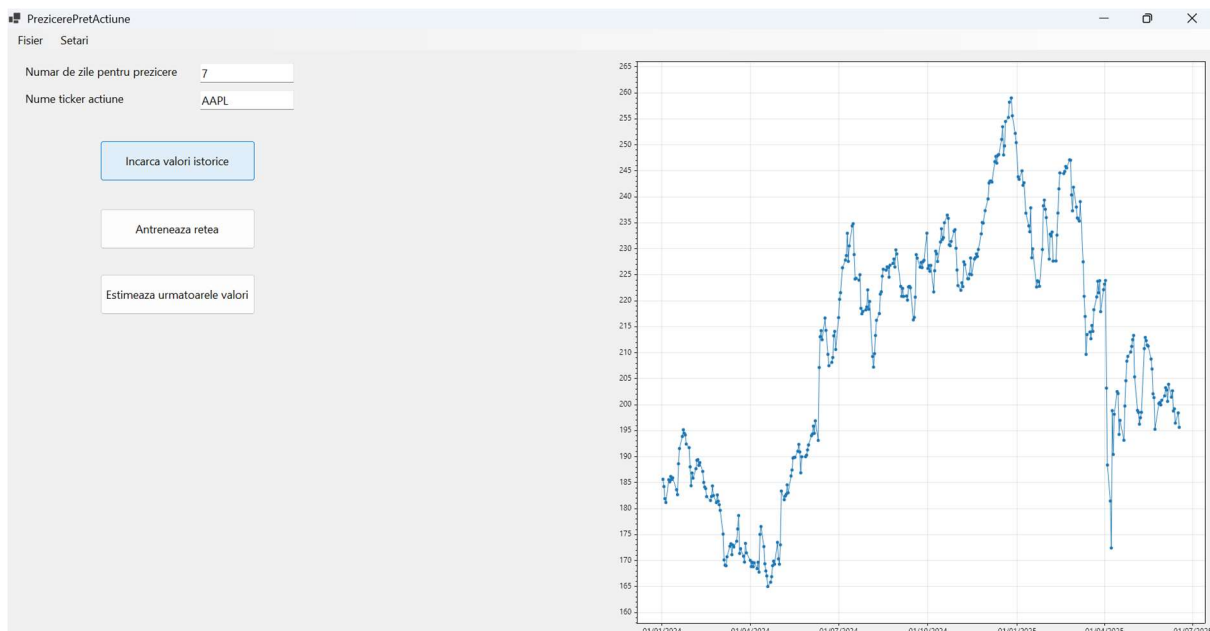
Interfața principală (Dashboard-ul) integrează un panou vizual central, inițial nepopulat, destinat afișării datelor grafice. Acest panou este însoțit de un set de trei butoane, care

corespund principalelor funcționalități disponibile în cadrul aplicației. Suplimentar, interfața include două câmpuri configurabile esențiale pentru personalizarea analizei: unul pentru specificarea numărului de zile anterioare ce vor fi utilizate în procesul de predicție, și celălalt pentru introducerea simbolului (ticker-ului) acțiunii a cărei valoare se dorește a fi estimată.



**FIGURA 8 - DASHBOARD INITIAL**

În figura 8 vedem cum prețul acțiunilor pe ultimul an e încărcat și afișat în graficul din dreapta.



**FIGURA 9 - AFISARE GRAFIC VALORI**

Pentru achiziția datelor istorice referitoare la prețurile acțiunilor, se utilizează interfața de programare a aplicației (API) oferită de Alpha Vantage.

În situația în care un simbol bursier pentru acțiune nu este selectat sau furnizat, utilizatorul este notificat în consecință.

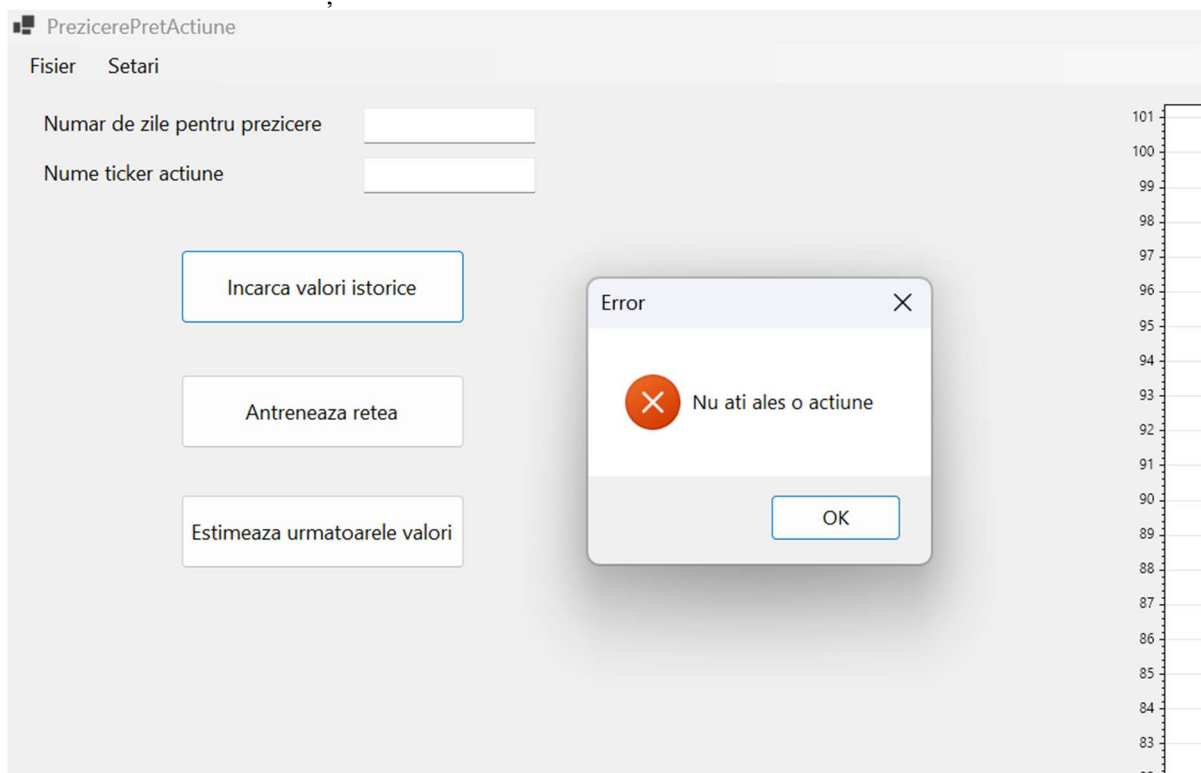


FIGURA 10 - ERROR - NICI UN TICKER ALES

La apăsarea butonului Antrenaza retea in cazul in care numărul de zile nu sunt alese suntem informați ca implicit va fi aleasa valoarea 5 (o săptămâna lucrătoare).

