Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Hálózati Rendszerek és Szolgáltatások Tanszék

Ceffer Attila

ALGORITMIKUS KERESKEDÉS KÜLÖNBÖZŐ HARDVER PLATFORMOKON

Tudományos Diákköri Konferencia

KONZULENS

Dr. Levendovszky János

BUDAPEST, 2013

Összefoglaló

Jelen dolgozat a különböző hardver platformokon (CPU és GPGPU) implementált algoritmikus kereskedéssel foglalkozik A kereskedési algoritmust egy NARX neurális hálózat valósítja meg. A valós idejű analízis, predikció és kereskedési döntések meghozása központi szerepet játszanak a modern pénzügyi szolgáltatásokban. Ezért az algoritmusoknak – a különböző hardverplatformokon való - futási idő vizsgálata alapvető fontossággal bír a gyakorlatban is implementálható elektronikus kereskedés számára. Így válik lehetővé ugyanis az alacsony frekvenciás kereskedésről a magas frekvenciás (intraday) kereskedésre való átmenet.

Ebben a munkában rövid történeti áttekintés után egy korszerű neurális (NARX) előrejelző algoritmus kerül bemutatásra, több tanító eljárás és költség függvény felhasználásával. Az említett algoritmusok megvalósítása a szokásos egyprocesszoros és multiprocesszoros rendszerek mellett GPGPU-ra is megtörtént. Az eljárások teljesítőképessége valós adatokon kerültek tesztelésre, melyek bizonyították, hogy a párhuzamos implementáció felgyorsítja a kereskedést és ezáltal profitot eredményez még vételi-eladási árkülönbözet (bid-ask spread) és tranzakciós költség jelenlétében is. A dolgozat az eljárások részletes sebesség analízisét is tartalmazza.

Abstract

This work is concerned with algorithmic trading by NARX neural architectures implemented on different hardware platforms, such CPU and GPGPU. As real-time analysis, prediction and trading decisions play central roles in modern financial services, speed profiling of different algorithms running on a great variety of hardware platforms has become increasingly necessary.

In the present work, after a brief historical introduction, a present-day neural forecasting algorithm is introduced with the use of different training algorithm and error functions. The corresponding trading algorithms have been implemented on CPU and GPGPU. The performance of the methods is extensively tested real financial time series, such SP 500 and FOREX rates. As the performance analysis demonstrated parallel implementation can indeed speed up trading and leads to profit even in the presence of bid-ask spread and transaction costs. A detailed speed profiling of the methods is also included in the work.

Tartalomjegyzék

1 Bevezetés

Hosszú évszázadokon keresztül alig változott a tőzsdei kereskedés. Az 1600-as évektől kezdve, mikor az első részvényt kibocsátották, az 1970-es évekig az elemzés és döntéshozatal folyamata változatlan maradt. Az elektronikus kereskedés azonban átformálta a piacokat. Az informatika terjedésével olyan fejlett algoritmusok kezdték meghozni a kereskedési döntéseket, melyeket mély matematikai modellek támogatnak.

Az algoritmikus kereskedés tagadhatatlanul a modern pénzügyi világ leginkább kutatott területe. A XXI. században már bárki futtathat olyan kereskedői ágenst, mely teljesen önállóan, a másodperc törtrésze alatt képes döntéshozatalra, s a használó beavatkozása nélkül elvégzi a vételi vagy eladási akciót. Nem véletlen, hogy ezek a „robotok" egyre inkább elmozdultak az alacsony frekvenciás kereskedésről *(low frequency trading – napon túli)* a nagyfrekvenciás kereskedés felé *(high frequency trading – napon belüli)*, hiszen a nagy számítási teljesítmény kiaknázásával több profit elérése lehetséges azonos időn belül. Ehhez azonban olyan kifinomult algoritmusok és matematikai modellek szükségesek, melyek egyrészt nagy bizonyossággal meghatározzák az optimális tranzakciót, illetve a szükséges számítási kapacitást és ennek megfelelően az architektúrát (CPU, GPGPU) is optimálisan használják ki.

Jelen dolgozat egy, a véletlen értékpapír árakból álló idősorának megfigyelésén alapuló predikciós eljárással foglalkozik, mely a NARX típusú neurális hálózatot használja a jövőbeli ár-irányok meghatározására. Neurális hálózatok alkalmazása azért előnyös, mert nem igényelnek erős statisztikai feltételezéseket a mögöttes folyamatokról, mégis akár nemlineáris modelleket is képes megvalósítani. Mivel a négyzetes hiba alapú tanítási eljárások a várható profit tekintetében nem optimális kritériumfüggvények, a tanításhoz új költségfüggvények szükségesek, melyek e szempontból jobban teljesítenek, azonban még mindig gyorsan kiértékelhetőek egy adott célhardveren.

Míg négyzetes hiba alapú optimalizációra már jól bevált algoritmusok léteznek, addig ezen új költségfüggvényekre nem ismert ilyen tanítás, ezért a szokásos hiba-visszaterjesztéses eljárás helyett sztochasztikus szélsőérték-kereső algoritmusok alkalmazhatók. A jól ismert Boltzmann logaritmikus lehűtés mellett két fejlettebb

lehűtési változat kerül ismertetésre, melyek gyorsabban konvergálnak az optimális megoldáshoz.

Mivel a mai többprocesszoros és többmagos számítógép architektúrák párhuzamosan több feladatot futtathatnak, a szimulált lehűtés párhuzamosított változata is a vizsgálatok tárgyát képezi, mely az eredményt nemcsak hamarabb adja meg, de jobb is lesz a soros változatnál.

Bonyolult folyamatok modellezése esetén nem ritka több szakértő használata, ezért a predikciót egy neuronháló helyett már azok lineáris kombinációja fogja szolgáltatni. Az optimális súlyok megválasztása egy további optimalizációs kihívást hordoz, melyre három egyszerű, de hatékony eljárás kerül alkalmazásra.

A bemutatott modellekre nagysebességű, jól skálázódó multiprocesszoros és GPU alapú implementációt is készítettem, mely a kereskedési stratégia teljesítőképességét hivatott elemezni. Tesztelést egyrészt saját eszközökön végeztem, másrészt a BME új klaszter alapú szuperszámítógépén *(Superman),* mind FOREX, mind S&P 500 adatokon. A fejlesztett algoritmussal jelentős profit realizálható valós körülmények között, a tesztek nem középárfolyamokon, hanem bid-ask spread és kereskedési költség jelenlétében kerültek futtatásra.

A dolgozat a fent említett témákat és eredményeket az alábbi felépítésben tárgyalja:

- **1. fejezet:** Rövid történeti áttekintés és az algoritmikus kereskedés kihívásai;
- **2. fejezet:** A szimuláció módszertana, az alkalmazott modellezési eszköztár és számítási modell ismertetése;
- **3. fejezet:** A modell szabad paramétereinek beállítása;
- **4. fejezet:** A kifejlesztett algoritmus és a kereskedési stratégia leírása;
- **5. fejezet:** A szoftveres implementációk leírása, a CPU és GPU optimális kihasználásának kihívásai;
- **6. fejezet:** Az empirikus eredmények ismertetése, illetve azok értékelése;
- **7. fejezet:** Konklúziók levonása, továbbfejlesztési lehetőségek vizsgálata.

1.1 Rövid történeti áttekintés

A XVII. század hajnalán, 1602-ben megalakult a Holland Kelet-Indiai Társaság - a világ első részvénykibocsátója - mely a modern tőzsde kialakulásának egyik legfontosabb állomása. Hollandia a gyarmatai révén különösen meggazdagodott, Amszterdam a nemzetközi pénzforgalom központjává vált. Az első kibocsátás 1602 augusztusában történt, már ekkor komoly érdeklődés övezte a befektetetések ezen új formáját. (Júlia, 2013)

Európa szerte sorra nyitották meg kapuikat a tőzsdék, ahol különböző termékek, továbbá vállalatok értékpapírjaival lehetett kereskedni. Több mint kétszáz évvel később, 1817-ben megalakult a New York-i Értéktőzsde, korunk legnagyobb részvénypiaca. A tranzakciók bonyolítását a brókerek végezték, a tőzsde parkettjén kínáltak különböző árakat az egyes értékpapírokért, melyek értéke (utoljára jegyzett tranzakció kötési ára) folyamatosan változott.

Azóta sokat változott a világ. Még ma is megtalálható egy jellegében hasonló folyamat, azonban a papír alapú jegyzés lényegében teljesen megszűnt. A XX. század második felétől megkezdődött a tőzsdéken is a gépesítés, az ajánlatok összerendelését már egy informatikai infrastruktúra végezte el. 1971-ben a NASDAQ volt az első elektronikus részvény piac, s ekkorra tehető az első algoritmikus kereskedő robotok születése is. A 2000-es évekre már a világ tőzsdéi elérhetőek egy internetkapcsolattal felruházott személyi számítógép segítségével, így gyakorlatilag a világ bármely pontjáról intézheti pénzügyeit a befektető. Az eddig jellemző napon túli kereskedés szerepét kezdte át venni a napon belüli, amikor egy pozíciót ugyanazon a napon be is zárják, mint amikor kinyitották. Az árfolyamokat valós időben tudták figyelni, a döntések végrehajtásához pedig elegendő volt a billentyűzet Enter gombjának megnyomása. Látható, hogy az információs infrastruktúra fejlődése milyen változásokat hozott a tőzsdék világába, azonban ez csak a jéghegy csúcsa.

Már az első processzorok megjelenésekor is felmerült az igény az emberi tényező további csökkentésére. A befektető nem akarta egész nap számítógépe képernyőjét nézni, ezért olyan megoldások után kezdett nézni, melyek a döntési folyamatokat is automatizálják.

1.2 Az algoritmikus kereskedés és kihívásai

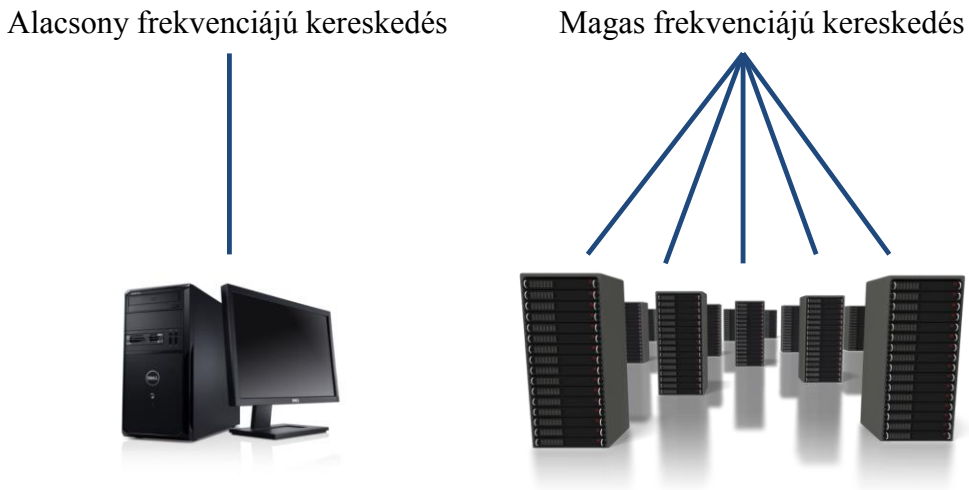
A technikai elemzés és fundamentális befektetési formák mellett körülbelül az 1980-as évektől kialakult az algoritmikus kereskedés. Ezek mögött mély matematikai modellek húzódnak, melyek kiértékelésére csak valamilyen nagy számítási képességű eszköz használható megfelelő sebességgel.

Jellemzően használt modellek például az úgynevezett „mean reverting" módszerek, melynek alapja, hogy az árfolyamok, vagy azok optimális lineárkombinációja teljesíti a mean-reverting tulajdonságot, azaz a folyamat egy hosszú távú átlaghoz tart. Ez úgy ad lehetőséget kereskedésre, hogy az átlagtól távol felvétetik a pozíció, majd visszatéréskor bezárásra kerül. Miért jelent ez optimalizációs kihívást?

A kereskedési költségek minimalizálása miatt a portfólió (értékpapírok lineárkombinációja) nem nulla elemeinek számát célszerű egy adott határ alatt tartani. Ez azonban azt jelenti, hogy például S&P 500 használatakor 500 lehetséges részvényből kell néhányat kiválasztani. A probléma a közismert Részhalmaz problémára polinom időben visszavezethető, melyből következik, hogy ez is NP-nehéz. (d'Aspremont, 2011) Épp ezért már alacsony értékekre is kezelhetetlen problémát eredményez, hiszen 5 kiválasztása esetén $\left(\frac{500!}{5!(500-5)!}\right) = 2.55 \times 10^{11}$ lehetőséget kell végignézni, mely még egy gyorsan kiértékelhető költségfüggvény mellett is esélytelen valós időben meghatározni. A megoldás (a kimerítő keresés mellett) lehet például valamilyen heurisztika alkalmazása, azonban nem biztos, hogy ezek a legjobb megoldást szolgáltatják, sokkal valószínűbb egy szuboptimális paraméterhalmaz. (Fogarasi, és mtsai., 2011) A másik eljárás a real-time kiértékelhetőségre olyan nagysebességű platformok teljesítményének kihasználása, melyek – akár egymagukban, akár klaszter alapon csatolva – már képesek megbirkózni a hatalmas számítási igénnyel.

Joggal merül fel a kérdés, hogy vajon más stratégia választásakor esetleg csökkenthető a komplexitás. Például elhagyva a portfólió választást és a mean-reverting stratégiákat, valamilyen más modellt illesztve gyorsabb lehet-e a rendszer, s ezáltal komolyabb profit érhető-e el. Általában elmondható, hogy a paraméter optimalizációs eljárások mindig nagy komplexitásúak, így mindenképpen érdemes olyan eszközökre implementálni a választott stratégiát, melyek párhuzamosan, egyszerre több műveletet képesek elvégezni.

Az egyik ilyen rendszer a mára már elterjedten használt multiprocesszoros, illetve klaszter alapú számítástechnika. Ebben az esetben több teljes értékű végrehajtó egység áll rendelkezésre, melyek például egy időben nem csak egy portfóliót képesek kiértékelni. A másik, még egyelőre nem olyan széleskörűen alkalmazott technológia, a GPU általános célú felhasználása, amikor a grafikus kártya teljesítményét használják ki, s számos számítási magja miatt bizonyos feltételek teljesülése esetén jóval gyorsabb végrehajtást eredményezhet, mint a szokásos CPU alapú architektúrák. Ennek bővebb leírása az 5.1 fejezetben található.



1. ábra PC és klaszter alapú megoldások az algoritmikus kereskedés világában

2 A modellezés módszerei és algoritmusai

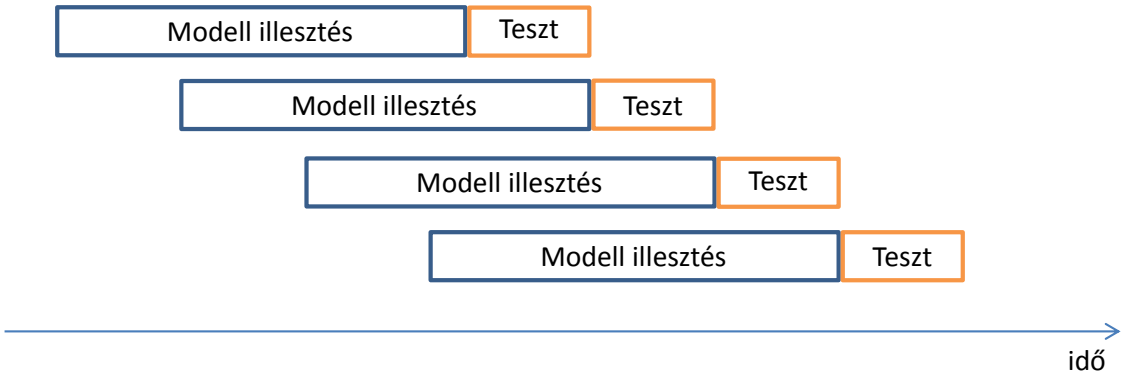
Ebben a fejezetben elsőként a modellezés módszertana kerül ismertetése, lesz szó a gyakorta elkövetett hibákról, melyek a szimuláció során jó eredményeket biztosítanak, azonban egy éles teszt során a fejlesztett algoritmus nem a várt célokat teljesíti. Ezután az alkalmazott matematikai modellek leírása következik, a neurális hálózatok elmélete, illetve azok paramétereinek beállítása szimulált lehűtéssel.

