Közgazdasági Szemle, XLVII. évf., 2000. december (988–1007. o.)

BENEDEK GÁBOR

Evolúciós alkalmazások előrejelzési modellekben – I.

Az előrejelző modelleket készítő közgazdászok lehetőségei igen korlátozottak voltak a számítógép megjelenéséig. A legegyszerűbb regressziós modellek vagy görbeillesztési feladatok is hosszú számítási időt igényeltek, a nagy adatbázisok kezelése pedig gyakorlatilag lehetetlen volt. Optimalizálási probléma esetén csak a legegyszerűbb esetekkel tudtak foglalkozni (például a lehetséges megoldások halmaza konvex és a célfüggvény konkáv). Nem véletlen tehát, hogy mind a statisztikában, mind az operációkutatásban a lineáris modellek terjedtek el. A számítógép megjelenésével és villámgyors fejlődésével (számítási sebesség, tároló kapacitás) lehetőség nyílt arra, hogy a modellek megalapozásához szükséges információkat óriásadatbázisok szolgáltassák. Egyre bonyolultabb összefüggéseket keresve, egyre több változót bekapcsolva, az elemzőknek rá kellett ébredniük arra, hogy „a világ nem lineáris". Nagyon bonyolult modellformát választva a hagyományos optimalizáló eljárások kudarcot vallanak (például lokális optimumban állnak meg), ismeretlen modellforma esetén pedig nem lehetett definiálni a feladatot (nincs célfüggvény).*

Az elmúlt harminc évben rengeteg kísérlet történt a fenti probléma megoldására. Az egyik legsikeresebb megközelítésnek az evolúciós elméletek alkalmazása bizonyult. Ez a tudományterület borzasztóan fiatal, rengetegen próbálják fejleszteni és alkalmazni, s ennek következtében óriási a különböző algoritmusoknak és alkalmazási területeknek a száma (*Maren–Harst* [1990], *Mammone* [1991] és *Winter* [1995]). Sok esetben azonban hiányoznak vagy hibásak a bizonyítások, és indokolatlanok vagy helytelenek az alkalmazások. Jelen tanulmánynak az a célja, hogy bemutassa az evolúciós eljárások elméleti hátterét és alkalmazási lehetőségeit közgazdasági előrejelzési feladatok esetében. Szeretnénk elérni, hogy az olvasó ezután

– könnyebben eligazodjon az evolúciós algoritmusokat felhasználó cikkeken, modelleken;
– meg tudja különböztetni az értelmes alkalmazásokat a „divattól";
– fel tudja ismerni, hogy mikor lehet és érdemes egy feladatot evolúciós módszerekkel megoldani;
– szükség esetén kipróbálja/alkalmazza az újonnan nyíló lehetőségeket.

A részletes algoritmusok és kódok ismertetései meghaladnák e tanulmány kereteit. Erre azonban nincs is szükség, ugyanis egy elemző közgazdásznak nem feladata, hogy bonyolult programokat készítsen. Szükség esetén ezek a programok rendelkezésre állnak. A megadott irodalomjegyzék segítségével azonban az érdeklődők a forráskódokhoz is könnyen hozzáférhetnek.

* A tanulmány a T25372. számú OTKA-kutatás keretében készült. A szerző köszönettel tartozik *Hideg Évának, Lipovszki Orsolyának* és *Pataki Attilának* a dolgozat megírásában nyújtott segítségért és ösztönzésért.

Benedek Gábor, Budapesti Közgazdaságtudományi Egyetem matematikai közgazdaságtan és ökonometria tanszék (benedek@sunshine.bke.hu).

Az evolúciós eljárásoknak a közgazdaságtanban amiatt van különleges jelentősége, hogy a közgazdaságtan az összes szereplőt haszonmaximalizáló, azaz optimalizáló szereplőnek tekinti. A legtöbb modell eleve feltételezi, hogy a szereplők ezt az optimális viselkedést az összes rendelkezésére álló információ alapján, analitikusan meg tudják határozni, és mindvégig ezt alkalmazzák. A valóságban azonban a környezet állandóan változik. (Már az is változást okozhat, ha a szereplők nem egyszerre hozzák meg amúgy optimális döntéseiket.) A szereplők a valóságban állandóan *tanulnak*, *alkalmazkodnak*, *fejlődnek*. Az evolúciós eljárások pontosan ezeket a folyamatokat szimulálják.

A két részben megjelenő tanulmány három fejezetből áll. Az elsőben az evolúciós elméletek általános tulajdonságait tárgyaljuk. A történeti és elméleti ismertetés mellett bemutatunk néhány konkrét alkalmazást is, különös tekintettel a közgazdasági alkalmazásokra. Végezetül összefoglaljuk azokat a feltételeket, amelyek esetén érdemes evolúciós algoritmusokkal próbálkozni. A második fejezetben két algoritmus – a *genetikus algoritmus* és a *neurális háló* – részletesebb ismertetésével foglalkozunk. Az ismertetéshez egyszerű szemléltető példákat alkalmazunk. A fejezet utolsó pontjában néhány további érdekes algoritmusról adunk rövid ismertetést. A Közgazdasági Szemle januári számában megjelenő második rész tartalmazza a harmadik fejezetet, amely célja, hogy egy valós problémán keresztül bemutassa, hogyan alkalmazhatók előrejelzési modellekben az evolúciós eljárások. A példa a részvényárfolyamok előrejelzésének problémája, és az ehhez kapcsolódó optimális portfólió-, illetve opcióvásárlási stratégia kialakítása. A probléma szemléltetéséhez szükséges adatbázis szintetikus.[1]

Evolúciós elméletek

Az ötlet

A szakirodalomban rengeteg evolúciós módszerrel, algoritmussal és alkalmazással találkozhatunk. Ezek nagyon különbözhetnek egymástól, aszerint, hogy milyen evolúciós modell szolgál az algoritmus alapjául, vagy akár ugyanolyan modell esetén milyen mechanizmus valósítja meg a modellt. Minden modellben közös azonban az, hogy valamilyen szinten a természet optimális kiválasztódási folyamatát próbálják utánozni.

Mit csinál a természet? Tegyük fel, hogy adott egy populáció számára valamilyen kezdeti állapot, valamint azt is, hogy ez az állapot optimális a populáció és a természet számára, például nem hal ki vagy nem szaporodik el túlságosan a populáció. Ebben az állandósult állapotban nincs fejlődés, nincs tanulás, és tökéletes az egyensúly, abban az értelemben, hogy nincs semmilyen olyan erő, ami kimozdítaná ebből az állapotból. A következő lépésben azonban megváltoztatjuk a populáció környezetét, ami valamilyen negatív hatással van rá. A populáció egyes egyedei elpusztulnak, mások életben maradnak. A következő generációba az életben maradtak gyerekei kerülnek, azok is szembesülnek a negatív hatással, elpusztulnak vagy életben maradnak, szaporodnak stb. Az egyedek természetesen nem egyformák, kis tulajdonságokban különböznek. E tulajdonságok azonban meghatározzák, hogy a negatív hatással szemben mennyire ellenállók. A jobban ellenállók életben maradnak, és képesek továbbörökíteni ezt a tulajdonságot. Sőt, a populáció akkor sincs feltétlenül halálra

[1] A tanulmányban felhasznált szoftverek a következők voltak: Pascal, C, Sugal algoritmus könyvtár (C), Clementine, SPSS. A tanulmányban szereplő néhány egyszerű példát érdemes olvasás közben, a jobb megértés céljából kipróbálni, ezért ezek a mindenki számára könnyen hozzáférhető Microsoft Excel program segítségével rekonstruálhatók.

ítélve, ha ez a tulajdonság (vagy ezek a tulajdonságok) a kezdeti generáció egyetlen egyedében sem fordult(ak) elő. Lehetőség van ugyanis a mutációra, azaz olyan öröklődésre, ahol a gyerek nem minden tulajdonsága vezethető vissza szülei, nagyszülei stb. tulajdonságaira, hanem vannak véletlen megváltozások is. Jó esetben, egy sokadik generációban újra helyreáll az egyensúly, ez az új állapot (tulajdonság-összetétel) lesz optimális.

Tekintsük ezt a problémát egy hagyományos optimumfeladatnak, amelyben az egyedek különböző tulajdonságai adják a lehetséges megoldások halmazát. (Minden egyed egy-egy lehetséges megoldás). Minden lehetséges tulajdonsághoz a természet „megmondja" mennyire jók az adott egyed túlélési esélyei, azaz adott egy, az összes lehetséges tulajdonság halmazán értelmezett függvény. A populáció az utolsó generációban megtalálja a függvény maximumát. A folyamatban az a különös, hogy anélkül találja meg az optimumot, hogy ismerné a célfüggvény alakját (tulajdonságait). (Hozzátesszük, hogy ezáltal a fejlődés által viszonylag gyorsan találja meg a populáció a számára optimális állapotot. Ennek azért van nagy jelentősége, mert ismeretlen célfüggvény esetében a legegyszerűbb megoldás az összes lehetséges eset kipróbálása, és az optimális megoldás kiválasztása, ez azonban óriási számítási időt vehet igénybe.) Emiatt a szempont miatt érdekes számunkra az evolúciós eljárás, hiszen – mint a bevezetőben elmondtuk – olyan problémákhoz keressük a megoldást, amelyben a célfüggvény nagyon bonyolult, akár ismeretlen tulajdonságú.