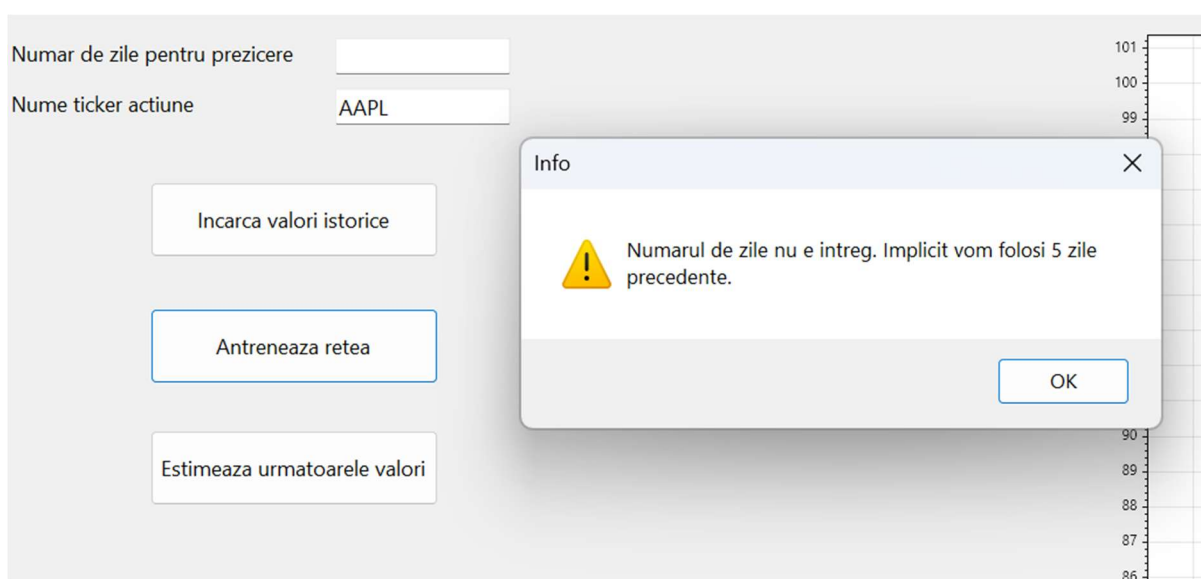
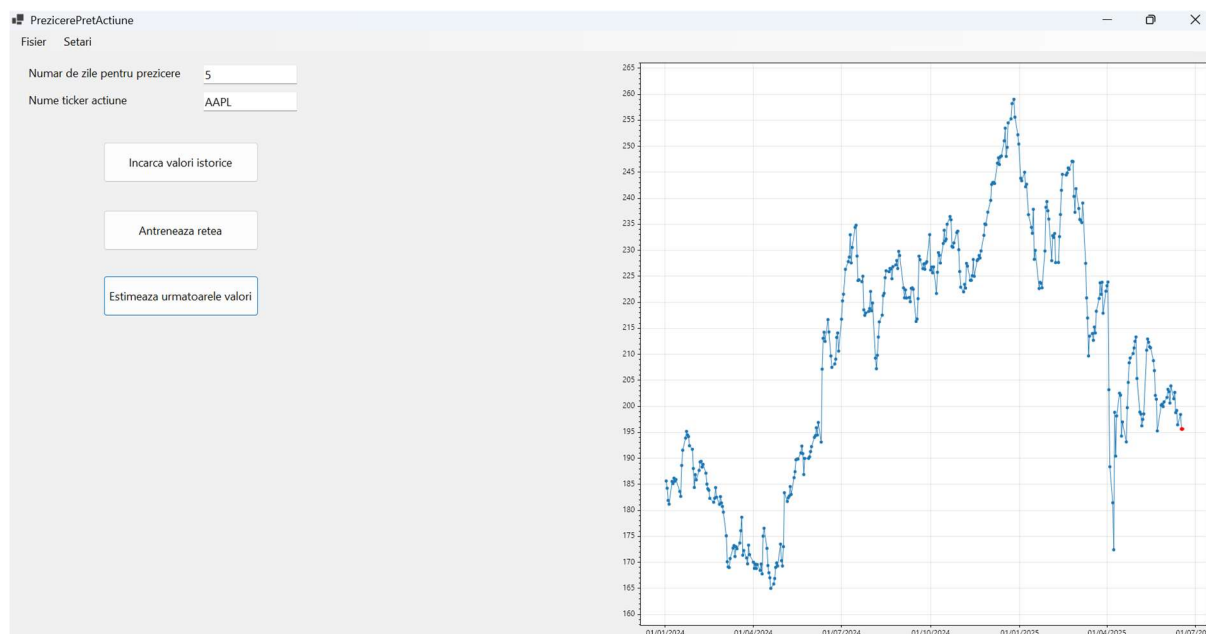


FIGURA 11 - NUMAR DE ZILE UTILIZATE LA PREZICERE

În urma finalizării cu succes a procesului de antrenament, este posibilă estimarea valorii subsecvente, obținându-se o predicție de 195.64.



**FIGURA 12 - AVEM REZULTAT**

Deși Figura 11 prezintă o reprezentare aparent punctuală, rezultatul unei examinări detaliate, vizibilă în Figura 12 la un nivel de magnificare superior, demonstrează că este, de fapt, o linie

roșie ce conectează valoarea observată cu predicția corespunzătoare.