Az algoritmus során alkalmazott pontos modell a 4. fejezetben, ezekre támaszkodva kerül ismertetésre.

2.1 A „backtest” folyamata

Egy stratégia, illetve kereskedési rendszer megfelelő értékelése elengedhetetlen annak jövőbeli használatához. A „backtest” arra próbál választ adni, hogy a kifejlesztett algoritmus milyen sikerrátával képes üzemelni, valós kereskedés során milyen profit várható, mekkora kockázati szint mellett. A feltételezés az, hogy a historikus teljesítmény a jövőben is hasonlóképp alakul, a profit várható értékét a tesztelés során elért eredményesség alapján szokás becsülni. Például, ha átlagosan egy hónap alatt 5 % profitot hoz 3 % maximális visszaesés mellett, akkor ez tekinthető a várható profit és maximális visszaesés szinteknek. Ez természetesen csak abban az estben igaz, ha a modell kellően közel áll a valósághoz, hibáktól és torzításoktól mentes. Általában az egyszerűség érdekében szokás eltekinteni bizonyos tényezőktől, melyek kisebb-nagyobb mértékben befolyásolják az eredményességet. Ezek leírása a 2.1.1 fejezetben találhatók.

A backtest során jellemző az úgynevezett csúszóablakos *(sliding window)* tesztelés. Ekkor az algoritmus tanítása egy olyan múltbeli adathalmazon történik, mely a teszt halmaz adatainál korábbi. Tanítás alatt az algoritmus szabad paramétereinek beállítását kell érteni, azaz például egy neuronháló háló esetén annak súlyait. A modellillesztés után a teszt idősoron történik az eredményesség kiértékelése, ekkor derül ki, hogy milyen múltbéli profitot sikerült volna elérni. A folyamatot a 2. ábra szemlélteti.



2. ábra Csúszóablakos tesztelés

Fontos, hogy a kiértékelő halmaz adatai a tanító halmaztól teljesen független legyen, ellenkező esetben már a tanítás során ismeretes lesz jövőbeli információ, mely egy valós kereskedési helyzetben nem fordulna elő. (Bandy, 2013)

A teszt halmaz mérete tetszőleges, azonban célszerű a tanítási mintahalmaznál rövidebbre választani, például 1 hónap tanító adat mellett 1 hét teszt adat. Amennyiben a teszt halmaz hossza egy, azt on-line tanításnak nevezik, hiszen ilyenkor minden időpillanatban újra kell tanítani a rendszert. Ez neuronhálók esetén általában nemkívánatos, hiszen a tanítási fázis ilyen modelleknél sok időt vesz igénybe, s esetlegesen emiatt maradhat le egy-egy pozíciónyitási lehetőségről.

2.1.1 A tesztelés során tipikusan elkövetett hibák

Ha egy kész stratégia jó eredményeket ér el a backtesten, vagy már-már túl jókat, érdemes elgondolkodni azon, hogy valóban helyes-e a teszt eredménye. A gyors algoritmus fejlesztés és kiértékelés érdekében bizonyos tényezőket el kell hanyagolni, azonban ez rossz esetben durván alulbecsli a kockázati szintet és túlbecsüli a profit szintet. Ebben az alfejezetben – a teljesség igénye nélkül - néhány olyan hibáról lesz szó, mely komolyan befolyásolja egy stratégia backtest teljesítményét.

2.1.1.1 „Előretekintés”

Mint a neve is mutatja, ez egy olyan hiba mely esetén az algoritmus olyan adatokhoz fér hozzá, mely a tanítás során nem, csak később volna hozzáférhető. Egy időben rosszul eltolt mátrixszorzás okozhat ilyet, vagy már magában az algoritmus hátterében is hiba húzódik meg. Legyen például egy olyan stratégia, mely minden kereskedési nap nyitásakor vásárol, majd az adott nap maximális árán elad. Ezzel az a

baj, hogy az adott nap maximális ára csak a nap végén derül ki, azonban akkor már akár elég távol lehet attól az árszinttől.

Számos előretekintési hiba véthető és sokszor egyáltalán nem nyilván való az ok, csak az látszik, hogy a backtest során a stratégia kiváló eredménnyel szerepel, míg éles kereskedés esetén nemhogy profitot, de nagy veszteségeket realizál. (Chan, 2013)

2.1.1.2 „Túlélési torzítás”

A most vizsgált algoritmus működjön úgy, hogy olyan S&P 500-as részvényeken kereskedjen, melyek a lista elemét képezték 2013-ban, a teszt pedig a 2012-es és 2013-as évet tartalmazza, amennyi értelemszerűen rendelkezésre áll az adott pillanatig. A stratégia kizárólag vételi pozíciót nyithat, s ezeket akár hónapokig megtartja. A backtest szép eredményeket mutat, azonban egy komoly hibát tartalmaz. A S&P 500-as lista folyamatosan változik, mivel elképzelhető, hogy egy-egy cég idővel nem felel meg a követelményeknek, esetleg csődbe megy. Azonban a 2013-as lista csak olyan cégeket fog tartalmazni, melyek jól működnek 2013-ban, azaz nem történt velük komoly baj az elmúlt időszakban. Ez viszont lényegében egy fajtája az előretekintési torzításnak, hiszen 2012-ben még nem tudható, hogy mely vállalatok lesznek részei a listának. A stratégiának, mely kizárólag long pozíciót nyit, nagyon kedvez, mert csak jól teljesítő papírokkal kereskedik.

A kiküszöbölése relatíve egyszerű, mindig az adott évhez tarozó listákkal kell dolgozni, így a stratégia olyan részvényeket is vásárolhat, mely később kikerül a listáról. (Chan, 2013)

2.1.1.3 „Kukucskálás”

Legyen most egy tetszőleges vizsgált stratégia olyan, hogy a szabad paramétereit úgy állították be, hogy több tesztelési eredmény közül azt választották ki, amelyik a teszt fázisban a legjobban szerepelt. Az elvárt profit cél pedig az ehhez tartozó profit mennyiség. Vajon a stratégia a jövőben is teljesíteni fogja a kitűzött mértékeket? Nem valószínű, ugyanis megint véletlenül olyan adatokat használtak fel, mely a jövőre jellemző (bele „kukucskáltak” a teszt adatokba). Az értékelést mindig egy a tanítástól független adathalmazon kell végezni, s ennek eredményét kell figyelembe venni.

2.1.1.4 Shortolási kényszerek

Egy további elkövethető egyszerűsítés, vagy hiba a shortolási kényszerek figyelmen kívül hagyása. A nagy kapitalizációjú cégek részvényeit lehet shortolni, azaz először eladni, majd később visszavásárolni, ezáltal hasznot termelni eső piacok idején is. Azonban ez a művelet nem mindig engedélyezett. A 2008-as gazdasági krízis idején, amikor a Lehman Brothers csődöt jelentett, az összes értékpapír rövidre eladását a SEC *(U.S. Securities and Exchange Commission)* beszüntette. Értelemszerűen ekkor nem volt lehetőség részvényekkel a csökkenő USA piacon eredményesen kereskedni.

Megjegyzendő, hogy opciók kereskedésével ez megkerülhető, azonban ezzel a dolgozat keretein belül nem foglalkozom. (Chan, 2013)

2.1.2 Felhasznált adatok

A szimulációk generált (például AR1) helyett valós, historikus FOREX, illetve S&P 500 adatokat használnak. A FOREX adatok 8 db közkedvelt, nagy forgalmú devizapár minden kötési adatát tartalmazza kötési ár illetve forgalom formában. Az árak nem csak középárfolyamokat jelent, hanem tartalmazzák mind a vételi, mind az eladási árfolyamok múltbeli értékeit. Ez nagyon fontos ahhoz, hogy a valósághoz közelebb lehessen hozni a modellt, vizsgálni lehessen a stratégiát a másodlagos hatások jelenlétében. Ez sajnos a szimuláció során növeli a komplexitást, azonban elengedhetetlen a teljesítmény megfelelő értékeléséhez.

FOREX esetén lehetőség nyílik gyakorlatilag tetszőleges időfelbontásban tesztelni, hiszen mindössze mintavételezni kell az adatokat. Ezek a nagyfrekvenciás idősorok a Dukascopy svájci bank és FOREX brókercég weblapjáról ingyenesen elérhetőek bármely regisztrált ügyfél számára. Az adatok pontossága kérdéses lehet, azonban a piacon a legmegbízhatóbb szolgáltatók közé tartoznak, nem jellemzőek hibák, illetve mentes az árfolyam manipulációktól.

A részvénypiaci tesztekhez a S&P 500 lista 2012-es cégeit tartalmazza, s a tesztelés 2002-től 2012 év végéig tartott. Látható, hogy ez nem tökéletes megoldás, túlélési torzítást tartalmaz, azonban a 2008-a pénzügyi válság adatsorait is tartalmazza, így szignifikánsan eső piacok környezetében is tesztelésre került az algoritmus.

Ezek az adatok alacsony frekvenciásak, napvégi záróárakat (továbbá nyitó, minimum és maximum) és forgalmat tartalmaznak. Nem ismeretes azonban az ajánlati

könyv tartalma, így a bid és ask árak sem. Az algoritmus egy egyszerű eljárással állítja ezeket elő, a lehető legkisebb különbözetet *(spread)* számolja fel, ami általában 1 cent. Ez sajnos a valóságban nem mindig igaz, különösen nagy mennyiség kereskedése esetén, illetve akkor, ha alacsony likviditású a választott értékpapír.

Az idősorok a Yahoo Finance adatszolgáltatótól származnak, ahonnan ingyenesen letölthetőek a napvégi korrigált *(adjusted)* árak. Ez megint kritikus a tesztelés szempontjából, mivel ezek az adatok mentesek céges folyamatok eredményeként előálló torzításoktól, mely a valós kereskedés során okoz problémát, azonban a letöltött adatokból hibás következtetést vonhat le egy algoritmus:

- Ilyen például az osztalékfizetés, amikor a tulajdonosok a cég pénzállományának egy részét megkapják, s ez a részvény árán meglátszik, csökkenni fog. Ezt az algoritmus egy csökkenő mozgásnak vélheti, azonban nyilvánvalóan nem igaz.
- A másik, talán még nagyobb problémát jelentő árfolyamtorzítás, amikor a részvények számát módosítják (például megduplázzák) változatlan kapitalizáció mellett. Ez a duplázás esetén az árfolyam feleződését jelenti, ami kereskedés esetén hatalmas profitot, vagy veszteséget jelentene, holott megint csak szó sincs erről.

A nagyfrekvenciás historikus S&P 500 adatok nagyon drágák, több tízezer dollár kifizetése mellett lehet csak jó minőségben hozzájutni, ráadásul hatalmas adatmennyiséget is jelentenek.

2.2 Néhány gondolat az idősorok elméletéről

Az idősorok olyan statisztikai adatsorok, melyek egyik dimenziója az idő. Az elméleti idősorok minden eleme egy valószínűségi változóként értelmezhető, s egy adott eloszlás alapján generálódik (például normál eloszlás). Annak ellenére, hogy a dolgozat elsősorban tőzsdei idősorokra koncentrál, az alkalmazott eljárások némi körültekintéssel tetszőleges idősorra alkalmazhatók. Leírásukhoz az egyik elterjedt eljárás az *autoregresszív folyamatok* modellje, mely a következő időpillanatbeli árat az előző néhány értékből határozza meg. Legyen $\boldsymbol{s}_t$ az értékpapír árakat tartalmazó vektor a t-ik időpillanatban és $W \sim N(0, \boldsymbol{K})$ normál eloszlású zaj. Az elsőrendű autoregresszív folyamat

$$\boldsymbol{s}_t = \boldsymbol{A}\boldsymbol{s}_{t-1} + W_t ,$$

melyben a *t*-ik árat a *t-1*-ik determinálja, illetve közvetve a régebbiek is a *t-1*-iken keresztül. A folyamatra szuperponálódik még egy nulla várható értékű, ***K*** kovariancia mátrixú Gauss zaj is. (Fogarasi, és mtsai., 2011)

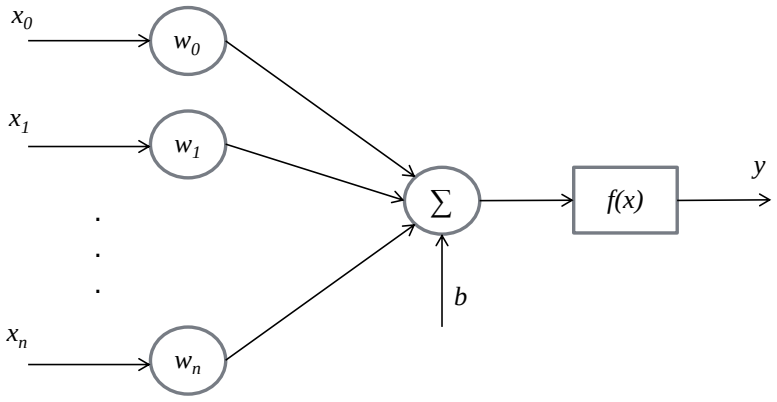
Természetesen számtalan más modell is létezik idősorok leírására, az alábbiakban egy NARX hálózat alapú modell kerül ismertetésre, mely az idősorok autoregresszív természetét is kihasználja a jövőbeli árak predikciójához.

2.3 Neurális hálózatok felhasználása előrejelzésre

A neurális hálózatokat széleskörűen használják olyan helyeken, ahol egy pontos, minden részletre kiterjedő modell felállítása nem lehetséges, vagy azért, mert nem ismert a folyamat működése, vagy mert annak szimulációja nem végezhető el valós időben. Ilyen problémák esetén célszerű egy olyan általános modell használata, mely nem igényli a paraméterek, sőt a működési mechanizmus ismeretét sem. Ezen eljárások természetesen nem fognak tökéletes eredményt adni, de ez legtöbbször nem is követelmény, hanem általában elég, ha egy bizonyos hibahatár alatt teljesítenek (pl. négyzetes hiba). A neurális hálózat paraméterei tanítás útján beállítódnak úgy, hogy annak kimenete a lehető legkisebb eltérést mutassa a modellezni kívánt folyamattól.

A hálózatok építőelemei az elemi neuronok, melyek a bemenetükre érkező adatokat súlyozva összeadják, s a kimenetet egy aktivációs függvény alkalmazásával állítják elő, mely jellemzően nemlineáris. Szokás még egy küszöb bevezetése is, mely elérése esetén aktiválódik a kimenet.

Az alábbi ábrán egy *n* bemenetű neuron felépítése látható. A bemeneti vektort az $x_0 ... x_n$, a súlyokat $w_0 ... w_n$, a küszöböt *b* szemlélteti. A végső kimenet az összegző kimenetére alkalmazott *f(x)* függvény alkalmazásával áll elő.