Feltehetjük a következő kérdést: nem véletlenül találta-e meg a populáció az optimális tulajdonságokat? A válasz pozitív, a véletlennek óriási szerepe van fejlődésben. Viszont ez a véletlen nem „vak". Nem össze-vissza bolyongunk egy számunkra teljesen ismeretlen helyen, hanem fejlődési irányokat, utakat derítünk fel. Az útról persze többször is letérünk, de a lényeg az, hogy minden következő lépést valamilyen módon az előző lépések helyessége befolyásol. Azt lehet tehát mondani, hogy a tanulás vagy fejlődés irányított.

Ez a megközelítés az evolúciós eljárások egyik fő iránya. Célja az optimalizálás, az algoritmust pedig *genetikus algoritmus* néven említi a szakirodalom. Természetesen sok minden még nincs tisztázva; hogyan zajlik az öröklődés, hány egyed van egy populációban, hány generáción keresztül figyeljük meg a populációt stb. Ezekre a kérdésekre a genetikus algoritmussal foglalkozó rész adja meg a választ.

Nézzük most tovább az evolúciós eljárásokat! Sikerült találni egy olyan eljárást, ami egy diszkrét változásra egy hosszú folyamat után optimális választ ad. Sok esetben azonban nagyon gyakori a változás, és nagyon rövid idő áll rendelkezésre ahhoz, hogy a változásra a helyes válaszreakciót megadjuk. Azért nehéz a gyors válasz, mert nem ismerjük azt a függvényt, ami minden egyes környezeti állapothoz megadja az optimális (vagy megfelelő) megoldást. A természetben rengeteg olyan eset fordul elő, amikor nagyon gyorsan kell válaszreakciót adni. Ilyen például az arcfelismerés, ahol a szembe érkezőről nagy mennyiségű információt kell valamilyen függvénynek feldolgozni, és végeredményül az adott személyt meghatározni. A természetben az ilyen típusú függvények keresése és állandó módosítsa, javítása a *tanulás*, így kézenfekvő, hogy a modellezéshez az agy működését (neuronok működését) használták. A *neurális háló* tehát olyan tanulóalgoritmus, amely nagy mennyiségű példa alapján egy adott inputhalmaz és outputhalmaz közti összefüggést próbálja megtalálni. Sok esetben az előrejelző modellek készítésekor is a fenti problémával szembesülünk. Nagy mennyiségű adattal rendelkezünk, de bonyolult és sokféle összefüggés lehet az adatok között, és a modellezést az adatok zajossága is bonyolíthatja. Ebben az esetben az összefüggések keresését érdemes lehet a neurális hálóra bízni. A neurális hálóval a későbbiekben részletesen foglalkozunk.

Végül meg kell jegyeznünk, hogy bár a genetikus algoritmus és a neurális háló az evolúciós elméletek két talán legfontosabb pillére, rengeteg más evolúciós módszer létezik, és nagyon sok „vegyes" eljárás található – mind algoritmus, mind alkalmazás szinten. Így például a neurális háló nemcsak függvénykeresésre, hanem optimalizálásra is alkalmazható, a genetikus algoritmus felhasználható a neurális háló hierarchiájának kialakításához, hagyományos optimalizáló módszereket (például gradiensmódszer) és heurisztikus optimalizáló eljárásokat (például *simulated annealing*) kapcsolhatunk össze a genetikus algoritmussal stb. Nincs lehetőség ezek teljes bemutatására, csupán megemlítünk néhány eljárást.

Miért kell a számítógép?

A legkézenfekvőbb válasz az, hogy a számítógép rendkívül nagy sebességgel képes számításokat elvégezni, *nagyméretű adatbázisokat tárolni,* illetve ezen adatbázisokon különféle műveleteket végrehajtani (például: keresés, rendezés stb.). Ennél azonban sokkal nagyobb a számítógép és a *programozás* megjelenésének a jelentősége. Ennek illusztrálására vegyük például a klasszikusnak számító utazóügynök-problémát.[2] Ahhoz, hogy az optimális utat megtaláljuk, minden lehetőséget végig kell számolni. Ezt az eljárást *teljes leszámolásnak* nevezzük. Ez n darab város esetében összesen $(n\text{-}1)!$ darab lehetőséget jelent. Ha „kézzel" látunk neki a feladatnak, akkor 3-4 várost szinte ránézésre meg tudunk oldani, 7-8 városnál nagyobb feladathoz viszont hozzá sem kezdünk. Ha a számítógépre bízzuk a keresést, akkor a számítási idő 14-16 város esetében éri el azt a határt, amelyet már nehezen tartunk elfogadhatónak. A valóságban azonban sok olyan analóg feladat van, ahol a „városok" száma több száz is lehet. Úgy tűnik, ez a számítógép felfedezése ellenére is reménytelen a feladat.

A számítógép azonban óriási teret enged a *visszacsatolásnak*. A számítások eredményei elraktározhatók, összehasonlításokat, rendezéseket hajthatunk végre, melynek segítségével olyan irányba mozdulunk el, ahol az optimális út megtalálásának nagyobb a valószínűsége. Ebben az esetben az egyszerű mechanikus számítások helyett bonyolultabb algoritmusok használatáról beszélünk. Természetesen feltételezhetjük azt, hogy „kézi számolás" esetén is valamiféle algoritmust követünk, ám a korábbi eredményeket tárolni, újra átnézni a számítógépnél senki sem tudja hatékonyabban.

Sok esetben – így az utazóügynök-problémánál is – az algoritmusok alkalmazása nem garantálja, hogy az optimális megoldást kapjuk eredményül. Viszont minél több időt engedünk az algoritmus számára, annál közelebb kerülünk az optimális megoldáshoz, és már viszonylag rövid idő alatt is elfogadható – az optimum közelében levő – eredményt kapunk.

A numerikus módszerek alkalmazásának még egy előnyét/hátrányát szokták hangsúlyozni. Ez a számítógép által elkövetett kerekítési hiba, amelyet az okoz, hogy egy végtelen törtet csak véges hosszúságig képes a számítógép tárolni. Így a számítógép esetében sajnos nem igaz, hogy $a + b = b + a$. Emiatt – s ez a hátrány – vigyáznunk kell akkor, amikor ugyanazon kiinduló értékkel nagy számú műveletet hajtunk végre,[3] mert a kerekítési hibák kumulálódnak, és a végeredmény nagyon pontatlan lehet. A kerekítési hibák előnye viszont, hogy az instabil egyensúlyi állapotokból bizonyos idő elteltével kitér a rendszer (lásd *Simonovits* [1998] és *Balla–Benedek* [1999]).

[2] Az utazóügynök-feladatban n darab várost kell egy ügynöknek úgy bejárni, hogy minden várost csupán egyszer érint, és a megtett összes út a lehető legrövidebb. Az egyes városok közti távolságok adottak.

[3] Például kamatoskamat-számítás.

Alkalmazások közgazdasági területen

Ebben a pontban néhány tényleges, illetve lehetséges alkalmazást mutatunk be. Ezek a példák szemléltető célt szolgálnak, és nem törekszünk sem a teljességre, sem pedig az alkalmazások valamilyen csoportosítására, klasszifikációjára. Nézzünk először két klasszikus – ám nem közgazdasági – alkalmazást!