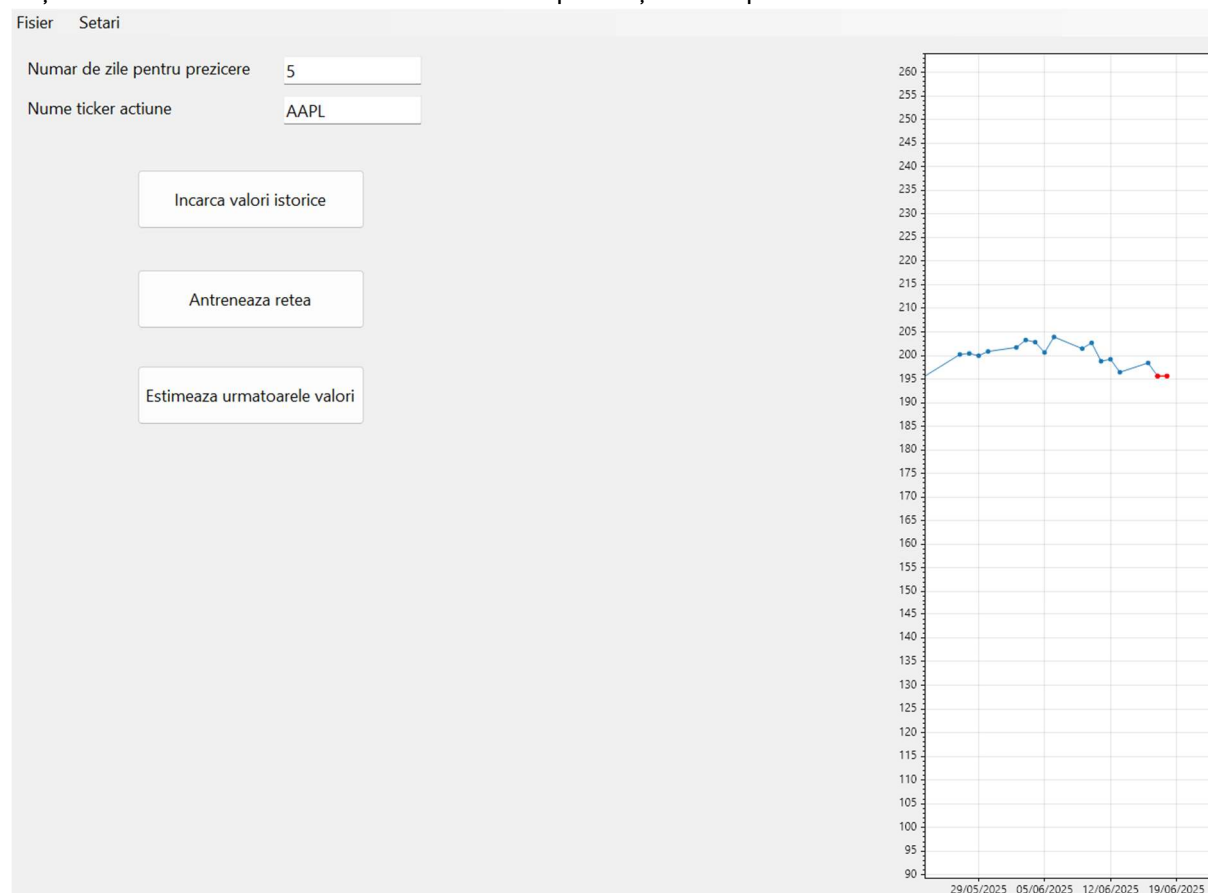


FIGURA 13 - ZOOM IN GRAFIC

Secțiunea de preferințe a aplicației include un panou dedicat configurării parametrilor rețelei neuronale de tip LSTM. Această interfață permite ajustarea următoarelor atribute: rata de învățare (learning rate), tipul metodei de normalizare a datelor utilizate, funcția de pierdere (loss function) aplicată în faza de antrenament, numărul maxim de epoci de antrenament și, respectiv, pragul de precizie la care procesul de antrenare poate fi oprit.



PrezicerePretActiune

Viteza de invatare

Tip de normalizare Logaritm ▾

Funcție de pierdere MSE ▾

Numarul maxim de antrenamente

Precizie antrenament in procent

OK Salveaza

Aceste configurații pot fi salvate, iar la reluarea execuției programului, setările definite anterior vor fi reutilizate.

## MODULUL LSTM

Modulul LSTM este conceput pentru a prelua și procesa datele furnizate de modulul principal. Acesta operează ca o aplicație de consolă, acceptând argumente specificate prin linie de comandă, conform formatului:

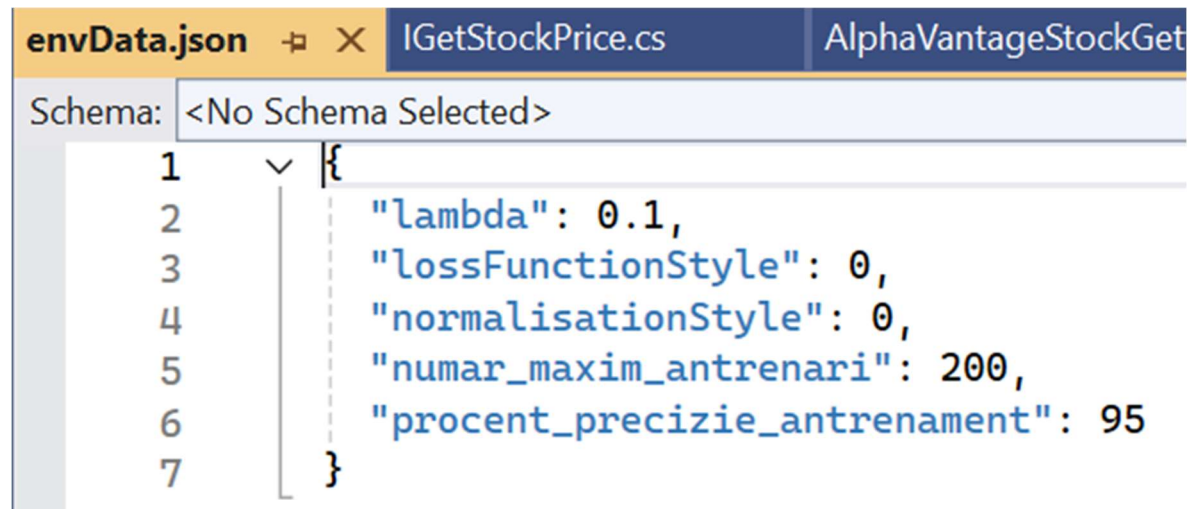
```
PrezicerePretActiuni.exe -c="<cale_fisier_configurare>" [-
forward="<cale_fisier_date_intrare>"] [-t="<cale_fisier_date_antrenament>" ] -
i=<număr_intrări>
```

Argumentul -c i se atribuie un șir de caractere reprezentând calea relativă către un fișier ce conține parametri de configurare. Acești parametri includ rata de învățare, metoda de normalizare a datelor, funcția de pierdere utilizată, numărul de epoci de antrenament și pragul procentual de acceptare pentru convergența antrenamentului.

Argumentul -i specifică numărul de intrări, indicând mai precis **câte zile anterioare vor fi utilizate pentru procesul de estimare.**

Funcționalitatea aplicației include două moduri de operare distincte: **predicția** sau **antrenamentul**. Estimarea unui rezultat se poate realiza pe baza unui model preexistent prin utilizarea comenzii -forward, sau, alternativ, aplicația poate fi antrenată folosind datele dintr-un fișier specificat prin comanda -t.

Toate fișierele relevante, incluzând cele de configurare, de antrenament, de intrare și de ieșire, sunt formate conform standardului **JSON**.



```
1  {
2      "lambda": 0.1,
3      "lossFunctionStyle": 0,
4      "normalisationStyle": 0,
5      "numar_maxim_antrenari": 200,
6      "procent_precizie_antrenament": 95
7  }
```

FIGURA 14 - EXEMPLU FISIER DE CONFIGURARE

Structura de clase e afișată în figura 15 :