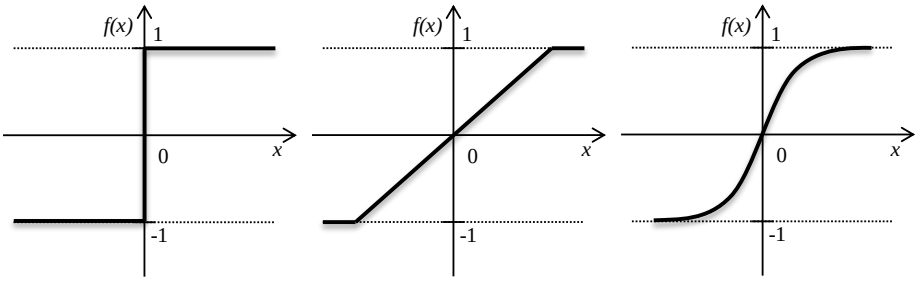


3. ábra Az elemi neuron felépítése

A kimenet formálisan:

$$y = f\left(w^T x - b\right)$$

Lineáris átmeneti függvény használata esetén az összegző kimenete egyben a neuron kimenete is, azonban általában célszerű nemlinearitást vinni a rendszerbe azáltal, hogy az aktivációs függvény a lineáris függvénytől különbözik. Döntési problémák esetén például általában csak az érdekes, hogy a válasz igen, vagy nem, venni, vagy eladni célszerű. Ez esetben csupán a kimenet előjele érdekes, tehát célszerű a szignum függvényt használni. Bizonyos helyzetekben, komplex problémák esetén javíthatja a hatékonyságot, a telítéses lineáris, vagy szigmoid jellegű függvény használata. (Horváth, 2006)



4. ábra Szignum, telítéses lineáris és tangens szigmoid aktivációs függvény

Az elemi neuronokat rétegekbe szokás szervezni, melyből egy hálózatban tetszőleges számú lehet. Nem érdemes egyrétegű hálózatokat használni, mert azok kifejezőereje jóval szerényebb, mint a kettő, vagy ennél több réteget tartalmazó hálózatoké. Az egy réteget tartalmazó hálózat olyan problémára tud csak megoldást adni, melynek bemeneti paraméterterében a mintapontok lineárisan szeparálhatók.

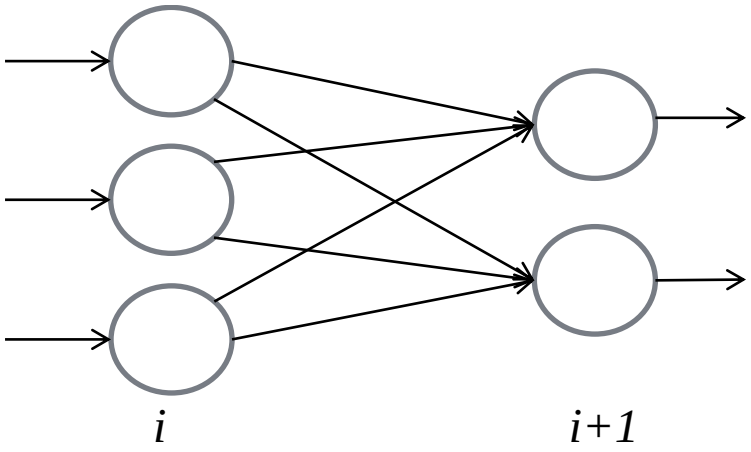
Kettő vagy több réteg alkalmazása esetén a *bemeneti* rétegnek kizárólag bemeneti jel szétosztó szerepe van, processzálást nem végez. Feldolgozás a középső, *rejtett* rétegekben, illetve a *kimeneti* rétegben történik. A bemeneti, illetve kimeneti réteg méretét a használt input, illetve a kívánt output vektor dimenziószáma definiálja.

A többrétegű hálózatok egyik alapvető formája a *statikus,* mely nem tartalmaz sem visszacsatolást *(feedback),* sem késleltetést *(delay)*, így az aktuális kimenet kizárólag az aktuális bemenet értékétől függ. Jellemző felhasználási köre például a számjegy, továbbá arc felismerés, ahol egy bemeneti vektorra kell megmondani, hogy mely előre betanított mintához áll a legközelebb. A másik jellemző típus a *dinamikus,* mely tartalmazhat késleltetést és visszacsatolást is, s ezáltal képessé tehető időfüggő,

illetve szekvenciális minták felismerésére. Ennek következménye, hogy szélesebb körben alkalmazhatók ezek a hálózatok, illetve bizonyos problémák esetén kisebb hibával tudják becsülni a modellezni kívánt folyamatot. Ilyen probléma például a beszédfelismerés, a hibadetektálás, továbbá az idősorok predikciója is.

2.3.1 NARX neurális hálózatok

A többrétegű neurális hálózatok egyik közkedvelt struktúrája az előrecsatolt neurális hálózat *(Feed Forward Neural Network - FFNN),* mely a statikus csoportba tartozik, azaz sem visszacsatolást, sem késleltetést nem tartalmaz. Az ilyen típusú hálózat rétegekbe van szervezve, melyekben az elemi neuronok közt nincsen összeköttetés, hanem a rétegek között előrefele, a következő összes neuronjához. Azaz az *i*-ik réteg összes neuronjának kimenete az összes *i+1*-ik rétegbeli bemenetére van kötve. (Horváth, 2006)



5. ábra Az előrecsatolt neurális hálózat

A dinamikus csoport egyik képviselője a NARX (Nonlinear autoregressive network with exogenous input) egy olyan előrecsatolt neuronháló, mely késleltetést és visszacsatolást is tartalmaz.

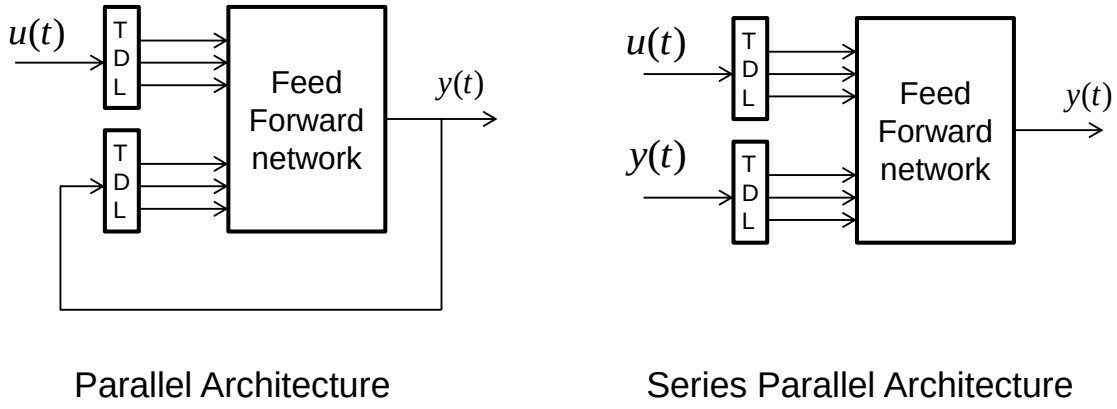
Az általa megvalósított függvény a következőképp írható le:

$$y(t) = f\left(y(t-1), y(t-2), ..., y(t-n), u(t-1), u(t-2), ..., u(t-n)\right),$$

azaz az aktuális kimenetet az előző kimenetek és egy exogén bemenet határozza meg.

A 6. ábra a hálózat kétféle megvalósítását mutatja. A „Parallel Architrcture" bemenetként az előző kimeneteket használja, ezáltal akár több lépéssel előre lehet

jósolni (*multi step ahead prediction*). A „Series Parallel” megvalósítás a valódi megfigyeléseket kapja bemenetként, így nem lehetséges, csak egy lépéses predikció (*one step ahead prediction*), de a bemeneti adatok így pontosabbak és a kimenet is jobban fogja követni a folyamatot. (Beale, és mtsai., 2012)



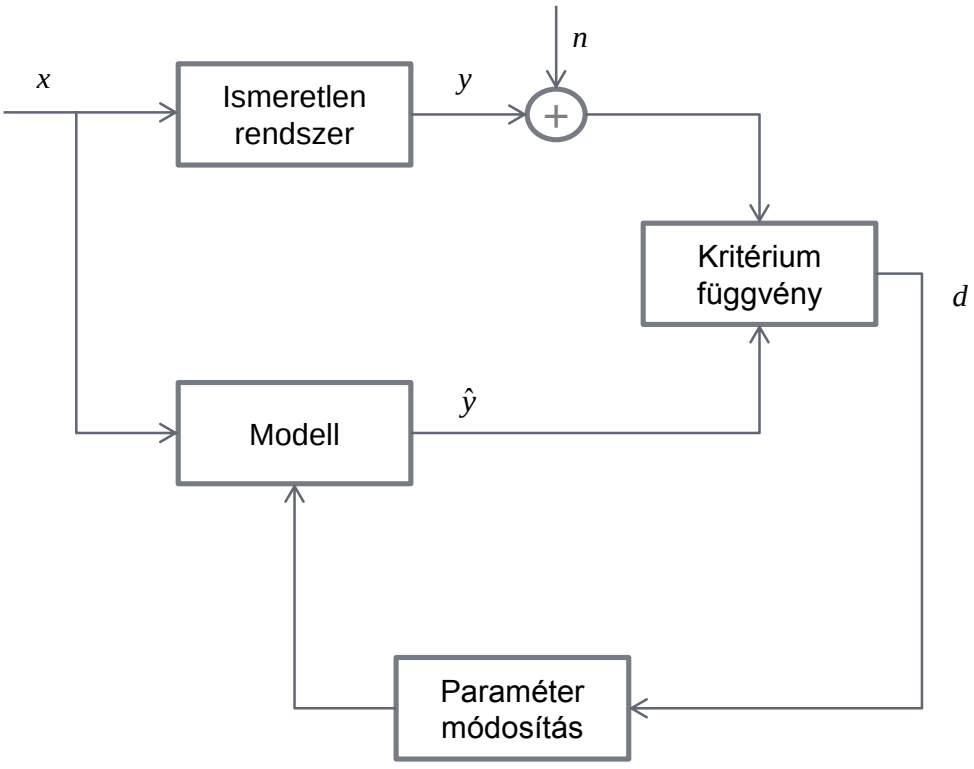
6. ábra A NARX hálózat kétféle megvalósítása

3 A szabad paraméterek beállítása

A megalkotott hálózatot a modellezni kívánt folyamathoz kell igazítani, paramétereit úgy beállítani, hogy valamely választott kritérium (*hibafüggvény*) szerint optimális eredményt adjon. Neurális hálózatok tanítása az élő szervezetek tanulásának analógiájára történik – a környezetükből nyert bemeneti-kimeneti minta párok, illetve szekvenciák alapján próbálják a kívánt folyamatot a legjobban közelíteni. Nagymennyiségű adat használatával elérhető, hogy a hálózat változó körülmények esetén is jól működjön, azokhoz adaptálódjon. Képes az adatok közti rejtett összefüggések feltárására, s ezáltal a jövőbeli állapotokat becsülni.

Az alkalmazott hibafüggvény lehet például az átlagos abszolút vagy a négyzetes hiba. A gradiens alapú tanító eljárásoknál olyan függvényt kell választani, hogy a hibafelület gradiense analitikusan kiértékelhető legyen. Sztochasztikus keresés esetén nincs ilyen megkötés, tetszőlegesen bonyolult, akár nem differenciálható függvény is választható. (Horváth, 2006)

A tanítás folyamatát az alábbi ábra illusztrálja.



7. ábra A tanítás folyamata

Látható, hogy a modellezni kívánt ismeretlen folyamathoz egy általában nulla várható értékű Gauss-inak feltételezett zaj adódik. Az optimalizálni kívánt modell paramétereit úgy módosítja a paraméter beállító, hogy a kimenete és a valós kimenet közötti kritérium függvény értéke minimális legyen. Ez a tanítás általános formája, mely hibaminimalizálás esetén jól alkalmazható, azonban a kereskedési algoritmus kissé más megközelítést alkalmaz, melynek leírása a 3.1 és 3.2 fejezetben kerül részletezésre.

3.1 Tanítási kritériumfüggvények

Egy predikciós modell tanításakor mindig valamilyen hiba minimalizálása a cél. Ez legtöbbször a négyzetes hiba, azonban kereskedés során nem optimális ezt a kritériumfüggvényt alkalmazni, hanem célszerű valamilyen más, a profitabilitáshoz közelebb álló hibát definiálni. Az alábbiakban a négyzetes hiba mellett három kereskedési eredmények szempontjából jobban teljesítő kritérium kerül bevezetésre.

3.1.1 Négyzetes hiba

A négyzetes hiba alkalmazása igen közkedvelt a tanítási eljárások során, mivel kiértékelése nem igényel komoly számítási kapacitást:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{y} - y\right)^2$$

A gradiens alapú szélsőérték kereső eljárások tipikusan ezt használják, többféle hiba-visszaterjesztéses algoritmus alapja, melyet számos alkalmazásban is sikerrel alkalmaztak.

3.1.2 Realizált profit

Gradiens alapú eljárások esetén ugyan nem használható, de a szimulált lehűtés segítségével alkalmazható ez a performancia függvény. Egy választott paramétertér mellett a prediktor szimulálása történik, s ezt a kimenetet kapja a kereskedési stratégia, mely valamekkora hasznot vagy veszteséget fog eredményezni. A tanítás során az a hálózat lesz kiválasztva, amely a tanítómintákon a legmagasabb profitot realizálta.

3.1.3 Sharpe ratio

Portfóliók és befektetési eszközök, stratégiák teljesítményének mérésére használt eszköz, mely a kereskedési eredményt egy kockázatmentes befektetés

hozamával hasonlít össze. Ezen kívül a stratégia kockázatával is kalkulál, amennyiben a szórás magas, a Sharpe ratio alacsonyabb lesz. (Sharpe, 1994) Számítása:

$$SR = \frac{E\left(R_a - R_{rf}\right)}{\sigma}$$

3.2 Tanítási eljárások

A paraméter optimalizációs eljárás számos módon elvégezhető, a dolgozat azonban csak a szokásos gradiens alapú, hibavisszaterjesztéses és sztochasztikus, szimulált lehűtési eljárásokkal foglalkozik.

3.2.1 Gradiens alapú eljárások

A gradiens alapú tanítóeljárásoknál a

$$\frac{\partial C(\boldsymbol{w})}{\partial \boldsymbol{w}} = \mathbf{0}$$

feltételt biztosító paramétervektor meghatározása a cél, ahol $\boldsymbol{w}$ a paramétervektor és $\boldsymbol{C}$ a hibafüggvény. Mivel bonyolultabb esetben – ha például a függvény a paraméterek nemlineáris függvénye – a gradiens kiértékelése nehézségekbe ütközik, olyan iterációs eljárásokat célszerű alkalmazni, ahol a rendszer paraméterei valamilyen algoritmus szerint változnak addig, amíg a gradiens értéke nulla, vagy megközelítőleg nulla nem lesz. Ilyen eljárás a hibavisszaterjesztéses (*back-propagation*) algoritmus is, ahol a hiba a hálózat kimenetétől annak elejéig halad, meghatározva ezzel a szükséges súlymódosításokat. A megoldás lineáris egyenletrendszerek megoldásával adódik, mely feladat nagy sebességgel végezhető bármely mai processzoron.

Az iteratív eljárásoknak a következő követelményeket kell kielégíteniük:

- az eljárás legyen konvergens
- a konvergencia sebesség minél nagyobb legyen
- a hibafelületek széles körére alkalmazható legyen – sokféle probléma esetén hatékony legyen

Sajnos jelen követelmények neurális hálózatok esetén csak ritkán teljesülnek, a hibafelület általában nem kvadratikus, több lokális szélsőértéke is lehet, melyben a tanító algoritmus könnyen elakadhat. (Horváth, 2006)

3.2.2 Sztochasztikus szélsőérték kereső eljárások

A fentebb említett problémán segítenek a sztochasztikus keresőeljárások. Ezek olyan eljárások, melyek úgy biztosítják a lokális minimumpontokból való kilépést, hogy bizonyos valószínűséggel megengedik a rosszabb irányba történő mozgást. Az egyik ilyen technika a *szimulált lehűtés*, mely során a teljes paramétertérből véletlenszerűen (adott eloszlással) választ egy pontot, majd kiértékeli annak hibáját. (Kirkpatrick, és mtsai., 1983)

Természetesen számos más tanító algoritmust lehetne alkalmazni (genetikus algoritmus, vagy heurisztikák), a szimulált lehűtést az egyszerű implementálhatóság miatt választottam.