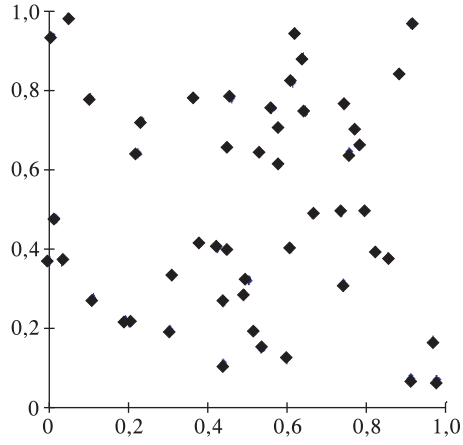
A számítógéppel dolgozók régi nagy álma, hogy képesek legyenek a számítógépet nem kézzel, hanem hanggal irányítani, diktálni tudjanak a számítógépnek, azaz a gép felismerje a beszédet. A feladat több szempontból is nehéz. A beérkező hang rengeteg információt tartalmaz, ebből nem mindenre van szükség a beszéd felismeréséhez. Nehéz azonban eldönteni, hogy mit lehet „levágni" vagy „elsimítani". A betűket egyenként felismerni lehetetlenség, de sokszor a szavakat is összemossuk az élőbeszédben. Sok szónak más az írása, mint a kiejtése. Minden ember egy kicsit máshogyan beszél, más magasságon, más tempóban, más kiejtéssel. A nehézségeket még hosszan sorolhatnánk. A megoldást többek között neurális hálók alkalmazásával keresik. A feladat egyébként kifejezetten „neurális hálónak való", hiszen analóg feladat az emberi agy beszédfelismerésével. A neuronoknak azt a „leképzést" kell megtalálniuk, amely minden beérkező beszédmintára a megfelelő írott szöveget adja. A számítástechnika jelenlegi állása szerint még nem megoldott a beszédfelismerés, de részsikerek már vannak. Ez azt jelenti, hogy háttérzajmentes környezetben, jól tagolt, meghatározott – néhány száz szavas – szókincsből álló beszédet képes felismerni a számítógép, abban az esetben, ha a beszélő megfelelő ideig tanította a saját beszédstílusára a neurális hálózatot. A fentihez kissé hasonló írott szövegfelismerést azonban gyakorlatilag már sikerült megoldani, és az ezt megvalósító programok a felhasználók rendelkezésére állnak. (Részletes ismertetés található*: Institute of Electrical an...* [1989] és *Holmes* [1993]).

A genetikus algoritmus alkalmazására számtalan példa található. Ezek közül egy kiragadott példa az automata sebességváltó. A feladat az, hogy optimális időpontokban kapcsoljunk egy-egy fokozattal előre gyorsításkor, azaz úgy váltsunk sebességet, hogy mondjuk 100 km/órára a lehető legrövidebb idő alatt gyorsuljunk fel. Bár az optimalizálandó függvény formája igen bonyolult – hiszen az adott fokozatok nyomatékgörbéi erősen függnek attól, hogy mekkora sebességnél történt a kapcsolás –, a genetikus algoritmus számára könnyen és gyorsan megoldható a probléma.

Nézzünk néhány alkalmazást közgazdasági területről! Elsőnek mindjárt megismételhetjük a már említett utazóügynök-problémát. A példafeladatban 50 várost kell a lehető legrövidebb útvonalon bejárni (lásd *Pham–Karaboga* [1998]). Ha a biztos optimummegoldást keressük, akkor 49! (azaz több mint 6×10^{64}) lehetőséget kellene végigszámolni. Ez még a mai leggyorsabb számítógépek számára is túlságosan hosszú feladat. A jelen esetben mind a genetikus algoritmus, mind a neurális háló alkalmazható a legrövidebb út megtalálásához. Mindkét algoritmus nagyon rövid – és közel azonos – idő alatt adott eredményt. A genetikus algoritmus által megtalált legrövidebb út hossza 5,58, a neurális hálóé 6,61 volt. Jelen esetben tehát a genetikus algoritmus jobb eredményt adott. Nem lehetünk azonban biztosak abban, hogy a megtalált 5,58 hosszúságú út a legrövidebb, csak azt tudjuk, hogy igen jó – optimumhoz közeli – megoldást kaptunk. Az egyes algoritmusok maguk is „tudják", hogy nem biztos, hogy optimális megoldáshoz jutottak, ezért a keresés, illetve a tanulás befejeződésének (leállásának) különböző kritériumokat adhatunk.

A feladat kicsit módosított változatát több, szállítással foglalkozó vállalat is alkalmazza. Egy olajtársaság számára például fontos kérdés, hogy meghatározott számú telephelyeiről milyen módon juttatja el a benzint a kutak számára. Itt nem egyetlen

1. ábra
Térkép (50 város)



„ügynökről" van szó, hanem többről, hiszen a társaságnak több olajszállító kamion áll a rendelkezésére. Ezek átlagos sebessége és tárolókapacitása is különböző lehet. A feladat az, hogy a telephelyekről a lehető legalacsonyabb költséggel szállítsák az üzemanyagot a kutakhoz. A problémát genetikus algoritmus segítségével oldották meg (lásd *Himanen* [1998]).

A következő alkalmazást mobiltelefon-társaságoknál fejlesztették ki. A probléma az, hogy a kommunikációhoz szükséges rádió adó-vevő központokat milyen sűrűn és hova helyezzék el. Ha egy körzet nincs lefedve, akkor onnan nem lehet telefonálni, ha pedig nincs elég sűrűn lefedve – megfelelő kapacitással –, akkor csúcsidőben nem lehet telefonálni (*túlterhelés*). Látható, hogy a feladat különösen bonyolult, hiszen nemcsak a földrajzi adottságokat kell figyelembe venni, hanem a lehetséges használat mértékét is. A problémát többek között neurális hálózattal és genetikus algoritmussal kezelik.

Végül nézzünk néhány olyan példát, ahol a feladat kifejezetten a közgazdasági előrejelzés! Ennek egyik nagy területe a statisztika. Rendszeresen alkalmaznak genetikus optimalizálást, például nemlineáris regressziós modellek alkalmazása során, illetve a becsléselméletben bonyolult *loglikelyhood* függvények esetében. Ilyenkor ugyanis a célfüggvény nagyon bonyolult, és sok lokális optimummal rendelkezik, így a hagyományos optimalizáló algoritmusokat nem lehet használni.

Konkrét alkalmazásokat elsősorban szimulációs modellekben láthatunk (például világmodellben). Ilyen modellekben a szimulációt *feketedoboz*-függvénynek, a szimuláció egyes paramétereit külső – kívülről szabályozható – változónak tételezzük fel (például: nyersanyagok kitermelésének mennyisége), míg más változókat a szimuláció eredményváltozóinak tekintünk (például: népességnövekedés, a bioszféra állapota stb.). A szimuláció mindig egy adott paraméterbeállítás mellett ad választ (például: ha az őserdők fakitermelésének ütemét 20 százalékkal csökkentjük, akkor az átlagos GDP növekedés 0,5 százalékponttal csökken stb.). A szimuláció optimalizálása esetén mi definiálhatjuk a külső változóinkat, azok lehetséges értékét, és mi dönthetjük el, hogy mely eredményváltozók milyen értékeit tartjuk kívánatosnak. (Például: milyen legyen az egyes nyersanyagok kitermelésének a mértéke annak érdekében, hogy meghatározott idő alatt a GDP legalább 1 százalékponttal növekedjen és a környezet minősége ne romoljon.) Szimulációk opti-

malizálása tipikusan a genetikus algoritmus számára kitalált feladat. Ráadásul, a genetikus algoritmus képes több célfüggvény esetén az efficiens megoldások meghatározására is (*Benedek* [1999*b*]).

Végezetül nézzünk példát egy olyan előrejelzési modellre, amelyet rendszerint bankok alkalmaznak! A feladat az, hogy a bank által üzemeltetett ATM bankjegy-automatákban optimális szinten legyen az adott bankjegyek mennyisége. Az optimális itt azt jelenti, hogy ne legyen nagyon sok, mert ennek napi kamata költségként jelentkezik az adott banknál, de ne is legyen nagyon kevés, mert a kifogyó automata erősen rontja az adott bank imázsát. Továbbá, ha túl gyakran kell feltölteni az automatát, akkor a szállítási költség lesz nagyon magas. Pénzváltó-automaták esetében különösen fontos és nehezen megválaszolható kérdés az, hogy az egyes valutafajtákból mekkora összeg legyen raktározva. A feladatot neurális háló és genetikus algoritmus segítségével lehet kezelni. A neurális hálózat segítségével, a múlt tapasztalatai alapján meg tudjuk határozni az adott bankjegy-automata várható forgalmát (akár mint valószínűségi változót). Ezután a genetikus algoritmus segítségével meghatározzuk az optimális feltöltési stratégiát. Az optimalitás itt megint több szempontot érinthet, ugyanis nem biztos, hogy a legalacsonyabb üzemeltetést kell választani akkor, ha a véletlen hatások miatt ez nagyon kockázatos (gyakori kifogyás).

Az alkalmazások sorát sokáig folytathatnánk, célunk azonban most csupán annyi volt, hogy az Olvasó megismerkedjen egy-két konkrét példával. Így remélhetőleg a következő fejezetben az algoritmusok tanulmányozása is könnyebb lesz.