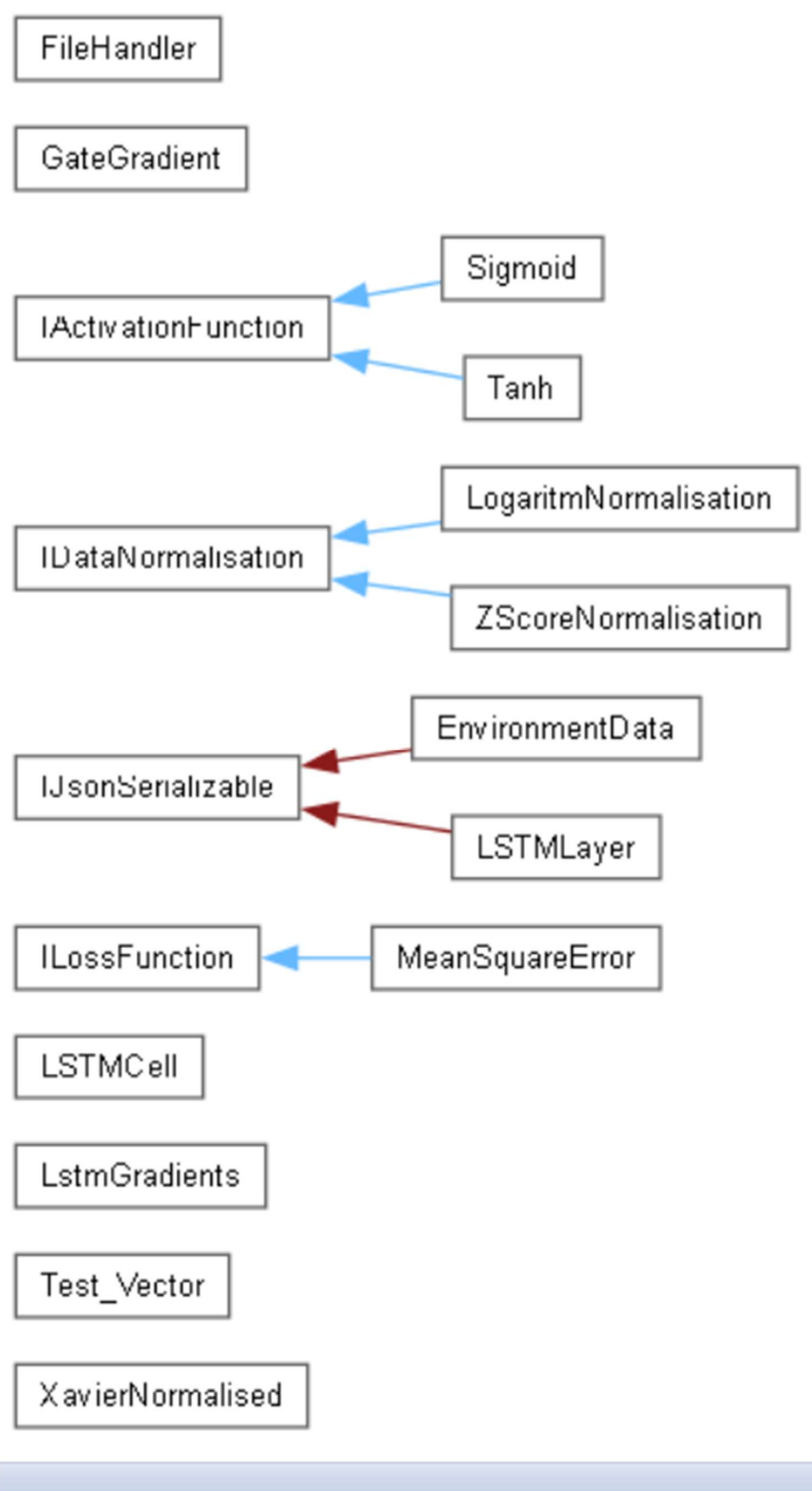


FIGURA 15 - STRUCTURA DE CLASE LSTM

Datele de configurare sunt încărcate și păstrate într-un obiect de tipul `EnvironmentData`.

Arhitectura modulară a aplicației include interfețe dedicate care facilitează extinderea și personalizarea modului de operare al modului LSTM. Acestea permit integrarea facilă a noi metode de normalizare a datelor, a diverselor funcții de activare, precum și a funcțiilor de pierdere alternative.

Pentru a asigura **extensibilitatea** și **manevrabilitatea** codului în ceea ce privește selectarea dinamică a funcțiilor de normalizare, a funcțiilor de activare și a funcțiilor de pierdere, s-a adoptat **design pattern-ul Factory Method**. Această abordare permite instanțierea flexibilă a obiectelor corespunzătoare acestor funcționalități, bazându-se pe parametrii de configurare din fișierul de configurare primit din linia de comanda (cu parametrul `-c`).

În esență, pentru fiecare categorie de funcționalitate (normalizare, activare, pierdere), a fost definită o **interfață abstractă** (sau o clasă abstractă de bază). Această interfață stabilește contractul pe care toate implementările concrete trebuie să-l respecte, asigurând uniformitate în modul de interacțiune.

Mecanismul de fabricare este încapsulat în clasa **EnvironmentData**, care, pentru a garanta consistența datelor la nivelul întregii aplicații, este implementată conform design pattern-ului **Singleton**.

Clasele care gestionează stocarea sau recuperarea datelor în format JSON vor implementa interfața abstractă **IJsonSerializable**. Această decizie de design asigură o interfață uniformă pentru serializarea și deserializarea obiectelor, facilitând stocarea și reutilizarea datelor într-un mod structurat și previzibil.

Implementarea rețelei LSTM a fost concepută sub forma unui strat de celule LSTM. Astfel, am structurat conceptul de strat LSTM într-o clasă **LSTMLayer**, iar conceptul de celulă LSTM a fost implementată într-o clasă separată, **LSTMCell**. În cadrul clasei **LSTMCell** sunt stocate valorile de ieșire din porți, în timp ce în clasa **LSTMLayer** sunt gestionate logica de antrenament și estimare, precum și valorile aferente fiecărei iterații de antrenament, esențiale pentru procesul de backpropagation.

Antrenamentul rețelei e implementat în funcția `Train`. Mai jos am dat un excerpt de cod cu aceasta funcție:

```

void LSTMLayer::Train(vector<vector<double>> in_set, int stride, double lambda)
{
    std::vector<Test_Vector> train_set;
    std::vector<Test_Vector> test_set;
    double error = 0;
    double eroare_tinta = 0;
    int num_antrenari = 0;
    EnvironmentData* env_data = EnvironmentData::getInstance(0,0,0,
DataNormalisationStyle::Logarithm, LossFunctionStyle::MSE);

    try
    {
        PrepareTraining(in_set, &train_set, &test_set, stride);
        eroare_tinta = CalcEroareTinta(&test_set);

        do {
            for (Test_Vector tv : train_set)
            {
                TrainLSTM(tv, lambda);
            }
            //acum testam daca aceasta epoca de antrenament a fost
indeajuns.
            error = TestLSTM(&test_set);
            num_antrenari++;

        } while (num_antrenari < env_data->getNumarMaximAntrenari() &&
error > eroare_tinta);

    }
    catch (const invalid_argument e)
    {
        throw;
    }
}

```

În prima parte a codului, sunt definite variabilele necesare pentru antrenament, cum ar fi **train\_set** și **test\_set**, care conțin datele de antrenament și testare. De asemenea, sunt inițializate variabilele **error** și **eroare\_tinta**, care vor fi folosite pentru a evalua performanța rețelei, precum și **num\_antrenari**, care urmărește numărul de epoci de antrenament. În continuare, este obținută o instanță a obiectului **EnvironmentData**, care conține parametrii relevanți pentru procesul de antrenament, precum stilul de normalizare a datelor și funcția de pierdere utilizată, în acest caz **MSE (Mean Squared Error)**.

După definirea variabilelor și inițializarea acestora, se pregătesc seturile de antrenament și testare utilizând metoda **PrepareTraining**. Aceasta împărțea setul inițial de date în două subseturi: unul pentru antrenament și unul pentru testare. De asemenea, eroarea țintă este calculată prin apelarea funcției **CalcEroareTinta**, care folosește setul de testare.

În cadrul unui ciclu **do-while**, sunt efectuate epoci de antrenament, în care pentru fiecare element din setul de antrenament, rețeaua este antrenată folosind metoda **TrainLSTM**. În

fiecare iterație a antrenamentului, sunt folosite datele din setul de antrenament, iar **lambda** este utilizat pentru reguarea antrenamentului. După fiecare epocă de antrenament, rețeaua este testată folosind metoda **TestLSTM**, iar eroarea este calculată pentru a evalua performanța modelului. Antrenamentul continuă până când se ating condițiile de oprire, adică până când numărul de epoci depășește limita maximă definită sau eroarea obținută este mai mică decât eroarea țintă.

În cazul în care apar erori, codul include un bloc **try-catch** care prinde excepțiile de tip **invalid\_argument**, asigurându-se astfel că eventualele erori sunt gestionate corect și pot fi relansate pentru a fi tratate la un nivel superior.

Acest cod implementează un proces iterativ de antrenament al rețelei neuronale LSTM, evaluând continuu performanța pe setul de testare și ajustând parametrii în fiecare iterație pentru a îmbunătăți acuratețea modelului.