3.2.2.1 Szimulált lehűtés

A kimerítő keresés jellegű eljárások legnagyobb hátrányát, miszerint ezek nagy paramétertér esetén túl nagy komplexitással rendelkeznek, azáltal kerüli meg, hogy a teret nem szisztematikusan, hanem véletlenszerűen járja be. Az új állapotot az aktuális hőmérséklettől függő valószínűséggel fogadja el, mely a lehűtés során fokozatosan csökken. Az eljárás elején nagy valószínűséggel fogad el rosszabb állapotot (s ezzel lehetővé teszi, hogy kiugorjon lokális minimumokból), míg a hőmérséklet csökkenésével ez a valószínűség egyre kisebb lesz. Az elfogadás valószínűsége a következő alakban írható fel:

$$p = e^{-\Delta E/T},$$

ahol ΔE az aktuális és az előző állapot energiájának (hibájának) különbsége, T pedig a hőmérséklet. Az algoritmus akkor áll meg, ha eléri a kívánt hőmérsékletet, vagy ha egymás után túl sokszor utasította el az új állapotokat. Ez utóbbi a sztenderd algoritmusnak nem része, azonban célszerű bevezetni, mivel úgy csökkenti a szükséges lépések számát, hogy az eredményen alig ront. Ha a hűtés kellően lassú, az algoritmus bizonyíthatóan megtalálja az optimális megoldást. (Kirkpatrick, és mtsai., 1983)

A szokásos *Boltzmann lehűtéshez* nagyon lassú, logaritmikus lehűtési tervet kell használni a következőképp:

$$T_k = \frac{T_0}{\ln k}$$

Az algoritmus C szerű pszeudokódja alább látható.

```
while(!stop) {
    for (int i = 0; i < iterationsAtT; i++){
        oldParameterSet = parameterSet;
        neighbor(); // Generate a neighbor
        currentCost = cost();
        if (currentCost < bestCost){
            bestCost = currentCost;
            bestParameterSet = parameterSet;
            consRej = 0;
        } else{
            if (rand() > exp((prevCost-currentCost)/
temperature)){
                // Reject
                parameterSet = oldParameterSet;
                consRej++;
            }
        }
        prevCost = currentCost;
    }
    cool();

    // Check for stopping
    if(temperature < minTemperature) stop = true;
    if(consRej > maxConsRej){
        stop = true;
    }
}
```

A Boltzmann lehűtés egyik legnagyobb hátránya a lassú konvergencia. A következő pontokban tárgyalt technikák egyrészt a megoldás minőségét, másrészt a konvergencia sebességet hivatottak javítani.

3.2.2.2 Adaptív szimulált lehűtés

A fent részletezett lehűtési technika továbbfejleszthető az úgynevezett újrahűtési (*re-annealing*) módszerrel. Ennek lényege, hogy egy bizonyos számú iterációs lépés után az aktuális paramétertér visszaáll az eddig talált legjobbra, s ebből a pontból újraindul a lehűtés egy magasabb hőmérsékletről. Ezen kívül a szomszédok egy hőmérséklettől függő eloszlás szerint generálódnak, mely kezdetben az egyenletes eloszláshoz hasonló, míg alacsonyabb hőmérsékleten a *Cauchy eloszláshoz* hasonló sűrűségfüggvénnyel rendelkezik:

$$\alpha_{k+1}^{i} = \alpha_{k}^{i} + y^{i}\left(B_{i} - A_{i}\right),$$

ahol α az *i*-ik dimenzióban, a *k*-ik időpillanatban lévő paramétervektor, *A* és *B* az i-ek dimenzióban a paraméter alsó és felső határa,

$$\alpha_{k+1}^{i} \in \left[A_i, B_i\right].$$

A képletben *y* a következő sűrűségfüggvényhez tartozó eloszlásfüggvény:

$$g_T\left(y\right) = \prod_{i=1}^{D} \frac{1}{2\left(\left|y^i\right| + T_i\right)\ln\left(1+1/T_i\right)},$$

melyben *D* a paramétervektor dimenziószáma. T_i az *i*-ik pillanatban vett hőmérséklet. Ezek alapján az eloszlásfüggvény a következőképp adódik:

$$y^i = \operatorname{sgn}\left(u - \frac{1}{2}\right) T_i \left[\left(1+\frac{1}{T_i}\right)^{\left|2u^i - 1\right|} - 1\right],$$

ahol

$$u^i \in U\left[0,1\right]$$

az egyenletes eloszlás szerint generált változó.

A fentebb részletezett fejlesztések eredménye, hogy a lehűtés tovább gyorsítható:

$$T_i\left(k\right) = T_{0i} e^{\left(-c_i k^{1/D}\right)},$$

ahol

$$c_i = m_i e^{\left(-n_i/D\right)}.$$

m_i és n_i segítségével a lehűtés hangolható.

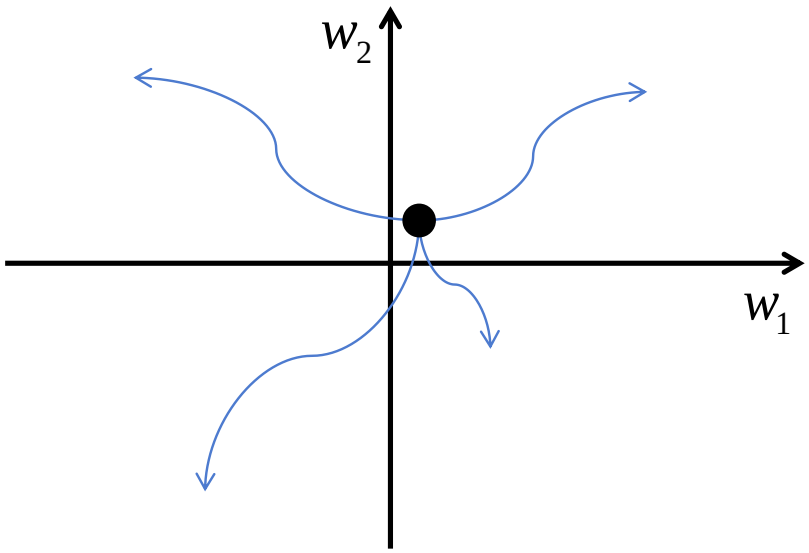
Az algoritmus széleskörű tesztelésen esett át, és bizonyíthatóan jobb megoldást talál, alacsonyabb komplexitás mellett. (Ingber, 1989)

3.2.2.3 Párhuzamos szimulált lehűtés

A véletlen bejárás miatt két futtatás eredménye különböző lehet, hiszen nagy valószínűséggel nem ugyanazt a minimumot találják meg. Ezért a lehűtést célszerű többször lefuttatni egymás után, ezzel javítva a megoldás minőségét. Ezek a futtatások azonban egymás kimenetétől függetlenek, így párhuzamosan is végezhetők. A mai

informatikai architektúrák kifejezetten alkalmasak ilyen jellegű feladatok számítására, hiszen az egyes magok, illetve processzorok egymástól függetlenül működhetnek.

Az egyik lehetséges módszer a lehűtés párhuzamosítására, hogy adott paramétervektorból elindul több lehűtés, melyek mindegyike talál egy minimumot, s ezek nagy valószínűséggel különbözőek lesznek. Ezután a legjobb pontot kiválasztva innen lehet indítani a következő iterációt, mely újra futtat néhány lehűtést.



8. ábra A párhuzamos lehűtés működése kétdimenziós paramétertér esetén

Ezzel a technikával egyrészt a megoldás minősége javul (mert nagyobb valószínűséggel találja meg a globális minimumot), másrészt egy többprocesszoros architektúrán lényegesen gyorsabban találja azt meg. (Ram, és mtsai.)

Az algoritmus C szerű pszeudokódja az alábbiakban látható:

```
for (int j = 0; j < nofParSAIt; j++){
    parfor (int i = 0; i < nofParSA; i++){
        sa.at(i).setParameterSet(parameterSet);
        sa.at(i).anneal();
    }
    bestCost = sa.at(0).getBestCost();
    int bestAt = 0;
    for (int i = 1; i < nofParSA; i++){
        if (bestCost > sa.at(i).getBestCost()) {
            bestCost = sa.at(i).getBestCost();
            bestAt = i;
        }
    }
    parameterSet = sa.at(bestAt).getBestParameterSet();
}
```

3.3 Szakértőegyüttesek alkalmazása

Bonyolult feladatoknál a nagy komplexitás miatt célszerű részfeladatokra dekomponálni, majd a részeredmények aggregálásával képezni a megoldást. A részfeladatok megoldása sokszor könnyebb feladat, mint azt egyben kezelni. Egy lehetséges dekomponálás például a problématér felosztása az egyes szakértők között, de az is lehetséges, hogy mindegyik megoldja a teljes feladatot, de ezek a modulok valamilyen szempontból különböznek egymástól (eltérő neuron szám, rejtett réteg szám). Az egyes megoldások kombinációja bármelyik modul megoldásánál is jobb eredményt nyújthat, mivel az egyes eszközök várhatóan a feladat más-más részénél lesznek jobbak, azaz egymástól *függetlenül* hibáznak. (Horváth, 2006)

Az előrejelző algoritmus ezen utóbbi megközelítést alkalmazza, azaz minden hálózat a teljes feladatot oldja meg, s ezek lineáris kombinációja szolgáltatja a predikciót.

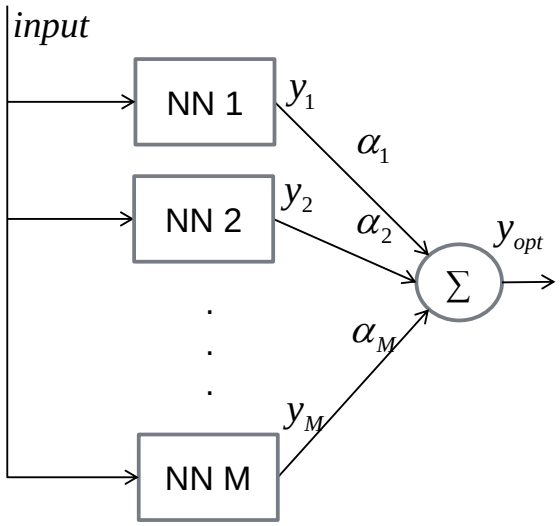
3.3.1 Szakértők optimális lineárkombinációja

A hálók lineáris kombinációjának ötletét az adja, hogy egy probléma megoldása során általában egynél több háló konstruálása történik, majd ezek közül csak a legjobb kerül felhasználásra. Jellemzően a struktúrák terében is történik keresés, azaz többféle háló kerül kipróbálásra, melyek kisebb-nagyobb eltéréseket mutathatnak egymástól. Ha csak a legjobb hálózat kerül alkalmazásra, akkor valójában az a rengeteg járulékos eredmény, ami keletkezik a többi tanulása során, elveszik.

A hálók lineáris kombinációja többféle módon elképzelhető, most azonban csak a legegyszerűbb, fix súlyozású moduláris architektúra kerül kialakításra. A képződő válasz formálisan a következőképp írható le:

$$y_{opt} = \sum_{i=1}^{m} w_i y_i \ ,$$

ahol *w* a választott súlyok, míg *y* az egyes szakértők kimenetei. A következő ábra ezt szemlélteti.



9. ábra Szakértők lineáris kombinációja

Mivel az egyes modulok mindegyike egy *n* rétegű előrecsatolt neuronháló, az egész struktúra tekinthető egy *n+1* rétegű neuronhálónak. Ez a kialakítás azonban több lényeges eltérést mutat. Egyrészt a modulok tanítása egymástól függetlenül történik, továbbá az integráló réteg súlyai is eltérő módon kerülnek meghatározásra. Ez pedig nagyon fontos, hiszen ezáltal a túlilleszkedési hajlam a teljes rendszerre vetítve csökken. (Horváth, 2006)

A továbbiakban a két kulcskérdés tárgyalása következik:

- Hogyan kell megválasztani az egyes szakértőket, illetve
- Hogyan kell beállítani az integráló réteg súlyait.

3.3.1.1 Pontos és különböző szakértők elmélete

A tanított hálózatok kimenetinek nyilvánvaló különbözőséget kell mutatniuk, hiszen amennyiben mindegyik szakértő pontosan ugyanazt a kimenetet szolgáltatja, tetszőleges 0-tól különböző súlyozás esetén az eredő válasz megegyezik az egy hálózat válaszával. Bizonyítható, hogy az egyes szakértőket úgy kell megválasztani, hogy azok

1. Minél pontosabbak legyenek és
2. minél különbözőbb legyen a kimenetük.

Ez az úgynevezett pontos és különböző elv. Különböző és pontos eredmények elérésének számos módja van, az algoritmus egyrészt különböző hálóméreteket konstruál, másrészt a szimulált lehűtés biztosítja, hogy az egyes tanítások eredménye (paraméterek) kellőképpen különbözzenek.

A következő alfejezetekben az integráló réteg súlyainak beállítására alkalmazható eljárások kerülnek ismertetésre.

A feladatot az teszi nehézzé, hogy a súlyokra két fontos megkötést kell alkalmazni:

1. A súlyok összege legyen egységnyi.
2. Minden súly legyen nem negatív.

Az egységnyi súlyozás biztosítja, hogy az eredő kimenet a megfelelő tartományban legyen, a nemnegativitás mögötti gondolat pedig az, hogy az egyes szakértők a feladathoz valamennyire jól „értenek”, azaz az eredő kimenetben nem jelenhet meg egyetlen komponens válaszának ellentettje sem. (Horváth, 2006)

3.3.1.2 Uniform súlyozással megvalósított integráló réteg

A legegyszerűbb, mégis igen hatékony módszer egyszerűen ugyanazt a súlyt rendelni minden egyes hálózathoz. Ezáltal mindkét feltétel triviálisan teljesül, s a kiszámításuk is elenyésző többletköltséggel jár.

Az egyes súlyok ezáltal *m* szakértő esetén a következőképp alakulnak:

$$w_i = \frac{1}{m}.$$

3.3.1.3 Súlyozás szimulált lehűtés segítségével

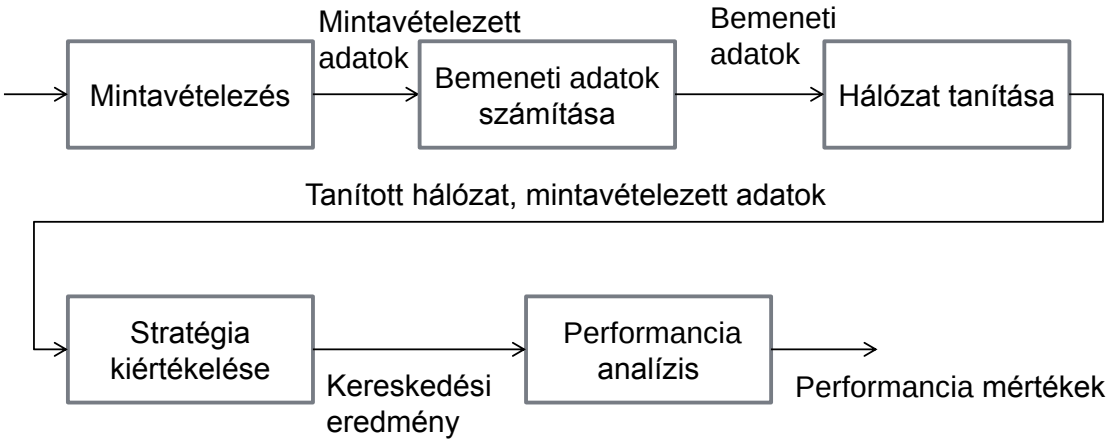
Nemcsak az egyes szakértők, hanem az integráló réteg súlyai is hasonló módon meghatározhatók. Így tetszőleges költségfüggvény alkalmazására nyílik lehetőség, azonban a tanítás költséges, hiszen minden egyes szakértő szimulálására szükség van a tanítás során. A szükséges két feltételt tudja teljesíteni azáltal, hogy olyan pontokat generál, ami megfelel a kényszereknek.