Szükséges feltételek az evolúciós elméletek alkalmazásához

Ebben a pontban kifejezetten az előrejelzési modellekben történő evolúciós alkalmazások szükséges feltételeit vizsgáljuk. Ezeket a feltételeket sokkal lazábban kell értenünk, mint más matematikai modellek szükséges feltételeit, mivel sokszor elképzelhető az evolúciós algoritmusok alkalmazása abban az esetben is, ha az alábbi feltételek nem mindegyike adott. Jelen pontban inkább azokat a jelenségeket soroljuk fel, amelyek esetén az evolúciós eljárások sikeresek lehetnek. Ezeket neveztük *feltételeknek*.

Analógia. Előnyös, ha az adott problémához, amelyet evolúciós eljárásokkal kívánunk megoldani, található valamilyen analóg evolúciós példa, illetve a szereplők cselekvése ezen elmélet segítségével magyarázható. Ilyen lehet a mintafelismerés, a fejlődés, a tanulás. A legtöbb közgazdasági előrejelzési alkalmazásban ez a feltétel adott.

Komplexitás. A megoldandó feladatnak elegendően bonyolultnak kell lenni ahhoz, hogy érdemes legyen evolúciós eljárásokat alkalmazni. Nagyon sok más módszer ismert, és mint megállapítottuk, az evolúciós algoritmusok nem vezetnek mindig a legjobb megoldáshoz, sokszor csak elég jóhoz. Egyszerűbb feladatok esetén érdemesebb matematikai és statisztikai módszereket vagy teljes leszámlálást alkalmazni.

Adat. Az evolúciós eljárások legsikeresebben akkor alkalmazhatók, ha már „elveszünk az adatok között”, azaz roppant nagy mennyiségű és nehezen átlátható adat birtokában kell valamilyen döntési modellt meghatároznunk. Természetesen az adathalmaznak relevánsnak, többé-kevésbé teljesnek és hibamentesnek kell lennie az adott feladatra nézve. Ha nincs összefüggés, azt az evolúciós eljárások általában szintén meg tudják állapítani, viszont ha hibás és/vagy hiányos adatbázissal dolgozunk, akkor az algoritmusok könnyen félrevezetődhetnek, és hibás következtetéseket állapíthatnak meg.

Külső információ. Az evolúciós eljárásokat nem szabad olyan esetekben alkalmazni,

amikor egyéb információk lehetővé teszik az egyszerűbb módszerek alkalmazását. [Például ha ismert – vagy feltételezzük, hogy ismert – a modellforma, és csak annak paramétereit akarjuk becsülni, vagy ha a célfüggvényről tudjuk, hogy egyetlen lokális (= globális) optimális pontja létezik.] Az evolúciós eljárásokkal akkor érdemes próbálkozni, ha nem tudunk, vagy nem akarunk hipotéziseket megfogalmazni az adott problémára vonatkozóan. Továbbá az evolúciós algoritmusok eredményeit mint automatikusan – a számítógép által – megfogalmazott hipotéziseket kell kezelni, s ezeket további statisztikai-matematikai módszerekkel érdemes ellenőrizni.

Az algoritmusok működése

A genetikus algoritmus

A genetikus algoritmust 1975-ben fedezte fel J. H. Holland (*Holland* [1975]). Azóta az algoritmus rengeteg továbbfejlesztést, módosítást megélt, és különböző változatait alkalmazzák. Mi az alapalgoritmust *Pham–Karaboga* [1998] kötet alapján ismertetjük.

A genetikus algoritmus hátterében az úgynevezett *sémaelmélet* húzódik. A *séma* az egyes egyedek valamilyen halmazát reprezentálja, azaz a populáció valamely részhalmazát jelenti. Séma lehet például a négyjegyű számok halmazán a 4*2* reprezentáció, ahol a * helyére tetszőleges értékek kerülhetnek. Egy sémát két paraméter határoz meg: a hossza és a rendje. A hosszúság az első és utolsó fix számjegy között található számjegyek számát, a rend pedig a meghatározott értékű számjegyek sorszámát jelenti. Az elmélet alapján tehát az egyes sémák eloszlása egyik generációról a másikra mindig a séma rendjétől, hosszától és túlélési értékétől függ. A genetikus algoritmus célja, hogy meghatározza azt a sémát (génkombinációt), amely az adott probléma szempontjából optimális.

A genetikus algoritmusok gyakorlatilag semmit sem tudnak (semmit sem használnak ki) az optimumproblémával kapcsolatosan, és nem foglalkoznak közvetlenül a paraméterekkel. Ehelyett kódokkal dolgoznak, amelyek az egyes paramétereket reprezentálják. Ezért az *első kérdés* az, hogyan kódoljuk a problémát, hogyan ábrázoljuk a paramétereket. Mivel a genetikus algoritmus a lehetséges megoldások populációjával foglalkozik, és nem egy darab lehetséges megoldással, ezért *a második* kérdés az, hogyan határozzuk meg a kezdeti populáció egyedeit. A *harmadik* kérdés az, hogy a további generációk meghatározásához (fejlődés) melyek a megfelelő genetikus műveletek (öröklődés). A *negyedik* kérdés a visszacsatolás kérdése, azaz a célfüggvény alapján meg kell határozni az egyedek túlélési valószínűségét. *Végül* az algoritmusnak valamilyen kritériumot kell szabni a keresés befejezésére.

Reprezentáció. Tegyük fel, hogy az $f(\mathbf{x})$ függvény maximumát szeretnénk meghatározni. Ebben az esetben a populáció egyedeit $\mathbf{x}$ különböző értékei képviselik. Kérdés, hogy a számítógép számára milyen formában adjuk meg ezt az értéket. A reprezentáció formájától függően ugyanis igen különböző lehet a genetikus algoritmus eredménye, azaz ugyanannak a problémának más-más reprezentációja különböző pontosságot és futási időt adhat. A numerikus optimalizálás esetében általában két reprezentációs módszert szoktak alkalmazni (*Michalewicz* [1992], *Davis* [1991]). A gyakrabban alkalmazott módszer a *bináris kódolás*, azaz az $\mathbf{x}$ változót kettes számrendszerben ábrázoljuk. Ennek az előnye, hogy ilyenkor a legnagyobb a lehetséges sémáknak a száma. A szakirodalomban nagyszámú különböző bináris kódolási módszer lelhető fel (például: *Uniform coding; Gray scale coding*). A másik lehetséges módszer egyszerűen egész vagy valós vektorként ke-

zeli **x**-et.[4] Bináris kódolás esetén fontos feladat annak meghatározása, hogy hány bitet használjunk az optimalizálandó változó(k) ábrázolására.

Kezdeti populáció. Az optimalizálás megkezdése előtt a genetikus algoritmusnak szüksége van néhány kezdeti megoldásra. Az egyik lehetséges módszer, hogy a számítógép véletlenszám-generátora segítségével előállítjuk **x** néhány értékét, s ezek képviselik az első generáció egyedeit. A másik lehetséges módszer, hogy külső információt (is) felhasználva, az optimális megoldás környékéről indítjuk az algoritmust. Ebben az esetben nyilvánvalóan valamilyen előzetes tudásunk van a problémáról, s ezt kihasználva várhatóan az első esetnél rövidebb idő alatt jutunk optimális megoldáshoz.[5]

Genetikus műveletek. Három hagyományos művelet használatos: *szelekció*, *keresztezés* és *mutáció*, valamint egy további – ritkábban alkalmazott – az *invertálás*. A genetikus optimalizálás során e műveleteknek számtalan fajtáját és paraméterezését alkalmazhatjuk, továbbá nem is szükséges e műveletek mindegyikét használnunk, mivel minden egyes művelet egymástól függetlenül működik. Az alkalmazott genetikus műveletek mennyisége és minősége az adott probléma, illetve a reprezentáció függvénye. A műveleteket bináris kódolást feltételezve mutatjuk be (részletesen lásd *Whitely–Hanson* [1989], *Baker* [1985]).

Szelekció. A szelekciónak az a célja, hogy több olyan egyedet gyártson, amelynek a túlélési esélye magasabb, és kevesebb olyat, amelynek a túlélési esélye alacsonyabb. A szelekció lényegesen befolyásolja, hogy milyen rövid idő alatt kezd az algoritmus az optimális megoldáshoz konvergálni. Általában két módszert alkalmaznak: az *arányos (proportional)* – más néven „*rulettkerék*" *(roulette wheel)* – és a *rangsorolt (ranked)* szelekciót. Az előbbi módszert úgy képzeljük el, hogy adott egy óriási rulettkerék, ahol minden egyed arányosan annyiszor szerepel, amekkora a túlélési esélye. Annyiszor pörgetjük, ahány egyedet a következő generációba szánunk, így a nagyobb túlélési esélyű egyedek nagyobb valószínűséggel – többször – kerülnek a következő generációba. Rangsorolt szelekció esetén egy korábbi generációból egy egyednek legfeljebb egy replikációja kerülhet a következő generációba. Ekkor minden egyedhez egy véletlen számot generálunk úgy, hogy annak várható értéke annál magasabb legyen, minél magasabb az egyed túlélési esélye. Végül e szám alapján rangsoroljuk az egyedeket, és kiválasztjuk az első *n* darabot a következő generáció számára.