Metoda **CalcEroareTinta** calculează eroarea utilizând metoda **MSE** (Mean Squared Error) pentru fiecare ieșire a fiecărui vector din setul de test și returnează o eroare globală unică pentru întreaga mulțime de test.

După finalizarea procesului de antrenament, funcția **TestLSTM** poate fi utilizată pentru a verifica dacă, utilizând greutatea actualizate, modelul obține performanțe corespunzătoare și pe setul de testare. În acest context, se calculează eroarea pe întregul set utilizând metoda **MSE** (Mean Squared Error), iar valoarea obținută poate fi comparată cu eroarea țintă stabilită anterior.

Funcția **TrainLSTM** implementează procesul de antrenament pentru o rețea LSTM, începând cu inițializarea variabilelor necesare pentru calcularea gradientilor și urmează pașii de **forward pass**, **backward pass**, și actualizarea greutăților.

## 1. Inițializarea variabilelor

În această etapă sunt definite matricile pentru gradientii greutăților și ale bias-urilor. Fiecare dintre aceste variabile este setată inițial la 0:

```
std::vector<std::vector<double>> grd_Wa(num_unit_ascuns,  
std::vector<double>(num_feature, 0.0));  
std::vector<std::vector<double>> grd_Wf(num_unit_ascuns,  
std::vector<double>(num_feature, 0.0));  
std::vector<std::vector<double>> grd_Wi(num_unit_ascuns,  
std::vector<double>(num_feature, 0.0));  
std::vector<std::vector<double>> grd_Wo(num_unit_ascuns,  
std::vector<double>(num_feature, 0.0));
```

```

std::vector<std::vector<double>> grd_Ua(num_unit_ascuns,
std::vector<double>(num_unit_ascuns, 0.0));
std::vector<std::vector<double>> grd_Uf(num_unit_ascuns,
std::vector<double>(num_unit_ascuns, 0.0));
std::vector<std::vector<double>> grd_Ui(num_unit_ascuns,
std::vector<double>(num_unit_ascuns, 0.0));
std::vector<std::vector<double>> grd_Uo(num_unit_ascuns,
std::vector<double>(num_unit_ascuns, 0.0));

std::vector<double> grd_ba(num_unit_ascuns, 0.0);
std::vector<double> grd_bf(num_unit_ascuns, 0.0);
std::vector<double> grd_bi(num_unit_ascuns, 0.0);
std::vector<double> grd_bo(num_unit_ascuns, 0.0);

```

Aceste matrici și vectori vor stoca gradientii calculați în procesul de **backpropagation through time** (BPTT), care vor fi folosiți ulterior pentru actualizarea greutăților și a bias-urilor.

## 2. Obținerea funcției de pierdere

Aici se obține instanța funcției de pierdere, care va fi utilizată pentru calcularea erorii și a derivatelor acesteia:

```

const ILossFunction* lossFunction =
EnvironmentData::getInstance(0,0,1,DataNormalisationStyle::Logaritm,LossFunctionS
tyle::MSE)->GetLossFunction();

```

## 3. Trecerea forward pentru fiecare pas de timp

În această etapă, rețeaua face un **forward pass** pentru fiecare pas de timp **t** din vectorul de test.

La fiecare pas, se calculează ieșirile celulei LSTM, care sunt stocate într-un vector **gates**:

```

//intai facem forward un T numar de pasi
gates = new vector<LSTMCell>;
gates->resize(T);
for (int t = 0; t < T; t++)
{
    (void)ForwardPass(test_vect.get_Test_Vector()[t]);
    (*gates)[t].resize(num_unit_ascuns);
    for (int i = 0; i < num_unit_ascuns; i++)
    {
        (*gates)[t].ag[i] = cell_gates->ag[i];
        (*gates)[t].fg[i] = cell_gates->fg[i];
        (*gates)[t].ig[i] = cell_gates->ig[i];
        (*gates)[t].og[i] = cell_gates->og[i];
        (*gates)[t].state[i] = cell_gates->state[i];
        (*gates)[t].out[i] = cell_gates->out[i];
    }
    delta_loss[t] = lossFunction-
>GetLossDerivate(test_vect.get_Rezultat_Vector(), cell_gates->out);
}

```

În fiecare iterație a buclei, se realizează un **forward pass** al rețelei, iar ieșirile sunt stocate pentru fiecare celulă LSTM. De asemenea, se calculează derivata erorii pentru fiecare pas de timp, folosind **GetLossDerivate**.

#### 4. Calcularea gradientilor prin backward pass

După ce forward pass-ul este efectuat pentru toate pașii de timp, se realizează **backpropagation** pentru a calcula gradientii greutăților și ai bias-urilor. Aceasta se face cu ajutorul metodei **BackwardPass**, care returnează gradientii calculați:

```
vector<LstmGradients> grd_gates = BackwardPass(delta_loss);
```

Gradientii sunt acum calculați pentru fiecare celulă LSTM.

#### 5. Calculul și acumularea gradientilor pentru greutăți și bias-uri

În continuare, gradientii pentru fiecare greutate și bias sunt calculați și acumulați. De exemplu, pentru greutățile **Wa**:

```
for (int t = 0; t < T; t++)
{
    for (int i = 0; i < num_unit_ascuns; i++)
    {
        for (int j = 0; j < num_feature; j++)
        {
            grd_Wa[i][j] += grd_gates[t].grd_ag_p[i] *
test_vect.get_Test_Elem(t)[j];
            grd_Wf[i][j] += grd_gates[t].grd_fg_p[i] *
test_vect.get_Test_Elem(t)[j];
            grd_Wi[i][j] += grd_gates[t].grd_ig_p[i] *
test_vect.get_Test_Elem(t)[j];
            grd_Wo[i][j] += grd_gates[t].grd_og_p[i] *
test_vect.get_Test_Elem(t)[j];
        }
        grd_ba[i] += grd_gates[t].grd_ag_p[i];
        grd_bf[i] += grd_gates[t].grd_fg_p[i];
        grd_bi[i] += grd_gates[t].grd_ig_p[i];
        grd_bo[i] += grd_gates[t].grd_og_p[i];
    }
}
```

Gradientii pentru **Wa**, **Wf**, **Wi** și **Wo** sunt calculați prin înmulțirea derivatelor erorii cu valorile de intrare corespunzătoare fiecărui pas de timp.

#### 6. Actualizarea greutăților și bias-urilor

În final, greutățile și bias-urile sunt actualizate utilizând gradientii calculați, cu un factor de învățare **lambda**:



```

//ajustam cu viteza de invatare - lambda
for (int i = 0; i < num_unit_ascuns; i++)
{
    for (int j = 0; j < num_feature; j++)
    {
        Wa[i][j] -= lambda * grd_Wa[i][j];
        Wf[i][j] -= lambda * grd_Wf[i][j];
        Wi[i][j] -= lambda * grd_Wi[i][j];
        Wo[i][j] -= lambda * grd_Wo[i][j];
    }
    for(int j = 0; j< num_unit_ascuns; j++)
    {
        Ua[i][j] -= lambda * grd_Ua[i][j];
        Uf[i][j] -= lambda * grd_Uf[i][j];
        Ui[i][j] -= lambda * grd_Ui[i][j];
        Uo[i][j] -= lambda * grd_Uo[i][j];
    }
    ba[i] -= lambda * grd_ba[i];
    bf[i] -= lambda * grd_bf[i];
    bi[i] -= lambda * grd_bi[i];
    bo[i] -= lambda * grd_bo[i];
}
}

```

În concluzie, funcția **TrainLSTM** implementează procesul complet de antrenament al unei rețele LSTM, incluzând forward pass-ul, backward pass-ul, și actualizarea greutăților folosind **gradient descent**. Aceasta optimizează rețeaua pentru a minimiza eroarea, ajustând parametrii pe baza gradientilor calculați din derivata funcției de pierdere.