3.3.1.4 Súlyozás kvadratikus programozás segítségével

Lehetőség nyílik négyzetes hiba optimalizálása esetén egy gyorsabb algoritmus használatára. A feladat megfogalmazható egy kvadratikus programozási feladatként, ahol a minimalizálandó költségfüggvény a négyzetes hiba, s a két feltétel beírható az egyenlőtlenségrendszerbe.

4 A feladat számítási folyamata

Ebben a részben a fentebb ismertetett modellek segítségével kialakított számítási modell kerül ismertetésre. A predikció alapú kereskedés működése az alábbi ábrán látható.



10. ábra A modell folyamatábrája

A felhasznált adatsor (FOREX esetén) először *mintavételezésre* kerül, mivel a rendelkezésre álló adatsor a legfinomabb felbontásban van eltárolva, s ebből kell előállítani a kisebb frekvenciás idősorokat. A program ezután a szükséges *kódolt adatokat* készíti el, továbbá különböző tőzsdei indikátorok értékét is kiszámolja, amit majd a hálózat bemenetként megkap. A *tanítás* ezután a fentebb részletezett szimulált lehűtés eljárás szerint történik, melynek eredményeképp előáll az optimális prediktor. Ezután a hálózat a jövőbeli adatokon is futtatásra kerül (melyek a tanítás során nem ismertek, azaz attól független), ezzel megkapva az eljárás profitabilitását. A profit mellett a maximális visszaesés *(drawdown)* és *Sharpe ratio* is kiértékelésre kerül.

4.1 Bemeneti adatok kódolása

Neuronhálók segítségével számos probléma modellezhető, azonban a különböző problémák különböző megközelítést igényelnek. Többrétegű hálózatok hatékonyan alkalmazhatók például függvény approximációra és mintaillesztési feladatokra.

Függvény approximáció esetén egy valamilyen (akár nemlineáris) függvény jövőbeli minél pontosabb meghatározása a cél, a kimeneti értékkészlet általában nem

kötött. Mintaillesztés esetén a pontos függvényérték nem érdekes, hanem bizonyos döntési kérdésekre próbál választ adni. Ilyen kérdés lehet a kereskedés során mikor célszerű egy terméket megvenni, mikor érdemes eladni. Ilyenkor az értékkészlet lényegesen redukált, általában kettő vagy három kimenet lehetséges (Vétel, Eladás, Pozíció tartás). (Soman, 2008) eredményeiben a mintaillesztés jobban teljesít, így ez a fajta megközelítés kerül itt is felhasználásra.

Ahhoz, hogy egy döntési problémaként lehessen értelmezni a kereskedést olyan bemeneti adatok szükségesek, melyek ebben a transzformált (egyszerűsített) térben képesek reprezentálni a szükséges kereskedési döntéseket. Az alábbi egyszerű kódolás kerül alkalmazásra az idősor adatokon:

$$s_t - s_{t-1} > \alpha\left(\frac{1}{n}\sum_{i=2}^{n} s_i - s_{i-1}\right) \rightarrow y_t = 1$$

$$s_t - s_{t-1} < -\alpha\left(\frac{1}{n}\sum_{i=2}^{n} s_i - s_{i-1}\right) \rightarrow y_t = -1$$

$$\textit{Egyébként} \rightarrow y_t = 0$$

ahol y_t a kódolt kimenet, α egy előre megválasztott küszöb paraméter. A kimeneten így 1-el lesz kódolva a nagymértékű áremelkedés és -1-el a nagymértékű áresés. Ha az elmozdulás kicsi, a kimenet 0 lesz.

Ezek alapján a tanított hálózat megpróbálja eldönteni, hogy az aktuális időpillanatban milyen döntést célszerű hozni – 1 esetén vétel, -1 esetén eladás, 0 esetén nincs akció. (Soman, 2008)

4.2 Indikátorok

A fentebb ismertetett kódolási eljárás mellett úgynevezett indikátorok számításával állnak elő további bemeneti adatok. A jobb predikció érdekében az alkalmazott hálózatok ezeket az idősorokat is felhasználhatják.

Az indikátorok a kereskedési termékek múltbeli ár és forgalom adatai alapján matematikai algoritmus segítségével számolt számsorozatok, melyek segítenek a vételi-eladási döntések meghozásában. Általában valamely múltbéli megfigyelt törvényszerűség alapján következtetnek a jövőbeli ármozgásokra. Az alábbiakban

néhány közkedvelt indikátor kerül ismertetésre, azok, amelyeket a kereskedési modell felhasznál. (Kecskeméti, 2006)

4.2.1.1 SMA

Az egyik legalapvetőbb indikátor az egyszerű mozgóátlag (*Simple Moving Average*), mely az aktuális értékét az előző néhány árfolyamadat átlagolásával kapja. Formálisan:

$$SMA_t = \frac{1}{p} \sum_{i=t-p}^{t} s_i \ ,$$

ahol *p* a mozgóátlag periódusa. (Kecskeméti, 2006)

4.2.1.2 EMA

Ez a fajta indikátor az egyszerű mozgóátlagtól annyiban különbözik, hogy a régebbi ár adatokat kisebb súllyal veszi figyelembe, így alkalmazkodva az esetlegesen megváltozott körülményekhez. Míg az egyszerű mozgóátlag nagy késleltetést hordozhat (nagy *p* választása esetén), addig ez a trendfordulókra gyorsabban reagál. Formálisan:

$$EMA_t = EMA_{t-1} + \alpha \left(s_t - EMA_{t-1} \right)$$
$$EMA_1 = s_1$$
$$\alpha = \frac{2}{p+1}$$

4.2.1.3 RSI

Az RSI egy oszcillátor típusú indikátor, mely értéket 0 és 100 között vehet fel. 30 alatti szinteken *túladott,* s vételi jelzést ad, ha felfelé átlépi, míg 70 felett *túlvett,* és árcsökkenést prognosztizál, aláesés esetén. Számítása az árfolyamváltozások alapján történik a következő módon:

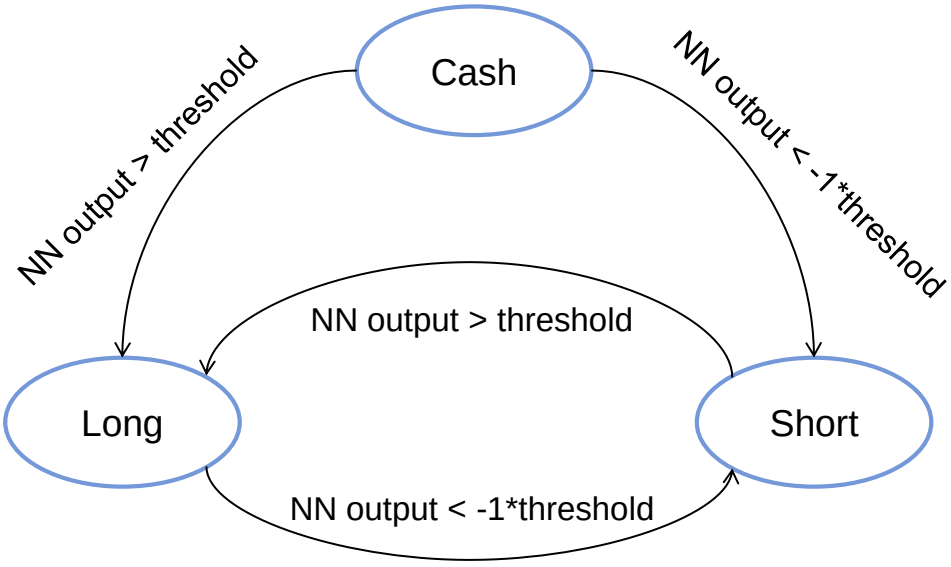
$$RSI_i = 100 - \left[100 / \left(1 + \frac{U}{D} \right) \right],$$

ahol *U* azon záróárak átlaga, melyek a megelőző *p* adat alatt emelkedés mellett következtek be és *D* azon záróárak átlaga, melyek a megelőző *p* adat alatt esés mellett következtek be. (Kecskeméti, 2006)

4.3 A kereskedési stratégia

A predikciós modell kimenete alapján kialakítható egy egyszerű kereskedési stratégia. A neurális hálózat -1 és +1 közötti kimenetet produkál. A bemeneti adatok kódolásához hasonlóan ezt is transzformálni kell, a kimenet alapján meg kell határozni a szükséges kereskedési lépést. Ha a kimenet egy előre megválasztott határérték fölé megy, a pozíció megnyitásra kerül. Ha már van egy ennek megfelelő pozíció (nagy pozitív érték esetén vételi), akkor nincs akció. Eladás irányban is hasonló a működés, ott ugyanezen határérték mínusz egyszeresét kell átlépni a short pozíció felvételéhez, illetve a long pozíció bezárásához. Ha a stratégia a „Cash" állapotból kilép, oda csak az algoritmus végén kerül vissza, azaz mindig lesz valamilyen befektetés, long, vagy short. (Soman, 2008)

Az ismertetett stratégiát az alábbi három állapot szemlélteti.



11. ábra Kereskedési stratégia

5 Implementációs megoldások és architektúrák

A fentebb ismertetett előrejelző algoritmus és kereskedési stratégia többféle architektúrán is megvalósításra került.

Egyrészt a szokásos egyprocesszoros, de többmagos implementációt kapott, melyben a parallel részek *OpenMP* segítségével megvalósítottak. Mivel az egyes neuronháló tanítások, illetve szimulált lehűtések egymástól függetlenek a párhuzamosítás ezek mentén kézenfekvő. Az OpenMP szabványt minden nagyobb fordító támogatja, mellyel egyszerű például a for ciklusok párhuzamosítása

Többgépes környezetben ennél bonyolultabb a helyzet. A számítási csomópontok között nem használható az OpenMP, azonban az MPI *(Message Passing Interface)* olyan kommunikációs primitíveket biztosít, melyekkel üzenet alapú kommunikáció valósítható a csomópontok, s ezen belül a processzek között.

A mai fejlődési irányokhoz igazodva a szoftver egy GPU-s implementációt is kapott, mely a CUDA környezetet használja. Ez a környezet az NVIDIA videokártyákhoz, illetve általános célú GPU-khoz alkalmazható.

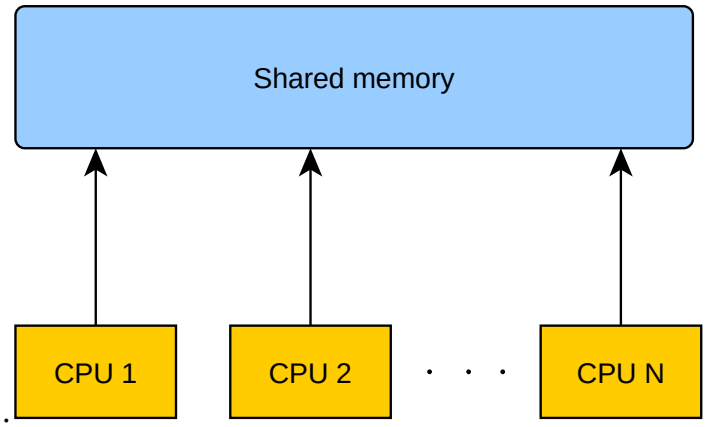
5.1 Párhuzamosítási megoldások

Ebben a fejezetben a két multiprocesszoros és GPGPU alapú párhuzamosítás kerül ismertetésre, melyek a fejlesztett alkalmazásban kerültek felhasználásra. A szoftver nem minden része képes párhuzamos futásra. Ennek oka vagy az, hogy nem lehet ilyen módon futtatni, vagy egyszerűen nem érte meg a fejlesztés plusz költségét, mivel olyan ritkán kerültek lefuttatásra. Az adatbetöltés, indikátorok számítása, mintavételezés egy szálon fut, azonban olyan rövid ideig tartanak, hogy még párhuzamos végrehajtás esetén sem végezne gyorsabban az egész szimuláció.

5.1.1 Osztott memória alapú párhuzamosítás

Amennyiben csak egy gép erőforrásait kell hatékonyan szétosztani, akkor célszerű alkalmazni az osztott memória elvet. Ilyenkor a párhuzamosan futtatandó résznél az operációs rendszer a végrehajtó egységek számához igazodva *szálakat* hoz létre. A szálak közös memórián keresztül kommunikálhatnak, illetve innen vehetik ki a

szükséges adatokat és ide írhatják vissza a számított eredményeket. A megoldás sematikus ábrája alább látható.



12. ábra Osztott memória alkalmazása

Ennek előnye, hogy a memóriát hatékonyan lehet kihasználni, mivel például a tanításhoz szükséges olykor nagymennyiségű, de több szál által is használt adatot csak egyszer kell tárolni. További előny, hogy a szálak létrehozása gyors, operációs rendszer szinten támogatott. Nyilvánvaló hátrány, hogy többgépes környezetben önmagában nem használható, hiszen olyankor nincs egy globális, közös memória. (Quinn, 2004)

5.1.2 Üzenet alapú párhuzamosítás

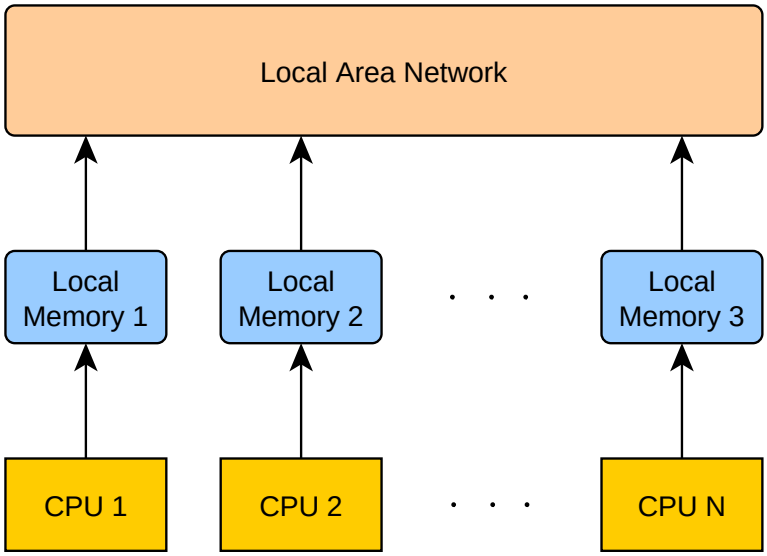
Az üzenet alapú párhuzamosítás némi plusz költség árán megoldja a fenti problémát. Ez esetben minden processzornak megvan a saját lokális memóriája, de azok össze vannak kapcsolva valamilyen hálózaton keresztül (pl. LAN). Természetesen közvetlenül nem férhetnek hozzá egymás memóriaterületeihez, de az *MPI szabvány*, s annak implementációi (pl. OpenMPI) lehetőséget biztosít üzenet alapú kommunikációra.

Minden processzoron a végrehajtandó állomány egy példánya fut, s ezek kommunikálhatnak egymással. Az architektúra sematikus vázlata a 3. ábrán látható.

A megoldás előnye, hogy általa sikerült többgépes környezetre kibővíteni az algoritmust, azonban ennek ára van. Egyrészt a kommunikációra időt kell fordítani, másrészt elérhetővé kell tenni a futtatható állományokat minden csomóponton, továbbá kell valamilyen koordináció a processzek között.