Keresztezés. Ez az a művelet, amely alapjában megkülönbözteti a genetikus algoritmust más optimalizáló módszerektől. A keresztezés során két létező egyed *(szülők)* két új egyedet *(gyerekek)* hoz létre. A szülőket szelekcióval határozzuk meg. A keresztezésnek jó néhány különböző fajtáját alkalmazzák [például: egy ponton történő *(one-point),* két

[4] Egy rövid példával szemléltetjük a két reprezentációs módszert. Tegyük fel, hogy a lehetséges megoldások egész számok, és az optimális megoldás valahol $x = 16$ körül található. Binárisan kódolva 16 = [0 1 0 0 0 0], viszont 15 = [0 0 1 1 1 1]. Ha az utóbbi irányból „közelít" az algoritmus, akkor esetleg olyan eredményt ad, hogy a [0 0 1 1 * *] séma helyes. Ezzel szemben ha egyszerű egész számként kezeljük, akkor az 1* (azaz tizenvalamennyi) séma a helyes irány.

[5] Általában az első módszert alkalmazzák vagy a két módszert kombinálják. Tegyük fel például, hogy az $f(x, y)$ függvényt szeretnénk maximalizálni, azzal a feltétellel, hogy $x = y$. A genetikus algoritmust, ha mint feketedoboz-szimulációt alkalmazzuk, akkor a feltételekkel nem tud külön mit kezdeni, és ezért azt az optimalizálandó függvénybe kell beépíteni, például: $g(x, y) = f(x, y) - a(x - y)^2$, ahol a egy megfelelően nagy, pozitív szám. Véletlen indítás esetén igen kicsi a valószínűsége annak, hogy egymás után két azonos számot generálunk, így nagyon sokáig eltarthat, amíg az algoritmus egy egyáltalán lehetséges megoldásra rátalál, majd ezek után vagy azonnal leáll, vagy újfent sokáig tart, amíg arra a sémára „rájön", ami $x = y$-t jelenti. Ha eleve olyan – akár véletlenül meghatározott – pontokból indítjuk az algoritmust, ahol $x = y$, akkor hamarabb jutunk megoldáshoz. E példát csupán szemléltetésül említjük, nyilvánvaló, hogy sokkal egyszerűbb megoldás az, ha a korlátozott probléma helyett a számítógép a nem korlátozott $f(x, x)$ problémát oldja meg.

ponton történő *(two-point),* ciklikus *(cycle)* és egyenletes *(uniform)* keresztezés]. A keresztezés töréspontjait külön heurisztika szabályozza, így a keresztezés során egyre közelebb kerülünk az ideális sémához. A 2. *a)–d)* ábrán az említett négy keresztezésre mutatunk példát.

2. a) ábra
Egy ponton keresztezés

1. szülő:	[**1 0 0 0 1 0 0 1 1 1 1**]
2. szülő:	[0 1 1 0 1 1 0 0 0 1 1]
1. gyerek:	[**1 0 0 0 1** 1 0 0 0 1 1]
2. gyerek:	[0 1 1 0 1 **0 0 1 1 1 1**]

2. b) ábra
Két ponton keresztezés

1. szülő:	[**1 0 0 0 1 0 0 1 1 1 1**]
2. szülő:	[0 1 1 0 1 1 0 0 0 1 1]
1. gyerek:	[**1 0 0** 0 1 1 0 **1 1 1 1**]
2. gyerek:	[0 1 1 **0 1 0 0** 0 0 1 1]

2. c) ábra
Ciklikus keresztezés

1. szülő:	[**1 0 0 0 1 0 0 1 1 1 1**]
2. szülő:	[0 1 1 0 1 1 0 0 0 1 1]
1. gyerek:	[**1** 1 **0** 0 **1** 1 **0** 0 **1** 1 **1**]
2. gyerek:	[0 **0** 1 **0** 1 **0** 0 **1** 0 **1** 1]

2. d) ábra
Egyenletes keresztezés

1. szülő:	[**1 0 0 0 1 0 0 1 1 1 1**]
2. szülő:	[0 1 1 0 1 1 0 0 0 1 1]
Minta:	[1 1 0 0 1 1 1 1 0 0 0]
1. gyerek:	[0 1 **0 0** 1 1 0 0 **1 1 1**]
2. gyerek:	[**1 0** 1 0 **1 0 0 1** 0 1 1]

Mutáció. A mutáció során az egyedeket bitről bitre megvizsgálja az algoritmus, és véletlenszerűen megváltoztathatja. A mutáció heurisztikája azt jelenti, hogy az egyes bitek milyen eséllyel változzanak meg *(mutációs ráta),* és csakúgy, mint a keresztezésnél, ez az érték dinamikusan változhat annak érdekében, hogy az ideális séma kialakuljon. Szemben a keresztezéssel, a mutációhoz elegendő egy szülő, és az eredmény egy gyerek, ahol a szülő kiválasztása ismét a szelekcióra van bízva. A mutáció segítségével az algoritmus új területeket fedez fel, ami azért hasznos, mert elvezeti az algoritmust a lokális optimális pontokból a globális optimum felé *(3. ábra).*

3. ábra
Mutáció

Szülő:	[0 1 0 0 1 **1** 0 0 1 1 1]
Gyerek:	[0 1 0 0 1 **0** 0 0 1 1 1]

Invertálás: A művelet során a szülőegyed megfelelő heurisztikával kiválasztott bitjei felcserélődnek, azaz 0-ról 1-re, 1-ről 0-ra változnak. Gyakorlatilag a mutáció is felfogható egy egybites invertálás műveletnek *(4. ábra)*.

4. ábra
Invertálás

Szülő:	[0 1 0 0 **1 1 0 0** 1 1 1]
Gyerek:	[0 1 0 0 **0 0 1 1** 1 1 1]

Többször is hangsúlyoztuk, hogy az egyes műveleteknek különböző paraméter-beállításai léteznek. Ilyen paraméterek: a *populáció mérete*, a *keresztezési ráta*, illetve a *mutációs ráta*. Számtalan publikáció foglalkozik e paramétereknek a genetikus algoritmus sebességére és pontosságára való hatásának vizsgálatával (*Schaffer és szerzőtársai* [1989], *Grefenstette* [1986], *Fogarty* [1989]). Természetesen maga az optimumprobléma képviseli a legjelentősebb hatást a genetikus algoritmus számára, és két különböző probléma esetén ugyanaz a beállítás egészen különböző eredményeket produkálhat. Ennek ellenére megfogalmazható néhány általános megállapítás. Nagy populáció (mintaméret) esetén a számítás lassabb lesz, azonban mivel az egyedek jobban lefedik a lehetséges megoldások halmazát, ezért nagyobb a valószínűsége annak, hogy globális optimumot kapjunk. A keresztezési ráta *(crossover rate)* meghatározza, hogy milyen gyakran történjenek a keresztezések. Túl alacsony ráta esetén nagyon lassúvá válhat a konvergencia, míg túl magas ráta esetén elképzelhető, hogy egyetlen megoldás körül ugrálunk. Végül a túlságosan magas mutációs ráta erőteljesen különbözővé teszi a populációt minden egyes lépésben, ezáltal instabilitást okozhat. Másik oldalról a globális optimum megtalálása nagyon nehéz olyan esetben, ha a mutációs ráta túl alacsony.

A genetikus algoritmusokat az teszi nehézzé, hogy ezen paraméterek megadására nincs „helyes" eljárás. Néhány problémafajtáról be lehet bizonyítani, hogy adott paraméter-beállítás esetén milyen gyorsan konvergál, illetve mekkora valószínűséggel találja meg az optimális megoldást, sok esetben azonban ismeretlen (fekete doboz) a célfüggvény. Ebben az esetben gépünk kapacitásától függően különböző paraméter-beállításokkal kell próbálkoznunk. Mikor végül elfogadjuk a genetikus algoritmus által optimálisnak vélt megoldást, tisztában kell lennünk azzal, hogy nem 100 százalékig biztos, hogy a valódi optimumban vagyunk. Azt azonban állíthatjuk, hogy az adott körülményekhez képest (vagyis, hogy semmit sem tudunk a problémáról), adott idő alatt (számításidő-kapacitás) a lehető legjobb eredményt kaptuk. Abban az esetben, ha mindenképpen meg akarunk győződni az eredmény helyességéről, vagy előzetes információnk van a célfüggvény tulajdonságairól, más módszerekkel érdemes kombinálni a genetikus algoritmust (például: *simulated annealing*).