Funcția **BackwardPass** implementează algoritmul de **Backpropagation Through Time** (BPTT) pentru rețelele LSTM. Aceasta este utilizată pentru a calcula gradientii asociați cu greutățile și bias-urile rețelei, pe baza erorii generate de ieșirile rețelei și a derivatelor funcțiilor de activare.

1. După initializarea variabilele urmează parcurgerea secvenței în ordine inversă (**Backpropagation**):

Algoritmul BPTT se bazează pe propagarea erorii de la ieșire spre intrare, calculând derivatele succesive pentru fiecare pas de timp. În acest scop, se iterează prin secvența de la ultimul pas de timp ( $t = \text{num\_intrari} - 1$ ) până la primul ( $t = 0$ ).

```

for (int t = num_intrari-1; t >=0; t--)
{
    gradients[t].resize(num_unit_ascuns);

```

2. **Adunarea erorii provenite din pasul următor:**

Pentru fiecare pas de timp, gradientii pentru ieșirile celulelor LSTM sunt acumulați.

Acest lucru este realizat folosind deltaLoss (eroarea de la pasul următor) și se adaugă la grd\_out.

```
for (int i = 0; i < num_unit_ascuns; i++)
{
    grd_out[i] += deltaLoss[t][i];
}
```

**Propagarea erorii la pașii anteriori:** Dacă nu suntem la primul pas de timp, se calculează erorile care se propagă înapoi prin rețea, utilizând matricele de greutate **Ua, Ui, Uf, Uo** și derivatele semnalelor de activare (sigmoid și tanh) pentru fiecare dintre ușile celulelor LSTM (ag, fg, ig, og).

```
if (t < num_intrari-1)
{
    //transpusa matricei U.
    for (int j = 0; j < num_unit_ascuns; j++)
    {
        for (int i = 0; i < num_unit_ascuns; i++)
        {
            grd_out[i] += Ua[j][i] * sigmoid->Derivate((*gates)[t + 1].ag_p[j]) * gradients[t + 1].grd_ag_p[j];
            grd_out[i] += Ui[j][i] * sigmoid->Derivate((*gates)[t + 1].ig_p[j]) * gradients[t+1].grd_ig_p[j];
            grd_out[i] += Uf[j][i] * sigmoid->Derivate((*gates)[t + 1].fg_p[j]) * gradients[t+1].grd_fg_p[j];
            grd_out[i] += Uo[j][i] * sigmoid->Derivate((*gates)[t + 1].og_p[j]) * gradients[t+1].grd_og_p[j];
        }
    }
}
```

**Calcularea gradientilor pentru fiecare componentă a celulei LSTM:** după ce s-a calculat eroarea de la pasul anterior, gradientii pentru fiecare componentă a celulei LSTM (ușile og, fg, ig, ag și starea state) sunt calculați. Pentru fiecare ușă, gradientii sunt calculați în funcție de eroarea acumulată (grd\_out) și de derivata funcției de activare respective.

De exemplu, pentru poarta de ieșire (og), gradientii se calculează astfel:

```
gradients[t].grd_og_p[i] = grd_out[i] * tanh->Output((*gates)[t].state[i]) * sigmoid->Derivate((*gates)[t].og_p[i]);
```

Gradientii pentru poarta de ieșire sunt calculați folosind derivata funcției sigmoid aplicate asupra ieșirii ușii și derivata funcției tanh aplicată asupra stării interne. Gradientii pentru starea internă sunt calculați pe baza gradientilor pentru ușa de ieșire și starea anterioară.

**Calcularea gradientilor pentru porțile fg, ig, și ag:**

Gradienții pentru porțile de uitare (fg), de intrare (ig), și de activare (ag) sunt calculați, folosind semnalele și derivările corespunzătoare pentru fiecare celulă.

De exemplu, gradienții pentru poarta de uitare (fg) sunt calculați astfel:

```
if (t > 0)
    gradients[t].grd_fg_p[i] = gradients[t].grd_state[i] * (*gates)[t - 1].state[i] * sigmoid-&gtDerivate((*gates)[t].fg_p[i]);
else
    gradients[t].grd_fg_p[i] = 0;
```

Aici, gradienții pentru ușa de uitare sunt calculați pe baza stării anterioare a rețelei, iar pentru primul pas de timp ( $t == 0$ ), nu există starea anterioară, deci gradienții sunt setați la zero.

**Returnarea rezultatelor:** după calculul gradienților pentru toate celulele LSTM, funcția returnează un vector de **LstmGradients** pentru fiecare pas de timp:

```
return gradients;
```

Funcția **ForwardPass** reprezintă un pas important în procesul de propagare înainte al rețelei LSTM. Acesta calculează activările fiecărui strat din rețeaua LSTM, pornind de la datele de intrare și utilizând greutatea și pragurile corespunzătoare fiecărei uși (gates) din celula LSTM. În cadrul acestei funcții, datele de intrare sunt utilizate pentru a calcula valorile pentru ușile de uitare, de intrare, de activare și de ieșire, conform formulelor LSTM.

**Explicatie pas cu pas:**

1. **Inițializarea variabilelor temporare:** Pentru fiecare unitate ascunsă, se initializează patru variabile temporare: temp\_forget, temp\_input, temp\_output, și temp\_activare. Acestea vor fi utilizate pentru a acumula valorile corespunzătoare fiecărei uși din celula LSTM.

```
double temp_forget;
double temp_input;
double temp_output;
double temp_activare;
```

**Calculul valorilor pentru portile de uitare, de intrare, de activare și de ieșire:** Se calculează fiecare dintre aceste valori pe baza datelor de intrare și a activărilor din unitățile ascunse anterioare. Fiecare ușă primește un set de greutăți, atât pentru datele de intrare ( $W_f$ ,  $W_i$ ,  $W_o$ ,  $W_a$ ), cât și pentru ieșirile unităților ascunse anterioare ( $U_f$ ,  $U_i$ ,  $U_o$ ,  $U_a$ ).