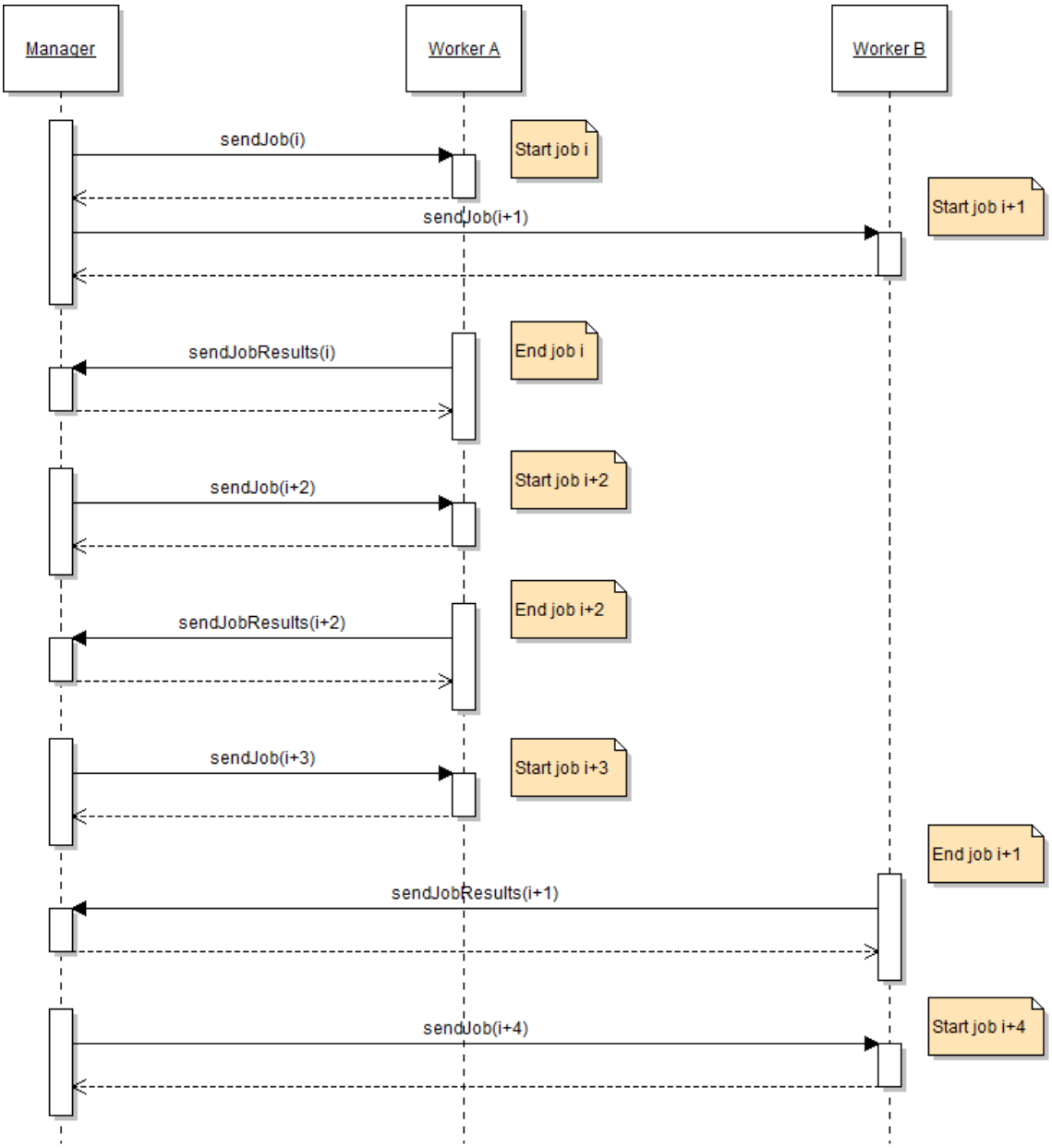
Ezt hatékonyan megoldja az úgynevezett *Manager-Worker architektúra*. Ilyenkor az egyik processz a Manager szerepét vállalja, ő fogja a munkákat szétosztani

az egyes számítóegységek – Worker-ek között. Az architektúra szekvencia diagramja a 14. ábraán látható.



13. ábra Üzenet alapú párhuzamosítás

A diagramon jól látszik a kommunikáció a processzek között. A Manager kioszt egy feladatot az A Worker-nek, ami elkezd dolgozni rajta. Eközben a Manager kioszt egy másik feladatot a B Worker-nek, ami szintén elkezd dolgozni rajta. Amint valamelyik dolgozó befejezi a munkát, üzenetet küld a menedzsernek, aki új munkát oszt ki neki (ha van ilyen). Ez az algoritmus mindaddig maximálisan kihasználja az összes CPU-t, ameddig van végrehajtandó feladat (az utolsó feladat futtatásakor lehet csak alacsonyabb kihasználás). Fontos azonban észrevenni, hogy ebben a formában az egyik végrehajtó egység nem végez „hasznos” munkát, mindössze ütemez. Ez természetesen kiküszöbölhető azáltal, hogy abban az időben, amikor például üzenetre vár, ő is elvállal egy feladatot úgy, hogy a koordináció azért még zökkenőmentesen folyhasson. Nem okoz gondot *heterogén rendszerek* használata olyan értelemben, hogy különböző számítási kapacitású gépek is bevonhatók a klaszterbe, mindössze kevesebb munkát fognak végezni. (Quinn, 2004)

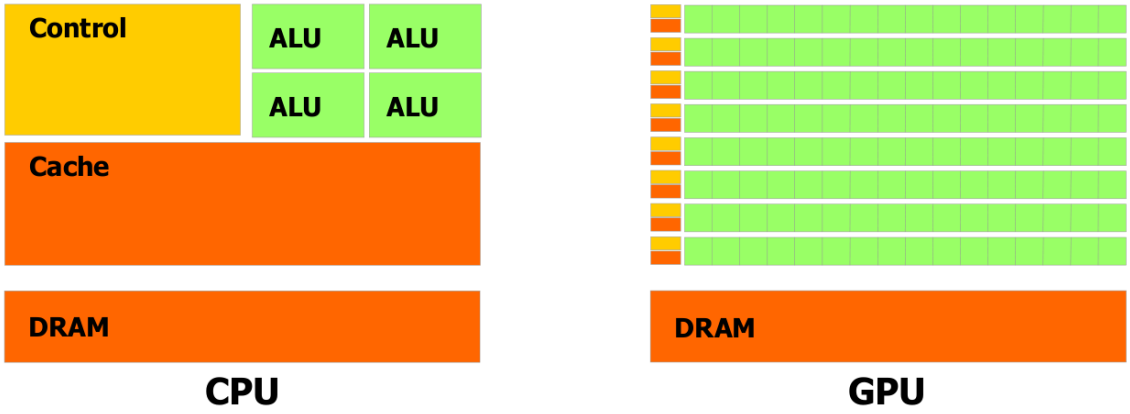


14. ábra Manager-Worker archtektúra szekvencia diagramja

5.2 A GPU számítási teljesítményének kihasználása a kereskedésre

A mai korszerűnek mondható informatikai eszközök 2-8 számítási maggal rendelkeznek, azaz egyszerre 2-8 szál futtatható párhuzamosan. Az egyes magokhoz nagyméretű (többszintű) gyorsítótár *(cache)* is kapcsolódik, melyek a lassabb memória műveletek mennyiségét hivatottak csökkenteni. A processzorok területének nagy részét az első, másod és harmadszintű cache tárak foglalják el, azonban a GPU-k terén kicsit

más a helyzet. Lényegesen nagyobb hangsúlyt fordítanak a valós számolóegységek nagy számára (*ALU* – az ábrán zöld színnel jelölve)



15. ábra CPU - GPU különbségek (NVIDIA, 2012 October)

Jól látható a 15. ábrán, hogy míg a CPU csak néhány számolóegységet tartalmaz (2-4-6 mag), addig a GPU-n ebből lényegesen több van. Egy mai átlagos videókártya több száz, vagy akár ezer ilyen egységet *(CUDA core)* is tartalmaz. Természetesen egy ilyen mag számítási kapacitása nem éri el a CPU egy magjának kapacitását, azonban amennyiben sok adaton kell ugyanazt a műveletet végrehajtani, ez az architektúra komoly sebességnövekedést hordozhat.

5.2.1 Kihívások

Az architekturális különbségekből azonnal kitűnik, hogy a két eszköz nem ugyanolyan jellegű műveletekben lesz erős. A következő pontokban olyan tényezők tárgyalására kerül sor, melyekre a fejlesztés során különös hangsúlyt kell fordítani.

5.2.1.1 Az adatok mennyiségének kérdése

Amennyiben például néhány szám (10-100) összegét, szorzatát kell meghatározni, nyilvánvaló módon nem kihasználható a GPU akár 1000 magja, hiszen mindössze 100 művelet végrehajtása szükséges. Az egyes műveletek természetesen más-más utasításokat, illetve utasítás sorozatokat igényelnek így nehéz megmondani mi számít kis adatmennyiségnek, általánosságban azért elmondható, hogy minél kevesebb az adat, annál kevésbé éri meg a GPU használata.

Szerencsére a mátrixok szorzás művelete már egészen kis méret esetén is sebességnövekedést jelent, nem véletlenül, hiszen a GPU-kat eredetileg is e műveletre

optimalizálták (projekció, forgatás – 4x4 transzformáció mátrixszal történő szorzás) – ez pedig kiemelkedő fontosságú egy neurális hálózat szimulációja (és tanítása) esetén is.

Egy másik fontos elemi művelet – a vektorok összeadása – esetén a kép már kissé árnyaltabb. Egy nemrég megjelent mérés eredménye, hogy bármely méret esetén a CPU (XEON 5660) gyorsabb volt a tesztelt GPU-val (Tesla M2050) szemben. (HPC)

5.2.1.2 A memóriaműveletek időigénye

A GPU a CPU-val ellentétben nem képes közvetlenül használni a rendszermemóriát (RAM). Minden olyan adatot, mellyel számolni fog, át kell tölteni a grafikus egység memóriájába, ami viszont a relatíve lassú *PCI Express* buszon történik.

A memória foglalás-felszabadítás a GPU-n egy összehasonlítás szerint 1-4 nagyságrenddel több időt vesz igénybe, mint a CPU-n. (Mem)
Ez a probléma jól kezelhető megfelelő memória menedzsmenttel, memória újrafelhasználással.

További kihívást jelent a másolási műveletek lassúsága is. Mivel ezek is a PCI Express buszon történnek, célszerű a számuk minimalizálása, illetve a kötegelt memóriamásolás alkalmazása. Elterjedt módszer, hogy minden adatot a program indulásakor felmásolnak a kártyára, majd az összes szükséges eljárást ott valósítanak meg, még akkor is, ha az nem jár sebességnövekedéssel.

5.2.1.3 Kernel hívás overhead-je

A GPU-n párhuzamosan futó programkódok *(kernelek)* hívása nagy járulékos költséggel rendelkezik. Hivatkozván egy másik mérési eredményre, ez az idő – használattól függően – 3-10 mikroszekundum, ami nagyon soknak mondható, mivel ugyanez a CPU-n mindössze 1-2 nanoszekundumot vesz igénybe. (Ker) Természetesen, amennyiben a tényleges munkavégzés során sok időt sikerül nyerni, továbbra is kiemelkedő teljesítményre lehet számítani.

Saját tesztjeim a neuronháló tanítása esetén azt mutatták, hogy egy átlagos adatmérettel indított kernel futása 1-2 mikroszekundum időt vesz igénybe, ami igencsak összemérhető a hívás többletköltségével. Ez pedig azt jelenti, hogy a GPU hasznos munkavégzése távol lesz az optimálistól (10-20%-os kihasználtság várható).

5.2.1.4 Elágazások, ciklusok problémája

Egy algoritmusban az elágazások és ciklusok igen nagy számban fordulnak elő, azonban a GPU-n ezekkel körültekintően kell bánni. A multiprocesszoronkénti egy utasításszámláló miatt, ha két egyedi számító mag különböző kódot akar futtatni, párhuzamos végrehajtás helyett sorosan fognak lefutni, ez pedig teljes mértékben ellehetetleníti a magas kihasználtságot. Ciklusok esetén ez a probléma nem feltétlenül jelentkezik, de ha a kód két különböző úton halad, sorosításra kerül.

5.2.1.5 Lebegőpontos műveletek pontossága

A grafikus kártyákon használt 32 bites float elegendő felbontást biztosít a képernyőn megjelenítendő szín-információk kódolásához, azonban nem feltétlenül alkalmas matematikai-statisztikai műveletek kellő precizitással való elvégzésére. Amennyiben viszont elégséges, célszerű duplapontos számok helyett ezek használata a CPU-s programváltozatban is, mely ott körülbelül 2x-es sebességnövekedést eredményez.

Lehetőség van persze GPU-n is duplapontos műveletek végzésére is, azonban az átlagos videókártyákon (nem Tesla) sebességük negyede, vagy nyolcada az egyszeresen pontos művelethez képest (A Tesla kártyákon fele sebesség érhető el az egyszereshez képest).

5.2.2 A GPU-s programváltozat módosításai

A bevezetett „Profit” függvény iteratív jellege miatt csak alacsony teljesítménnyel volt képes működni a grafikus vezérlőn. Olyan eljárás szükséges, mely még mindig közel áll a profit optimalizálásához, de a GPU-n gyorsan számolható. Az elkészített algoritmus a következőképp működik:

1. A szakértő kimenetét összeveti az abban az időpillanatban érvényes hozammal (negatív is lehet), azaz 1 esetén vételi pozíciót, -1 esetén eladási pozíciót alkalmaz,
2. majd a végén átlagolja ezeket a hozamokat.

Fontos különbségeket hoz ez az új eljárás. Egyrészt nem tud kalkulálni a kereskedési költséggel, mivel ahhoz ismerni kellene az előző időpillanatbeli kereskedési akciókat, de ez pontosan egy ciklussal nyilvánulna meg a kódban, ami a GPU-n lassan tud csak futni. Ez nagyfrekvenciás kereskedésnél igen veszélyes, mivel úgy tűnhet,

hogy nagyon magas a realizált profit (a sok apró kereskedés miatt), holott a másodlagos hatások teljesen felemésztik a profitot. Másrészt ez a stratégia 0 (nem várható lényeges elmozdulás az adott árszintről) kimenet esetén – mivel nem ismeri a múltbéli pozíciókat – úgy tekinti, hogy abban a pillanatban nincs is felvett pozíció. Ez valószínűleg nem nagy probléma, azonban az eredeti stratégia ilyen esetben azzal a pozícióval számolt, ami az előző időpillanatban felvétetett.

5.3 Az elkészült szoftver C++ implementációja

Mind a multiprocesszoros, mind a GPU-s programváltozat ugyanarra a struktúrára épül, mely az látható.

Neuronhálók két osztályát támogatja, az egyszerű *FFNN* az előrecsatolt neuronhálót, míg a *NARX* a NARX típusú hálót valósítja meg. Mindkettő a közös NN ősből származik, s *FFNNLayer* – rétegeket tartalmaznak, korlátlan számban. Minden réteghez tartozik egy kimeneti aktivációs függvény, mely *Linear* – lineáris, *LogSig* – logisztikus szigmoid, vagy *TanSig* – tangens szigmoid lehet. Az *NN* osztályban deklarált virtuális *sim()* – szimulációs függvényt a két leszármazott a működésnek megfelelően megvalósít.

A neuronhálók konfigurációit az *Options* beállítás-struktúra határozza meg. Az osztály kapcsolatban áll egy *XML* alapú betöltővel, melynek segítségével a tesztek paraméterei XML fájlokkal minden futtatás előtt beállíthatók. Ebben megadható a tanítás paraméterei is, továbbá a használt adatsorok beállításai is.