Túlélési valószínűségek. A túlélési valószínűségek meghatározására is igen sokféle módszert lehet alkalmazni. Abban az esetben, ha a feladatban a célfüggvény mellett más feltételek is szerepelnek, akkor a feltételeket a célfüggvénybe kell integrálni (lásd az 5. lábjegyzetet). Nyilvánvaló, hogy maximumfeladat esetén, minél magasabb az adott egyed-

nek megfelelő pontban a függvény értéke, annál magasabb az adott egyed túlélési esélye; minimumfeladat esetén pedig fordítva. A kérdés az, hogy mennyivel. Tegyük fel, hogy 4 egyed túlélési valószínűségét szeretnénk megadni. Legyen a függvényértékük rendre 1, 2, 3 és 4, továbbá legyen a feladat maximum probléma. Az egyik legegyszerűbb számítási lehetőség a *diszkrét korlát* állítása. Legyen ennek a diszkrét korlátnak az értéke mondjuk 2,5, és a korlát alatti egyedek túlélési valószínűsége 0, a korlát felettieké 1. Így az 1. és a 2. egyed túlélési valószínűsége 0, a 3. és a 4. egyedé pedig 1. Folytassuk e példát a már ismertetett szelekcióval! Tegyük fel, hogy három egyedet kell kiválasztani. Rulettmódszer esetén a rulettkeréken 1 darab 3-as és 1 darab 4-es szerepel. Háromszor forgatunk, így átlagosan a következő mintába 1,5 darab 3-as és 1,5 darab 4-es kerül. Elképzelhető az is, hogy adott esetben 3 darab 3-as kerül be, viszont 1-es és 2-es soha. Rangsorolás esetén minden egyedhez rendelnünk kell egy véletlen számot. Legyen ez a véletlen szám a [0; *túlélési valószínűség*] intervallumon egyenletes eloszlású. Adott esetben például rendre a következő értékeket kaphatjuk: 0, 0, 0,89, 0,41. A rangsor pedig emiatt a következő: 3, 4, 1, 2. Három egyedet kiválasztva megállapíthatjuk, hogy jelen esetben a rangsoroló módszer bármilyen véletlen számokat is generálunk, mindig a 3, 4 és 1 egyedeket választja ki.

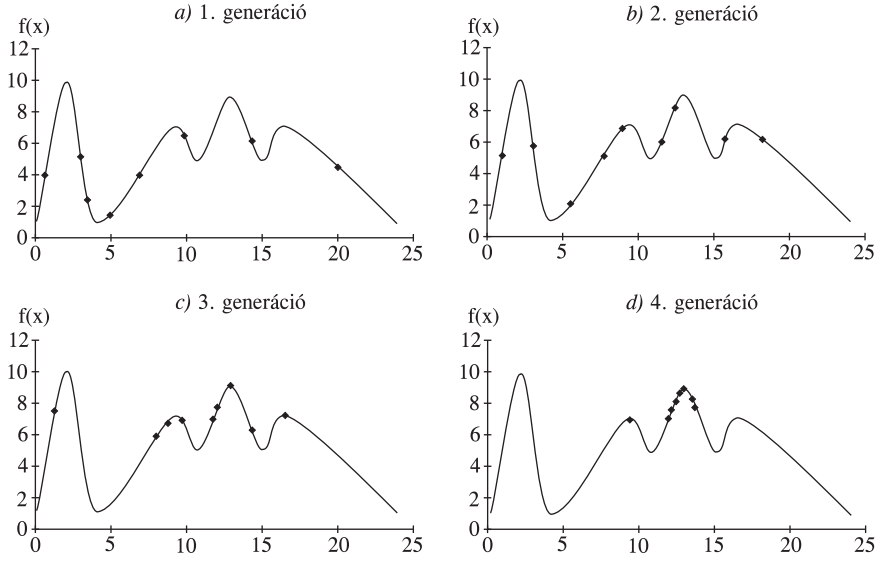
Nézzünk még egy módszert a túlélési valószínűség meghatározására, az *egyenletes eloszlást!* Ebben az esetben a függvényértékkel arányos túlélési valószínűségeket generálunk az egyedek számára, azaz a túlélési valószínűségek rendre a következők: 0,1, 0,2, 0,3, 0,4. Bár különböző alkalmazásokban más-más eljárásokat alkalmazhatnak, felhasználhatják korábbi generációk össztúlélési valószínűségeit, más eloszlásokból vehetnek véletlen számokat stb., a leggyakrabban alkalmazott módszer mégis az említett két eljárás.

Leállási kritérium. Az algoritmusnak elvileg akkor kellene megállni, amikor elérte a globális optimumot, gyakorlatilag azonban más helyen is megállhat. Megállhat a globális optimum környékén, megállhat lokális optimumban, annak a környékén vagy akár szuboptimális pontban. A számítógép csak „sejteni" tudja, hogy hol tart éppen, azt vizsgálva, hogy az újonnan generált egyedek milyen közel vannak az előzőkhöz, és milyen nagy az eltérés a függvényértékekben. Ezt nevezzük *konvergenciakritériumnak*. Amennyiben az algoritmust ez szabályozza, nagyon valószínű, hogy legalább lokális optimumot kapunk, továbbá minél kevésbé érzékeny a konvergenciakritérium, annál valószínűbb, hogy a globálisan optimális pont(ok)hoz közel leszünk. Ez azonban megnöveli a számítási időt. Annak érdekében, hogy ne lokális, hanem globális optimumban álljunk meg, egyrészt elegendően nagy számú populációval kell dolgoznunk, másrészt nem szabad sürgetnünk a konvergenciát. A szakirodalom *korai konvergenciának*, illetve *genetikus elsodródásnak* nevezi azt a jelenséget, amikor a globális megoldás helyett az algoritmus lokális optimumban áll le. Más leállási kritériumok is elképzelhetők és használatosak; például meghatározott *futási idő* vagy meghatározott *ciklusidő* (meghatározott generációszám). E két utóbbi esetben elvileg még a lokális optimum sem biztosított, illetve elképzelhető, hogy a túl hosszúra választott futási idő miatt az algoritmus már nem mozdul a kialakult stabil állapotból egy idő után. Legtöbbször a konvergenciakritériumot kombinálják ez utóbbi két kritériummal.

Az *5. a)–d) ábrán* vonallal a teljes leszámlálás eredményét, ponttal a genetikus algoritmus néhány generációjának egyes egyedeit jelöltük. Jól látható, hogy az algoritmus fokozatosan eltolódik a lokális optimum felé, és nem a globális optimumban áll le.

E tanulmány – a Közgazdasági Szemle 2001. januári számában megjelenő – második részének Függelékében egy rövid feladatot mutatunk be annak illusztrálására, amikor a konvergencia sebessége és minősége kiszámítható.

5. ábra
Genetikus elsodródás



A neurális háló[6]

Neurális hálóval több különböző feladatot is meg lehet oldani. Alkalmazható összefüggések feltárására, optimalizálásra, szegmentálásra és egyéb feladatokra. Ebben a pontban – *Fu* [1994] alapján – az első alkalmazásnak megfelelő modellt részletesen bemutatjuk, további két lehetőséget pedig röviden a következő pontban ismertetünk.

A neurális háló két objektumból áll: *csomópontokból (neuronok)* és az őket összekötő *élekből (szinapszis)*. A bejövő információ (inger, impulzus) egy-egy neuronba érkezik. Amennyiben az adott inger egy *küszöbértéket* meghalad, a neuron továbbküld egy jelet abba a neuronba, amellyel összeköttetésben van. Egy ilyen következő neuronba több neuron is küldhet impulzust. Ha ezek összessége meghaladja e neuron ingerküszöbét, akkor ez a neuron is továbbküld egy impulzust a következő neuronba stb. Végül az impulzus eljut néhány végső neuronba, s attól függően, hogy melyikbe jutott el, úgy kötünk valamilyen válaszreakciót az adott inputhoz. Tanulás során az intenzíven használt szinapszisok megerősödnek, a keveset használtak pedig megszűnnek. Az emberi agy tanulásának ezt a mechanizmusát utánozza le a számítógépes modell. Egyes elemeket azonban meg kellett változtatni, ugyanis még a leggyorsabb számítógépek sem képesek néhány száz neuronnál nagyobb hálózatot modellezni – szemben az aggyal, amelyben milliárdnyi neuron működik. Az első változtatás, hogy az egyes neuronok több más neuronnak is adhatnak impulzust, akár visszafelé is. A második változtatás, hogy minden szinapszisnak meghatározott az impulzusátadási hatásfoka, azaz meghatározott erővel *(súlyozással)* továbbítja az információt. A harmadik változtatás az, hogy az ingereket fogadó neuronokat különböző mértékű (intenzitású) ingerek érhetik, továbbá az ingerek továbbítására nem egy egyszerű küszöbérték-meghaladási kritérium létezik, hanem bonyolult függvények segítenek meghatározni a kimenő impulzus mértékét. Kivétel általában a beérkező ingereket fogadó neuron, ahonnan is az inger „egy az egyben” továbbí-

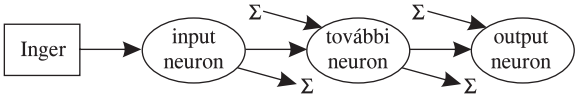
[6] E pont bővebb kifejtését lásd *Kocsis–Szabó* [2000] – a szerző által írt – Függelékében.

tódik. Végezetül, a tanulás mechanizmusát matematikai-statisztikai módszerek szabályozzák, amelyek közül a legismertebb és leginkább használatos módszer a *backpropagation* eljárás.