De exemplu, pentru ușa de uitare (temp\_forget), aceasta este calculată prin multiplicarea fiecărei intrări cu greutatea corespunzătoare și adunarea rezultatelor. Similar pentru ușa de intrare, ieșire și activare.

```
for (int k = 0; k < num_feature; k++)
{
    temp_forget += Wf[i][k] * x[k];
    temp_input += Wi[i][k] * x[k];
    temp_output += Wo[i][k] * x[k];
    temp_activare += Wa[i][k] * x[k];
}

for (int n = 0; n < num_unit_ascuns; n++)
{
    temp_forget += Uf[i][n] * cell_gates->out[n];
    temp_input += Ui[i][n] * cell_gates->out[n];
    temp_output += Uo[i][n] * cell_gates->out[n];
    temp_activare += Ua[i][n] * cell_gates->out[n];
}
```

**Adăugarea bias:** După calcularea valorilor pentru porțile de uitare, intrare, activare și ieșire, sunt adăugate biasurile corespunzătoare (bf, bi, bo, ba) pentru fiecare poartă. Aceste biasuri sunt valori care ajută la ajustarea activării rețelei.

```
temp_forget += bf[i];
temp_input += bi[i];
temp_output += bo[i];
temp_activare += ba[i];
```

**Calcularea activării pentru fiecare poartă:** După calcularea valorilor intermediare, activările pentru fiecare poartă sunt calculate folosind funcțiile de activare corespunzătoare. Porțile de uitare, de intrare și de ieșire utilizează funcția sigmoid, iar poarta de activare utilizează funcția tanh.

```
cell_gates->fg[i] = sigmoid->Output(temp_forget);
cell_gates->ig[i] = sigmoid->Output(temp_input);
cell_gates->og[i] = sigmoid->Output(temp_output);
cell_gates->ag[i] = tanh->Output(temp_activare);
```

**Actualizarea stării interne a celulei:** Starea celulei LSTM este actualizată folosind poarta de uitare și poarta de intrare. Starea anterioară este combinată cu noile valori de intrare pentru a obține noua stare. De asemenea, activarea de ieșire este calculată pe baza noii stări, utilizând poarta de ieșire.

```
cell_gates->state[i] = cell_gates->fg[i] * cell_gates->state[i] + cell_gates->ig[i] * cell_gates->ag[i];
if (cell_gates->state[i] > 0)
    cell_gates->out[i] = cell_gates->og[i] * tanh->Output(cell_gates->state[i]);
else
```

```
cell_gates->out[i] = 0;
```

**Returnarea ieșirii:** La final, funcția returnează vectorul de ieșire (out) al celulei LSTM pentru fiecare unitate ascunsă, care va fi utilizat în etapele ulterioare ale rețelei pentru propagare.

```
return cell_gates->out;
```

Funcția `PrepareTraining` este responsabilă pentru pregătirea setului de date de antrenament și de testare pentru o rețea LSTM, având în vedere parametrii specifici ai setului de date și ajustând datele pentru a fi procesate corect în cadrul rețelei. În această funcție, sunt realizate două etape importante: verificarea dimensiunii setului de date și împărțirea acestuia în subseturi pentru antrenament și testare.

**Verificarea dimensiunii setului de date:** Prima parte a funcției verifică dacă dimensiunea setului de date este suficient de mare pentru a putea fi utilizat într-un proces de învățare eficient. Dacă setul de date este prea mic în comparație cu numărul de intrări și unități ascunse, funcția aruncă o excepție (`invalid_argument`), semnalând că setul de date nu este adecvat.

```
if (num_intrari*2+num_unit_ascuns > in_set.size() * 30 / 100)
{
    throw std::invalid_argument("Marimea setului de date este prea mica!
    Incearca sa dai un set de date de cel puțin "+ std::to_string((num_feature * 2 +
    num_unit_ascuns)*10/3+1));
}
```

Această verificare presupune că setul de date trebuie să fie suficient de mare pentru a asigura o învățare corectă și pentru a putea folosi datele eficient în rețea.

**Calcularea numărului de elemente pentru antrenament:** În etapa următoare, funcția calculează numărul de elemente care vor fi utilizate pentru setul de antrenament. Aici se presupune că 70% din setul de date va fi utilizat pentru antrenament. Astfel, este calculată valoarea corespunzătoare pentru acest procentaj, iar intervalul de date pentru antrenament este determinat ținând cont de `stride` (pasul de deplasare al ferestrei de antrenament).

```
int num_elem_70_procent = (int)in_set.size() * 70 / 100;
// sliding window size e numarul_intrari
// vrem sa vedem care e indexul mai apropiat de acest 70% dupa ce il impartim.
int num_vector_antr = (int)(num_elem_70_procent - num_intrari) / stride+1;
// avem un set de numar de intrari elemente, si fiindca se suprapun se avanseaza
cum stride de acum. se scade 1 ca indexu porneste de la 0.
int idx_ultim = (num_vector_antr - 1) * stride + num_intrari - 1;
```

Acest calcul ajută la definirea unui interval de date pentru antrenament, care va fi folosit în continuare pentru a crea subseturi de date pentru rețea.

**Determinarea indexului de start pentru testare:** Funcția calculează indexul de start pentru setul de testare, care va fi plasat imediat după intervalul de antrenament. Acest index este determinat ținând cont de numărul de elemente de intrare și de numărul de unități ascunse.

```
int idx_test_vector_start = idx_ultim + num_unit_ascuns;
```

**Popularea setului de antrenament:** Acum, funcția creează subseturi de date pentru setul de antrenament. Se parcurge setul de date inițial și, pe baza dimensiunii ferestrei de antrenament (determinată de numărul de intrări și de pasul stride), sunt selectate datele corespunzătoare pentru antrenament și adăugate într-un obiect de tipul Test\_Vector.

```
for (int i = 0; i + num_intrari + num_unit_ascuns-1 <= idx_ultim; i += stride)
{
    //punem un vector de test cu iesire cu tot.
    tmp_set.assign_Test_Elem(in_set.begin() + i, in_set.begin() + i +
num_intrari);
    tmp_set.assign_Rezultat_Elem(in_set.begin() + i + num_intrari,
in_set.begin() + i + num_intrari + num_unit_ascuns);
    training_set->push_back(tmp_set);
}
```

**Popularea setului de testare:** După ce setul de antrenament a fost completat, funcția continuă să umple setul de testare. Pentru testare, setul de date este extras din partea rămasă a setului original, începând de la indexul determinat anterior. La fel ca în cazul antrenamentului, datele de intrare și rezultatele sunt adăugate într-un obiect Test\_Vector și apoi în vectorul test\_set.

```
for (int i = idx_test_vector_start; i + num_intrari + num_unit_ascuns <
in_set.size(); i += stride)
{
    tmp_set.assign_Test_Elem(in_set.begin() + i, in_set.begin() + i +
num_intrari);
    tmp_set.assign_Rezultat_Elem(in_set.begin() + i + num_intrari,
in_set.begin() + i + num_intrari + num_unit_ascuns);
    test_set->push_back(tmp_set);
}
```

În acest pas, funcția se asigură că datele pentru testare sunt corespunzător extrase și organizate pentru evaluarea ulterioară a performanței rețelei.

Funcția PrepareTraining pregătește seturile de date pentru antrenament și testare într-o rețea LSTM. Aceasta face mai întâi o verificare a dimensiunii setului de date pentru a se asigura că este suficient de mare. Apoi, împarte setul de date în două părți: 70% din datele inițiale sunt utilizate pentru antrenament, iar restul sunt alocate pentru testare. Fereastra de antrenament este generată pe baza unui parametru stride, care determină cum se mută fereastra pe setul de

date. Astfel, funcția creează seturi de date corespunzătoare pentru antrenament și testare și le organizează în obiecte `Test_Vector` pentru procesarea ulterioară.

## TESTARE SI REZULTATE

Rețeaua a demonstrat performanțe bune în testele realizate pe serii temporale simple, cum ar fi funcțiile liniare și cele sinusoidale. Pentru testele efectuate cu valori de acțiuni, au fost selectate 9 acțiuni de largă cunoaștere. După antrenament, au fost utilizate valorile estimate pentru data de 18 iunie a anului curent. În coloana **REAL** a fost înregistrat prețul de închidere pentru acea zi.