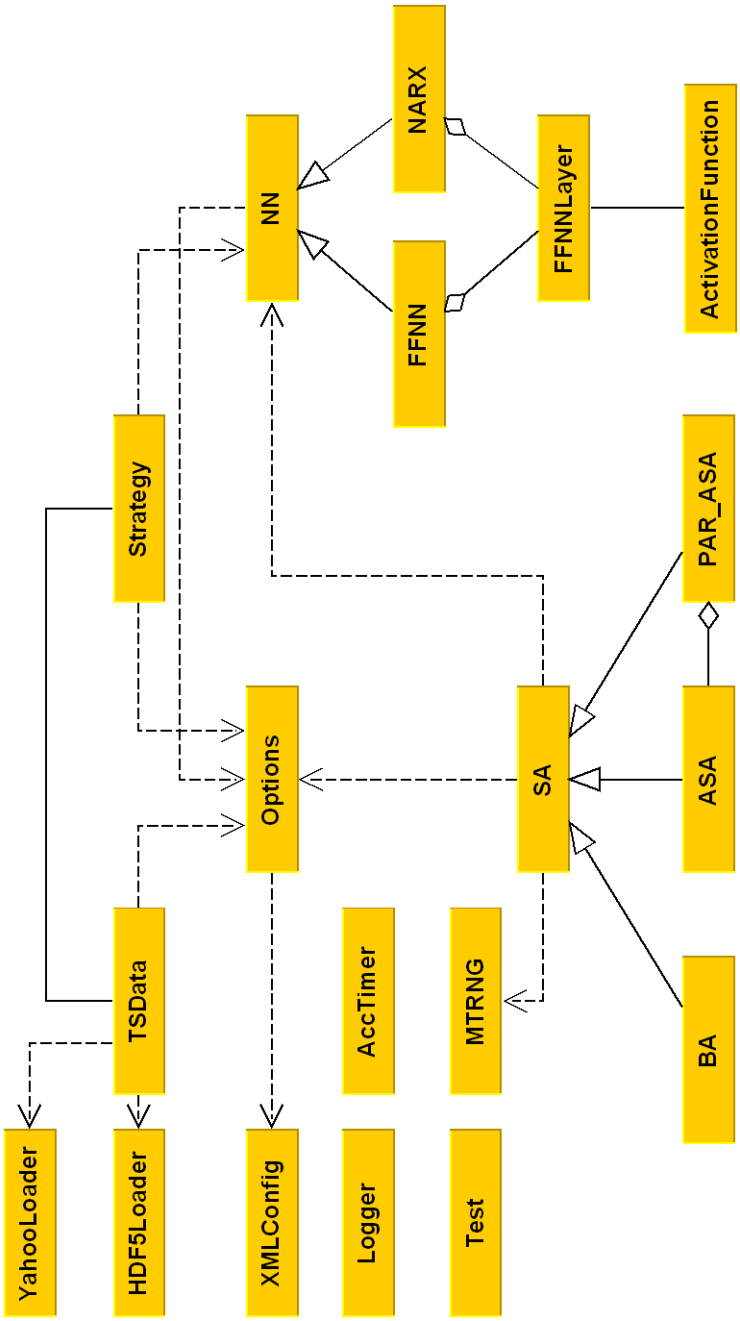
Tanítás terén kizárólag sztochasztikus, szimulált lehűtés alapú eljárás érhető el, azonban abból többféle. A közös SA osztályból leszármaztatott tanítási eljárások a *BA* – Boltzmann Annealing és az *ASA* – Adaptive Simulated Annealing. Az utóbbi párhuzamosított változata is elérhető a szoftverben, *PAR_ASA* néven. A *SA* osztály az *MTRNG* osztályt az ál-véletlen számok sorsolására használja fel, ami a *Mersenne-Twister 19937* generátort alkalmazza, mely a ma ismert legjobb ál-véletlen szám generátor (az elvárt statisztikai tulajdonságokat - például egyenletesség még 623 dimenzió mentén is - leginkább ez teljesíti).

A szükséges tesztadatok (idősorok) be, illetve letöltéséről a *TSData*, illetve a *HDFLoader* és *YahooLoader* komponens gondoskodik. A HDFLoader HDF5 fájlokat képes betölteni a fájlrendszerről, míg a YahooLoader a Yahoo Finance ingyenesen

elérhető napi árfolyamadatait tudja letölteni (ehhez természetesen internet kapcsolat megléte szükséges), s ezeket a többi modul rendelkezésére bocsátja. A *TSData* ezen kívül képes adatokat mintavételezni (finom struktúrájú adatok esetén), továbbá előállítja a szükséges adatsorok kódolt, illetve számított bemeneteit is (RSI, EMA indikátorok).

A *Strategy* osztály az alkalmazott stratégiáért felelős, amiből jelenleg csak az egyszerű mohó stratégia érhető el (ha a háló kimenete egy adott küszöbnél nagyobb – vétel, ha kisebb egy másik küszöbnél – eladás, egyébként tart).

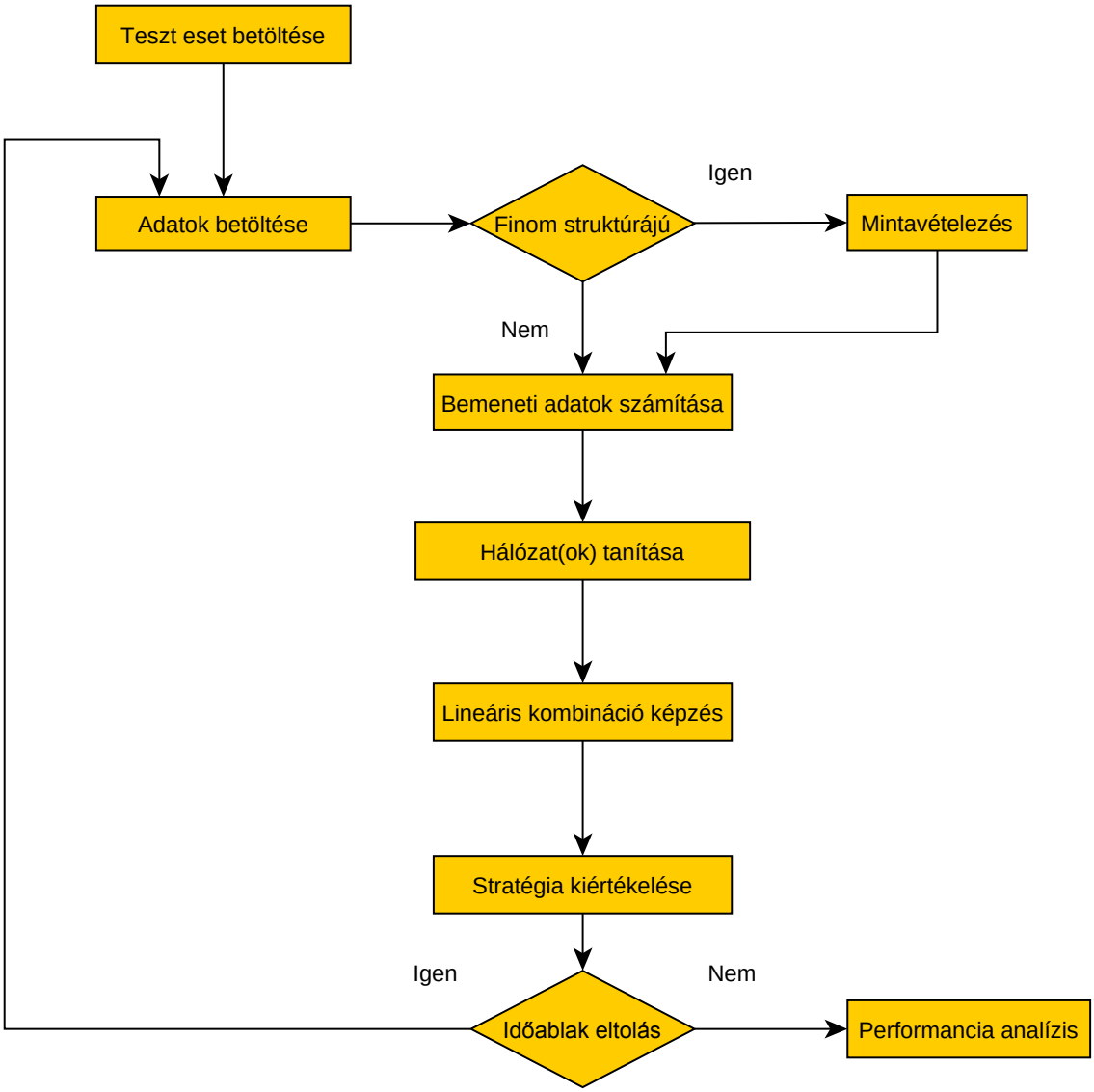
Három további osztály lett definiálva, a *Logger*, ami a naplózásért felelős (naplófájlba írja a futás során esetlegesen keletkező hibákat), az *AccTimer*, mely egy nagypontosságú időmérő osztály (ennek segítségével mérhető a tesztek futásideje), továbbá a *Test* osztály, mely az egységteszteket tartalmazza, normál futtatás esetén nem kerül felhasználásra.



16. ábra Az implementáció osztálydiagramja

A program dinamikus működését a 17. ábrán levő egyszerűsített folyamatábra mutatja. A futtatás elején a program betölti a *config.xml* állományt, mely a szimulációs teszteset leírását tárolja. Ez alapján betölti a megfelelő adatokat (beolvassa a fájlrendszerről, vagy letölti a Yahoo Finance-ről), s mintavételezi amennyiben az szükséges. A bemeneti adatok számítása az idősor adatokból ezután következik, melyek alapján a hálózatok tanítva lesznek. Ha több hálózat lineáris kombinációjának használatára van szükség, akkor ezután meghatározza az integráló réteg súlyait is. Miután a megfelelő kimenetek előálltak a teszt adathalmazra, kiértékeli a stratégia

eredményességét, majd eltolja az időablakot, hogy újabb időszakon vizsgálhassa az algoritmus teljesítőképességét. Ha minden időszakot letesztelt, elmenti a performancia eredményeket egy HDF5 fájlba, ami *MATLAB*-on belül megjeleníthető grafikusan.



17. ábra A működés folyamatábrája

6 A teljesítőképesség vizsgálata

Az elkészített kereskedési rendszer profitabilitás vizsgálata mellett annak számításigénye is kiértékelésre került. Az alábbiakban a tesztek leírásai és eredményeik következnek.

6.1 Futási idő analízis

Az algoritmus sebességének tesztelése saját eszközök mellett a BME klaszter alapú szuperszámítógépén is folyt. A szerver klaszter 30 gépből áll, de ezek közül a tesztekhez csak egy teljes gép került felhasználásra, mivel több géphez egyidejűleg hozzáférni csak hosszú várakozás után lehet (a szoftver természetesen fel lett készítve több gépen való futtatásra). A szuperszámítógép egy számítási csomópontja 2 db Intel Xeon X5660 CPU-t és 48 GB memóriát tartalmaz. Ez a saját teszt hardveremhez (Intel Core 2 Quad Q9400, 4GB RAM) képest körülbelül 3-4x gyorsabb futtatást jelent (4 mag – 12 mag), amit az alábbi táblázat is bizonyít.

	C++ MPI Home PC	C++ MPI Superman	CUDA (Profit ktsg)	CUDA (Opt. profit ktsg)
EUR/USD	372 sec	102 sec	457 sec	84 sec
XAG/USD	407 sec	111 sec	431 sec	81 sec
XAU/USD	387 sec	106 sec	417 sec	83 sec
Átlag	**389 sec**	**106 sec**	**435 sec**	**83 sec**

1. táblázat Sebességtesztek eredményei

A táblázatból jól látszik, hogy sikerült egy olyan szoftvert létrehozni, mely a jól skálázódik a különböző hardvereszközökön, a szuperszámítógép (Superman) 3,5x-eresére gyorsítja a végrehajtást (a tesztek azonos paraméterezés mellett futottak).

A GPU-s eredményekből kitűnik az alkalmazott költségfüggvényre való érzékenység. Amennyiben a pontosabb, másodlagos hatásokat is figyelembe vevő függvényt használja, sebessége igencsak alacsony. A direkt GPU-ra fejlesztett optimalizált profit függvényt azonban már gyorsan képes számolni, így az eredmények már sokkal jobbak. A teljes sebességprofil érdekében ez a függvény a CPU-n is implementálásra került, azonban ott már nem okozott számottevő sebességnövekedést (mindössze 3-4%). Azaz elmondható, hogy ez a környezet valóban a GPGPU eszközöknek kedvez inkább.

Megjegyzendő, hogy mindkét program változat tartogat még optimalizációs lehetőségeket, így az összehasonlítás nem tökéletes, hiszen nem ismeretes, hogy a változtatások milyen hatással lennének a sebességre.

6.2 Profit analízis

A kereskedési teljesítmény kiértékelését az alkalmazott költség függvény vizsgálatával kezdtem. A tesztek megmutatták, hogy a bevezetett új optimalizációs függvény (profit és Sharpe ratio) jobban teljesítenek ugyanazon az idősoron, mint a szokásos négyzetes hiba alapú tanítás. Négyzetes hiba esetén kizárólag a hiba-visszaterjesztéses eljárással került a hálózat tanításra, míg a többi kritérium függvény esetén soros, illetve párhuzamos lehűtés segítségével is tanítva lettek.

	EURUSD	XAGUSD	XAUUSD	Átlag
Négyzetes hiba	0,47%	0,99%	2,20%	**101,22%**
Profit soros	1,70%	0,13%	4,22%	**102,02%**
Profit párhuzamos	1,52%	6,82%	3,90%	**104,08%**
Sharpe soros	1,05%	2,03%	3,18%	**102,09%**
Sharpe párhuzamos	2,12%	5,88%	4,60%	**104,20%**

2. táblázat Alkalmazott költségfüggvény hatása a profitra (1 hónap, 100% kiinduló állapot)

Nem meglepő, hogy a párhuzamos lehűtés jobb megoldást talál, hiszen jóval több lehűtési lépést tud megtenni az algoritmus adott idő alatt. A Sharpe ratio és a profit is nagyjából egy szinten teljesített, az előbbi picit jobban szerepelt. Egy valós kereskedési rendszer esetén azt célszerű választani, ami a befektető szempontjából fontosabb. A profit függvényt akkor érdemes választani, ha csak ez a fontos, nem nagyon számít a kockázat. Ez esetben komolyabb ingadozások fordulhatnak elő az elért hozamokban. Sharpe ratio akkor éri meg, ha érdekes a profit szórása is. Ilyenkor jellemzően kisebb ingadozások jelentkeznek.

Az alábbiakban a szuperszámítógépen végzett tesztek eredményei láthatók. A tesztek két paraméterezés mellett kerültek futtatásra. Az elsőben egy rövidebb, gyorsabb tanítás történik, míg a másodikban a lehűtés lépésszáma az előző duplája. Az alkalmazott tanítási kritérium az elért profit.

Az algoritmus FOREX (*Foreign Exchange*) és S&P 500 adatokon lett tesztelve, FOREX esetén a felhasznált periódus 1 órás, s a teszt 1 évnyi (2011 januárjától 2012

januárjáig) időtartamot ölel át. Minden egy hét leforgása után a modellek újratanítódnak, a legutóbbi két hónap idősor adataival.

S&P 500 esetén napi árfolyamadatok kerültek felhasználásra, s a modellek minden egy hónap leforgása után tanítódnak újra, a két legutóbbi év adataival. A teszt 10 év adatain szimulál.