A *6. ábra* egy egyszerű inger beérkezését és haladását mutatja, ahol az egyes Σ jelek további neuronokat jelentenek:

6. ábra

A beérkező inger haladása



Tegyük fel, hogy nincsenek további (Σ) neuronok, és az első és második neuront összekötő szinapszis ereje *(súlya)* 1, a második és harmadik neuront összekötőé pedig 2. Tegyük fel azt is, hogy egy inger akkor továbbítódik az egyes neuronokból, ha az intenzitása meghaladja a 2-t. Ha az inputneuront érő inger intenzitása például 1 (és nem vonatkozik rá az ingerküszöb), akkor a második neuronba érkező jel: $1 \times 1 = 1$. Mivel azonban ez az érték nem éri el az ingerküszöböt, ezért az innen továbbhaladó jel értéke 0 lesz. Az outputneuronba a $2 \times 0 = 0$ érték kerül. Ha azonban az első inger intenzitása 3, akkor a második neuronba $1 \times 3 = 3$ érték kerül. Ez meghaladja az ingerküszöböt, ezért továbbítódik, s végül az utolsó neuronba a $2 \times 3 = 6$ érték jut el. A leggyakrabban használt ingerküszöbfüggvény a szigmoidfüggvény, de egyéb formákat is alkalmaznak *(7. a)–f) ábra)*.

Az ábrákon jól látható, hogy az egyes neuronok ingertovábbításának intenzitása 0 és 1 között van [kivéve az *f*) ábrát]. A legtöbb neurális hálózati alkalmazásban ezt használják, sőt az input- és az outputváltozókat is legtöbbször a [0; 1] intervallumba transzformálják. Ugyanakkor a neuronokba érkező összes intenzitás $[-\infty; +\infty]$ között értelmezett, ugyanis egyrészt több neuronból is érkezhet impulzus egy neuronba, továbbá a súlyok tetszőlegesen megváltoztathatják az egyes neuronokból távozó impulzus szintjét. Egy neuronba tehát ha n darab neuronból p_i súlyokon keresztül x_i értékek érkeznek (ahol $i = 1, 2, \ldots, n$), akkor a kimeneti érték, szigmoidfüggvény alkalmazása esetén:

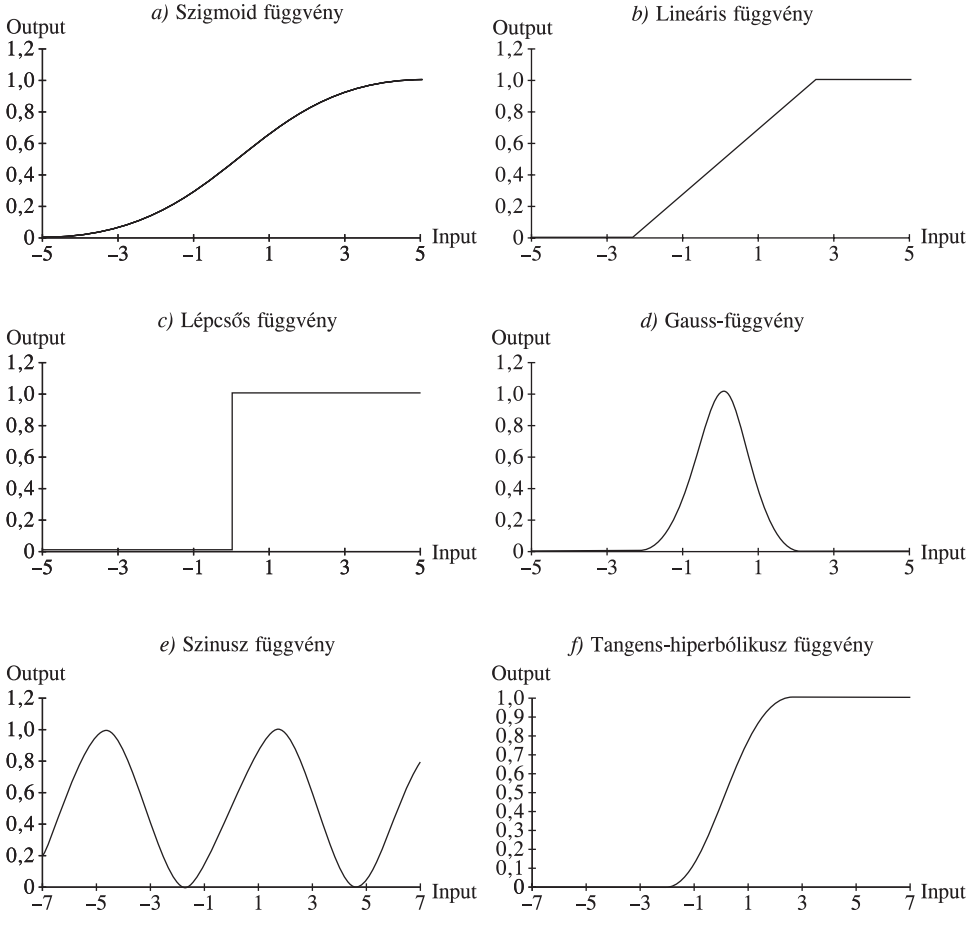
$$\left[1 + \exp\left(-\sum_{i=1}^{n} p_i x_i\right)\right]^{-1}.$$

Meg kell jegyeznünk, hogy a súlyozott inputértékeket nem feltétlenül összeadják (azaz $I = \Sigma p_i x_i$). Elképzelhetők más összegző függvények is, például szorzat ($I = \Pi p_i x_i$), kumulatív összegzés ($I_{új} = I_{régi} + \Sigma p_i x_i$), minimum-, maximum- stb. függvények.

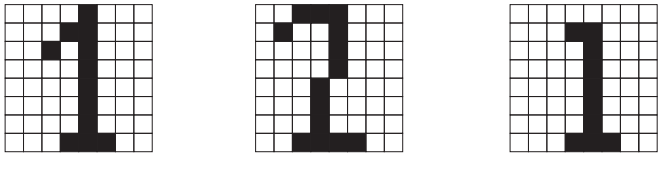
A hálózat felépítése. A neurális hálózat működését egy egyszerűsített karakterfelismerő modell segítségével mutatjuk be. Tegyük fel, hogy van egy 8×8-as négyzetünk, amelybe különböző számokat rajzolunk, 0-tól 9-ig. A 64 kis négyzet közül fekete lesz az általunk érintett, és fehér marad az általunk nem érintett négyzet. Miután kézzel rajzolunk, ezért egy-egy szám többször lerajzolva kicsit különböző alakokat ölthet, a számítógéptől viszont nem várjuk, hogy az összes lehetséges alakzathoz rendeljen valamilyen számot.

A feladat az, hogy néhány mintát megmutatunk a számítógépnek, miután ő megtanulja felismerni az egyes számjegyeket, hogy később olyan mintákat is felismerjen, melyeket korábban nem mutattunk meg neki. Ezek után bárki is rajzol újabb számjegyeket, a neurális hálózat megmondja mit rajzolt. A hálózatnak természetesen csak annyi információt adunk át, hogy egy-egy kis négyzet egy adott szám berajzolása után fekete lett, vagy fehér maradt. Tegyük fel, hogy fekete esetben az adott négyzetnek 1 értéket, fehér esetben 0 értéket adunk. Ezután azt szeretnénk, hogy a hálózat megmondja, hogy a berajzolt számjegy 0, 1, … vagy 9-es számjegy. Most, az emberi agyban lévő neuronhálózathoz hasonlóan felrajzoljuk a „saját” hálózatunkat.