18.06	ticker	ESTIMARE	REAL	CLOSE 17.06
Apple	AAPL	195,64	196,49	195,64
Google	GOOGL	176,127	176,44	177,23
Microsoft	MSFT	478,04	478,63	478,04
Airbus	AIR.PA	160,74	166,8	161,26
LuisVuitton	MC.PA	461,41	461,06	460,65
Shell	Shel	72,3	72,49	72,33
SAP	SAP	292,8	292,77	292,64
Ferrari	RACE	461,11	461,06	460,65
Christian Dior	DIO.F	435,0	433,8	435

**TABEL 1**

Se observă o deviație relativ mică între valorile estimate și cele reale. Această performanță trebuie analizată în contextul volatilității naturale a prețurilor bursiere, care impune restricții asupra preciziei predictibilității. În analiza prețului de deschidere din ziua anterioară, un aspect interesant constă în verificarea capacității rețelei LSTM de a identifica direcția de creștere. Dintre cele 9 acțiuni examinate, în 4 cazuri rețeaua a prezis corect direcția de creștere, rezultând astfel într-o rată de succes de 44,4%.

## CONCLUZII SI DIRECTII DE IMBUNATATIRI

Estimările bazate exclusiv pe prețurile valorilor anterioare utilizând rețele neuronale LSTM nu conduc la rezultate semnificativ mai bune decât un model aleatoriu, cu o rată de succes de aproximativ 50%. Totuși, prin integrarea unor informații suplimentare, cum ar fi prețurile materialelor anterioare sau evoluția concurenței, este posibilă o îmbunătățirea acestui procentaj.

O direcție de îmbunătățire este de a adăuga o analiză a sentimentului. Studiul lui Kemal Kirtac și Guido Germana evaluează performanța modelelor de limbaj OPT, BERT și FinBERT

comparativ cu dicționarul Loughran-McDonald în analiza sentimentului a peste 965.000 de articole financiare din SUA (2010-2023). Rezultatele indică o precizie de 74,4% în predicția randamentelor pieței pentru modelul OPT bazat pe GPT-3, depășind celelalte metode. Aplicarea unei strategii long-short generează un Sharpe (un indice de randament) de 3,05, cu un randament de 355% între august 2021 și iulie 2023. [7] Spre comparație la momentul scrierii acestui document indicele Sharpe al Vanguard FTSE All World e de 0,270.

O altă direcție de îmbunătățire ar fi utilizarea exclusivă a C++ pentru întreaga aplicație, inclusiv pentru dezvoltarea interfeței grafice, având în vedere că Qt oferă suport complet pentru crearea interfețelor grafice. Astfel, nu ar mai fi necesar să se utilizeze C# doar pentru partea de interfață grafică, ceea ce ar simplifica arhitectura aplicației și ar contribui la îmbunătățirea portabilității și performanței acesteia.

Utilizarea Qt cu C++ pentru dezvoltarea aplicațiilor, în loc de C# cu alte biblioteci, poate fi justificată prin mai multe motive tehnice și funcționale, în special în contextul cerințelor de performanță și portabilitate ale aplicației.

În primul rând, Qt este un framework extrem de performant, care este optimizat pentru aplicații ce necesită un control detaliat asupra resurselor hardware și o manipulare eficientă a memoriei. C++ este un limbaj care permite un control mai granular asupra proceselor de gestionare a memoriei și al resurselor sistemului, oferind astfel o performanță ridicată, esențială în aplicațiile care trebuie să ruleze rapid și eficient, mai ales atunci când sunt implicate volume mari de date sau procese intense de calcul.

De asemenea, Qt este un framework multiplatformă, ceea ce înseamnă că aplicațiile dezvoltate cu Qt pot fi portate cu ușurință pe diverse sisteme de operare, inclusiv Windows, Linux și macOS. Această portabilitate este un avantaj semnificativ pentru dezvoltatorii care doresc să creeze aplicații ce trebuie să funcționeze într-un mediu diversificat, fără a fi necesare modificări majore ale codului sursă. În contrast, aplicațiile dezvoltate cu C# sunt mai strâns legate de platforma Windows, iar portabilitatea acestora către alte sisteme de operare poate necesita mai mult efort sau utilizarea unor tehnologii suplimentare.

În plus, eliminarea codului responsabil pentru interfațarea și comunicarea dintre aplicația C# și C++ contribuie la simplificarea arhitecturii aplicației și la îmbunătățirea performanței generale. Aceasta reduce complexitatea și resursele necesare pentru gestionarea interoperabilității între cele două limbaje.



## BIBLIOGRAFIE

1. Glorot, Xavier, Yoshua Bengio. “Understanding the Difficulty of Training Deep Feedforward Neural Networks.” *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, vol. 9, 2010, pp. 249–256.
2. [https://erc.europa.eu/sites/default/files/2023-12/AI\\_in\\_science.pdf](https://erc.europa.eu/sites/default/files/2023-12/AI_in_science.pdf) 2023
3. <https://www.csiro.au/-/media/D61/AI4Science-report/AI-for-Science-report-2022.pdf> , 2022
4. [https://www.oecd.org/content/dam/oecd/en/publications/reports/2024/11/assessing-potential-future-artificial-intelligence-risks-benefits-and-policy-imperatives\\_8a491447/3f4e3dfb-en.pdf](https://www.oecd.org/content/dam/oecd/en/publications/reports/2024/11/assessing-potential-future-artificial-intelligence-risks-benefits-and-policy-imperatives_8a491447/3f4e3dfb-en.pdf) , 2024.
5. <https://www.imf.org/-/media/Files/Publications/WP/2025/English/wpica2025068-print-pdf.ashx>, 2025.
6. <https://www.imf.org/-/media/Files/Publications/WP/2025/English/wpica2025068-print-pdf.ashx>
7. [Sentiment trading with large language models](#), 2024
8. Greff, Klaus, et al.- “LSTM: A Search Space Odyssey.” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, 2017, pp. 2222–2232. IEEE, <https://doi.org/10.1109/TNNLS.2016.2582924>.
9. Impact of Data Normalization on Stock Index Forecasting - S. C. Nayak<sup>1</sup>, B. B. Misra, H. S. Behera.



ROMÂNIA  
MINISTERUL EDUCAȚIEI  
**UNIVERSITATEA „VASILE ALECSANDRI” DIN BACĂU**  
Calea Mărășești, Nr. 157, Bacău, 600115  
Tel. +40-234-542411, fax +40-234-545753  
www.ub.ro; e-mail: rector@ub.ro



**DECLARAȚIE DE AUTENTICITATE**  
privind elaborarea lucrării de finalizare studii

Subsemnata, Andreea-Corina Chelariu,  
declar pe propria răspundere, că:

- a) lucrarea a fost elaborată personal și îmi aparține în întregime;
- b) nu au fost folosite alte surse decât cele menționate în bibliografie;
- c) nu au fost preluate texte, date sau elemente de grafică din alte lucrări sau din alte surse fără a fi citate și fără a fi precizată sursa preluării, inclusiv în cazul în care sursa o reprezintă alte lucrări ale mele;
- d) lucrarea nu a mai fost folosită în alte contexte de examen sau de concurs;
- e) sunt de acord ca lucrarea să fie verificată prin orice modalitate legală, pentru confirmarea autenticității, consimțind inclusiv la introducerea conținutului acesteia într-o bază de date în acest scop;
- f) am luat la cunoștință faptul că este interzisă comercializarea unei lucrări științifice în vederea facilitării falsificării calității de autor al acesteia;
- g) în elaborarea lucrării nu am utilizat instrumente specifice inteligenței artificiale (IA).

Data

Semnătura

.....

.....