	Gyors tanítás		Lassú tanítás	
	Végső egyenleg	Maximális visszaesés	Végső egyenleg	Maximális visszaesés
Alcoa	591 %	79 %	1920 %	56 %
Caterpillar	123 %	62 %	146 %	72 %
HP	3985 %	39 %	2146 %	45 %
3M	1709 %	17 %	742 %	20 %
Verizon	281 %	46 %	393 %	42 %
Átlag	**1338 %**	**49 %**	**1069 %**	**47 %**

3. táblázat S&P 500 eredmények (120 hónap, 100% kiinduló állapot)

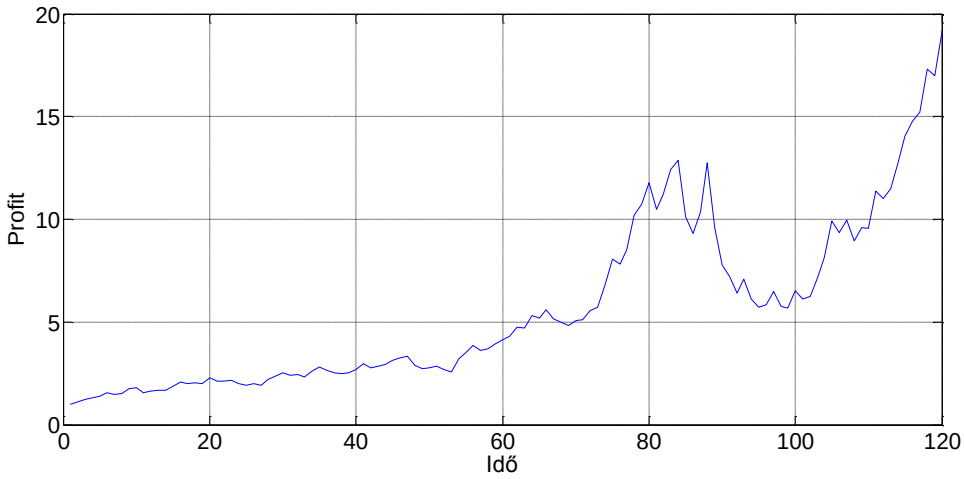
A 10 év alatt megszerzett átlagosan 10x-es profit nem nevezhető rossznak, azonban különös, hogy a kiterjedtebb tanítás nemhogy nem jelentkezik profit többlettel, hanem még romlik is valamelyest a teljesítmény. A grafikonokat vizsgálva elmondható, hogy a prediktorok a 2008-as „gazdasági válság" időszakát leszámítva kiemelkedően szerepeltek, ekkor azonban néhol hatalmas, 60-70%-os visszaesést szenvedtek el.

	Gyors tanítás		Lassú tanítás	
	Végső egyenleg	Maximális visszaesés	Végső egyenleg	Maximális visszaesés
EURUSD	137 %	4 %	142 %	4 %
XAGUSD	327 %	13 %	427 %	16 %
XAUUSD	166 %	6 %	137 %	4 %
Átlag	**210 %**	**8 %**	**235 %**	**8 %**

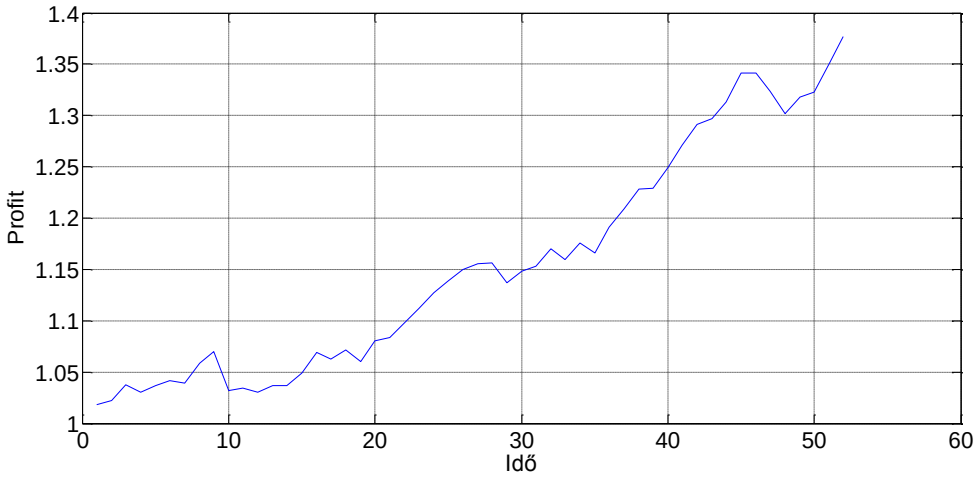
4. táblázat FOREX eredmények (12 hónap, 100% kiinduló állapot)

FOREX esetében az eredmények még jobbak, habár a tesztelt egy éves periódus nem mérhető az előbbi tíz éves periódussal. Az egy év alatt megszerzett 210 illetve 230 %-os profit igencsak jónak számít, s a maximális visszaesés is alacsonynak mondható.

Az alábbi két ábra egy-egy példa a kétféle instrumentum-teszt időbeli eredményességére. Az első ábrán jól látható a 2008-as nagy visszaesés, a második ábrán ilyesmi nem tapasztalható, a hozam egyenletesnek mondható.



18. ábra 120 hónap teszteredménye az Alcoa cég részvényein



19. ábra 52 hét teszteredménye az EURUSD devizapáron

A szuperszámítógép mellett GPGPU eszközön is tesztelésre került a stratégia. Ehhez a módosított profit költségfüggvényt használtam, mivel ez gyorsabban képes optimális eredményre jutni. A tesztek paraméterei úgy kerültek beállításra, hogy ugyanannyi idő alatt végezzen, mint a CPU-s változat. Ezáltal lehetőség nyílt a két platform összehasonlítására kereskedési profit tekintetében. A processzor a szofisztikáltabb célfüggvényt kevesebb lépésszámmal futtathatja azonos időn belül.

	Gyors tanítás		Lassú tanítás	
	Végső egyenleg	Maximális visszaesés	Végső egyenleg	Maximális visszaesés
Alcoa	1240 %	43 %	1650 %	57 %
Caterpillar	175 %	54 %	154 %	71 %
HP	2946 %	37 %	3746 %	44 %
3M	1071 %	24 %	1442 %	22 %
Verizon	143 %	68 %	303 %	41 %
Átlag	**1115 %**	**45 %**	**1459 %**	**47 %**

5. táblázat GPGPU kereskedési eredmények S&P 500-on (120 hónap, 100% kiinduló állapot)

Látható, hogy a GPU-s eredmények az új függvénnyel jobb eredményekkel zártak, mint a multiprocesszoros programverzió. Az elhanyagolt másodlagos hatások nem okoznak gondot, a profit jelentős.

Sajnos nagyfrekvenciás FOREX kereskedés esetén már nem mondható el ugyanez.

	Gyors tanítás		Lassú tanítás	
	Végső egyenleg	Maximális visszaesés	Végső egyenleg	Maximális visszaesés
EURUSD	79 %	26 %	76 %	28 %
XAGUSD	51 %	50 %	54 %	49 %
XAUUSD	66 %	38 %	65 %	40 %
Átlag	**65 %**	**38 %**	**65 %**	**39 %**

6. táblázat GPGPU kereskedési eredmények FOREX-en (12 hónap, 100% kiinduló állapot)

Látható, hogy semelyik termék esetén sem sikerült profitot realizálni. Ez a nagyfrekvenciás kereskedésnek, illetve az alkalmazott, optimalizált költségfüggvénynek tudható be. Ilyenkor az algoritmus sokat kereskedik, azt hiszi, hogy nagy profittal zárja a pozíciókat, azonban mivel figyelmen kívül hagyja a járulékos költségeket, a kereskedés során csak veszít.

Megjegyzendő azonban, hogy kereskedés előtt kiértékelhető a tanítás eredménye a jobb célfüggvénnyel is, ezzel mintegy validálva a teljesítményt. Így legalább nem kerül felhasználásra a rossz minőségű prediktor.

7 Konklúziók és továbbfejlesztési lehetőségek

A dolgozat az algoritmikus kereskedés hardver implementációival és ezek sebesség analízisével foglalkozott. Először bemutattam egy, a véletlen értékpapír árakból álló idősorának megfigyelésén alapuló predikciós algoritmust, mely egy NARX típusú neurális hálóval jósolja a jövőbeli ár-irányokat. Mivel a négyzetes hiba a jövőbeli profit maximalizálása szempontjából nem bizonyult optimális kritériumfüggvénynek, új költségfüggvényeket vezettem be, melyek javítják a predikció teljesítőképességét.

Sajnos, míg a négyzetes hiba alapú optimalizációra már jól bevált algoritmusok léteznek, addig ezen új költségfüggvényekre nem ismert ilyen tanítás. Ezért a szokásos hiba-visszaterjessztéses eljárás helyett sztochasztikus szélsőérték-kereső algoritmusokat használtam. A szimulált lehűtés jól kezeli az ilyen típusú problémákat, azonban a Boltzmann lehűtés túl lassan konvergál az optimális megoldáshoz, ezért egy fejlettebb lehűtés változatot, az adaptív szimulált lehűtést használtam. Tekintettel a mai többprocesszoros és többmagos számítógép architektúrákra, a lehűtésekhez kapcsolódó optimalizálási algoritmusok futtatását egyszerre több végrehajtóegység is végzi, ezáltal a megoldást nemcsak hamarabb kaptam meg, és ennek minősége is sokkal jobb lesz a soros változatnál.

Több szakértő lineáris kombinációjaként sikerült a predikció minőségét javítani, mely egy további optimalizációs problémát jelentett. Erre a feladatra három módszert használtam fel, kimerítő tesztelésen pedig az egyszerű, uniform súlyozású struktúra került.

A bemutatott modellekre nagysebességű, jól skálázódó multiprocesszoros és GPU alapú implementációt is készítettem, mely a kereskedési stratégia teljesítőképességét hivatott elemezni. A GPGPU alkalmazásával sikerült a profiton tovább javítani, bizonyítva ezzel az architektúra előnyét. Ehhez új költségfüggvényt kellett bevezetni, ez azonban csak az S&P 500 esetén bizonyult megfelelőnek, nagyfrekvenciás FOREX adatok esetén sajnos a lassabb, CPU architektúra előnyösebb.

Tesztelést egyrészt saját eszközökön végeztem, másrészt a BME új klaszter alapú szuperszámítógépén *(Superman)*, mind FOREX, mind S&P 500 adatokon, melyek bizonyították az algoritmus profitábilitását. A FOREX-en 1 év alatt elért több mint

200%-os eredmény (100 % kiinduló tőke mellett) jelentős, az S&P 500-on 10 év alatt elért körülbelül 13x-os eredmény, ha szerényebb is, jónak nevezhető, hiszen a tesztek valós körülmények között, nem középárfolyamokon, hanem bid-ask spread jelenlétében kerültek futtatásra.

A további kutatómunka során érdemes lenne az optimalizációt kiterjeszteni a neurális hálózatok struktúrájára is (rejtett réteg neuronjainak száma, rejtett rétegek száma).

Emellett célszerű a bemeneti adatokon egyfajta lényegkiemelést végezni, hiszen a neurális hálózat számtalan bemeneti adatot kaphat, de ezekből csak kevés jellemzi igazán az idősort. Akár hierarchikus struktúrát is lehetne építeni, melyben az első neuronháló lényegkiemelést végez, míg a második kizárólag a predikcióval foglalkozik.

A pénzügyi idősorok elemzésében egyelőre kevéssé használt eljárás a független komponens analízis (ICA), azonban a zajszűrő hatását fel lehetne használni a jobb tanítás érdekében. Egy nemrég megjelent publikáció (Chi-Jie Lu, 2009) sikerrel alkalmazta egy SVM alapú rendszer teljesítményének javítására.

Köszönetnyilvánítás

Ezúton szeretném megköszönni témavezetőmnek, Dr. Levendovszky Jánosnak, hogy a feladat során kiemelkedő szakmai tapasztalatával és tanácsaival segített munkámban.

Köszönöm ezen kívül a Hálózati Rendszerek és Szolgáltatások Tanszéken működő Pénzügyi Számításelmélet és Jelfeldolgozás Laboratórium tagjainak, hogy részt vehettem a projektben és ötleteikkel támogattak, továbbá Dr. Szeberényi Imrének a szuperszámítógép használatában nyújtott segítségét.

Irodalomjegyzék

[Online] // HPC Lab. - http://hpclab.blogspot.hu/2011/09/is-gpu-good-for-large-vector-addition.html.

[Online] // Memory Management Overhead. - http://www.cs.virginia.edu/~mwb7w/cuda_support/memory_management_overhead.html.

[Online] // Kernel Launch Overhead. - http://www.cs.virginia.edu/~mwb7w/cuda_support/kernel_overhead.html.

Bandy Howard B. Mean Reversion Trading Systems [Könyv]. - 2013.

Beale Mark Hudson, Hagan Martin T. és Demuth Howard B. MATLAB Neural Network Toolbox User's Guide. - [hely nélk.] : MathWorks, 2012..

Chan Ernest P. Algorithmic Trading; Winning strategies and their rationale [Könyv]. - [hely nélk.] : Wiley, 2013.

Chan Ernest P. Quantitative Trading [Könyv]. - [hely nélk.] : Wiley Trading, 2009.

Chi-Jie Lu Tian-Shyug Lee, Chih-Chou Chiu Financial time series forecasting using independent component analysis and support vector regression [Cikk]. - 2009..

d'Aspremont Alexandre Identifying small mean reverting portfolios [Cikk] // Quantitative Finance. - 2011..

Fogarasi Norbert és Levendovszky János Novel computational approaches to identifying and trading sparse, mean reverting portfolios [Beszámoló]. - 2011.

Galen Burghardt Ryan Duncan, Lianyan Liu Understanding drawdowns. - [hely nélk.] : Carr Futures, 2003..

Horváth Gábor Neurális hálózatok [Könyv]. - Budapest : PANEM, 2006.

Iebeling Kaastra Milton Boyd Designing a neural network for forecasting financial and economic time series [Cikk] // Neurocomputing. - 1996.. - 10. kötet.

Ingber Lester Very Fast Simulated Re-Annealing [Beszámoló]. - 1989.

Jason Sanders Edward Kandrot CUDA by Example [Könyv]. - 2010.

Júlia Zeiler Hollandia története II. [Online] // A TUDÁS 365+1 NAPJA. - 2013.. - http://www.valtozovilag.hu/t365/tux1218.htm.

Kecskeméti István Tőzsdei befektetések a technikai elemzés segítségével [Könyv]. - 2006.

Kirkpatrick S., Gelatt C. D. és Vecchi M. P. Optimization by Simulated Annealing [Cikk] // Science, New Series. - 1983..

Klimasauskas CC Applying neural networks. Neural networks in finance and investing [Könyv]. - 1993.

Lo Andrew W. The Statistics of Sharpe Ratios [Cikk] // Financial Analysts Journal. - 2002..

Mehta M. Foreign Exchange Markets, Neural Networks in the Capital Markets [Könyv]. - 1995.

Møller Martin F. A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning [Beszámoló]. - 1990.

NVIDIA CUDA C Best Practices Guide [Könyv]. - 2012 October.

Quinn Michael J. Parallel Programming in C with MPI and OpenMP [Könyv]. - 2004.

Quinn Michael J. Parallel Programming in C with MPI and OpenMP [Könyv].

Ram D. Janaki, Sreenivas T. H. és Subramaniam K. Ganapathy Journal of Parallel Simulated Annealing Algorithms // Parallel and distributed computing.

Randall S. Sexton Robert E. Dorsey, John D. Johnson Beyond backpropagation: Using simulated annealing for training neural networks [Beszámoló].

Sharpe William F. The Sharpe Ratio // The Journal of Portfolio Management. - 1994..

Soman Parashar Chandrashekhar An Adaptive NARX Neural Network Approach for Financial Time Series Prediction. - 2008. October.