7. ábra
Ingerküszöbfüggvények

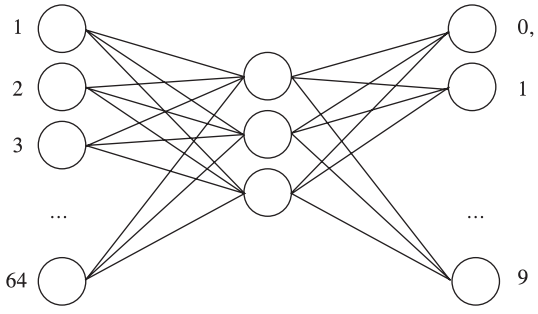


8. ábra
Az 1-es néhány elképzelhető rajza



A *9. ábrán* jól láthatóan három oszlopot különítettünk el. Minden oszlopban neuronok találhatók, de az egyes neuronoknak más-más a szerepe. Emiatt a szerepkülönbség miatt szokás őket elkülöníteni, külön oszlopba (vagy sorba) foglalni, és valamilyen gyűjtő megnevezéssel ellátni. Ez az elnevezés a *réteg*, azaz a réteg nem más, mint néhány

9. ábra
A neurális háló felépítése



neuronból álló halmaz. Három réteget szokás megkülönböztetni: *a bemeneti réteget, a köztes réteget* és *a kimeneti réteget*. A bemeneti réteg neuronjaiba érkezik az információ, a köztes réteg neuronjai dolgozzák fel azt, és a kimeneti réteg neuronjai adják meg a lehetséges válaszokat.

Az első oszlop a bemeneti réteg *(input layer)*. Az, hogy hány neuron kerül a bemeneti rétegbe, az mindig a magyarázó (bemeneti) változók számától és minőségétől függ.[7] Jelen esetben 64 darab olyan változónk van, melynek az értéke 0 vagy 1 lehet, attól függően, hogy az adott kis négyzet fehér vagy fekete. A középső oszlop a köztes vagy rejtett réteg *(hidden layer)*.[8] Ilyen oszlopból több is lehet, ez növeli a hálózat bonyolultságát, és ez által a tanulási kapacitást is. A középső rétegben található neuronok számának meghatározására több módszer is van. Legegyszerűbb az az eset, amikor a magyarázó és magyarázott változók számának és minőségének függvényében határozzuk meg az itt szereplő neuronok számát. Elképzelhető azonban az is, hogy a hálózat a tanulási folyamat során „jön rá arra", hogy az általa szerepeltetett neuronok száma még nem elegendő, azaz a tanuló adatbázis komplexitása is befolyásolhatja a köztes réteg(ek) nagyságát. Az utolsó oszlop a kimeneti réteg *(output layer)*.[9] Száma a magyarázott (kimeneti) változók (több is lehet!) számától és minőségétől függ. Jelen esetben 10 különböző kimeneti neuronunk van.

A hálózat működése. Tegyük fel, hogy a hálózat már mindent megtanult. Ebben az esetben a súlyok valamilyen optimális értékre vannak beállítva, és a továbbiak során ezek

[7] A bemeneti réteg felépítése mindig nagyon egyszerű. Legalább annyi neuron kell bele, ahány különböző magyarázóváltozónk van. Ha a hőmérsékletet akarjuk inputként használni, akkor egy neuron elég. A neuronba mindig az az érték kerül, ahány fok az adott megfigyelés esetében volt. Ha a változónk az, hogy MELEG volt-e vagy HIDEG, akkor is elég egy neuron: 0 lesz benne, ha MELEG volt, 1, ha HIDEG. Ha az egyik változó több lehetséges diszkrét érteket vehet fel: például ESŐ/HÓ/JÉG/SZÁRAZ, akkor itt már 4 neuront kell használni. Ha pedig a változóban egész értékek lehetnek, például hány száraz nap volt: 0, 1, …, 365, akkor lehet választani, hogy egy darab neuron reprezentálja a változót, és a bele kerülő értek az adott szám legyen, vagy 365 neuron. Látható, hogy nemcsak attól lesz bonyolult és nagyméretű az inputréteg, hogy sok változónk van, hanem attól is, hogy sok a lehetséges értékük.

[8] A köztes rétegbe kerülő neuronok száma nem egyértelmű. Nyilván ha kevés van, akkor egyszerű és gyors a tanulás, de nem biztos, hogy rá tud jönni minden összefüggésre. Ha viszont sok a köztes rétegben levő neuronok száma, akkor a tanulás lassú lehet, és elképzelhető a túltanulás veszélye is. Köztes rétegből több is lehet. Ilyenkor ugyanúgy kapcsolódik az előtte lévő és mögötte lévő neuronokkal, mint a *9. ábrán* látható egyetlen köztes réteg: Például: INPUT–KÖZTES1–KÖZTES2–KÖZTES3–OUTPUT.

[9] Az outputréteg neuronjainak száma megint egyszerűen meghatározható, és a tanulómintától függ, ugyanúgy, mint az inputrétegnél. Ha például az a válasz, hogy FEL KELL ÖLTÖZNI RENDESEN/NEM KELL NAGYON MELEGEN, akkor elég egy neuron (0, 1).

az értékek nem változnak. Mi történik ekkor, ha „mutatunk” a hálózatnak egy új rajzot? Először a bemeneti neuronokat feltöltjük a beírt számnak megfelelően nullákkal és egyesekkel. Ezután az összes bemenő neuronból a megfelelő súlyozással átkerülnek az értékek a köztes neuronokba. Majd innen újabb súlyozás után a kimeneti neuronokba. Ezek után csak meg kell vizsgálni, melyik kimeneti neuron érte el a legmagasabb értéket, és annak a neuronnak megfelelő számot rajzolták be előzőleg – legalábbis a hálózat szerint.

A tanulás. A legérdekesebb kérdés természetesen az a folyamat, amikor a hálózat súlyozása kialakul. Ez a folyamat a tanulás. A tanuláshoz szükséges egy tanuló-adatbázis, amelynek segítségével maga a tanulás megvalósul. Jelen példánkhoz egy ilyen tanuló-adatbázis lehet a következő: a megadott 8×8-as kockába valaki kézzel berajzol 20 darab 0-t, 20 darab 1-est stb. Minden rajzolás után megmondja a számítógépnek, hogy mit rajzolt. A számítógép elraktározza ezt a tanuló-adatbázist, amelynek formája valami hasonló lesz:

1. négyzet:	0 (üres)
2. négyzet:	0
3. négyzet:	1 (teli)
...	
63. négyzet:	1
64. négyzet:	0
A berajzolt szám:	5

Ez egy darab tanulási minta, így a mi példánkban 200 ilyen minta található. Tegyük fel, hogy egyesével mutatjuk meg a mintákat a neurális hálózatnak. Kezdeti állapotban a súlyszámok véletlenszerűen vannak megadva. Ha most olvasnánk be az adatokat, akkor a hálózat véletlenszerűen adná 0–9-ig a válaszokat. Tanuljuk meg most az első mintát! Ehhez először feltöltjük az inputneuronokat, azaz a mintának megfelelően a 64 neuron 0-kat és 1-eseket kap. Az értékek a kezdeti súlyozással eljutnak a köztes, majd végül az outputneuronokba, és valamilyen választ fognak adni, például a számítógép azt mondja, hogy a kettes a képen látható szám. A tanuló-adatbázisból azonban tudjuk, hogy a berajzolt szám valójában hányas volt, azaz összehasonlítjuk az eredményt a helyes outputértékkel. Tegyük fel, hogy különbözik, mondjuk a rajz valójában a nullát ábrázolta. Ekkor meg kell változtatni a súlyozást, méghozzá úgy, hogy a neurális háló is a nulla válaszreakciót adja ugyanarra a bemeneti képre. Ezután folytatódik a tanulás, veszi a következő mintát. Tegyük fel, hogy erre kijön, hogy nulla, pedig a beírt szám az egyes volt. Ekkor újból változtatni kell a súlyokon, de most már óvatosan, hiszen nem szabad, hogy az előzőleg megtanult nullát elveszítsük. A súlyszámok ilyen módon történő optimális beállítását bonyolult deriválási képletek határozzák meg. Ugyanaz a minta többször is lefuthat a tanulás során annak érdekében, hogy a hálózat által végül elfogadott súlyszámok stabilak legyenek.

A túltanulás veszélye. A túltanulás az a jelenség, amikor a tanulási folyamat során nem az általános problémát tanulja meg a hálózat, hanem a megadott adatbázis sajátosságait. Ennek kiküszöbölésére ketté kell választani az eredetileg meglévő adatbázist. Ez egyik része a tanuló-adatbázis, míg a másik része a tesztadatbázis. A tanuló-adatbázison végezzük el a tanítást, amelynek befejeztével olyan eredményhez jutunk például, hogy a megadott 100 esetből 95-öt eltalál a neurális modell. Ezután nem tanítjuk tovább a hálózatot, hanem megnézzük, mit mond az eddig ismeretlen tesztadatbázisról. Ha az ottani 100 esetből szintén csak néhányat ront el, akkor megelégedhetünk, hiszen az általános összefüggésekre sikerült a hálózatnak „rájönnie”. Ha azonban nagyon nagy a hibázási arány, akkor a hálózat túltanulta magát. Éppen ezért látjuk, hogy nagyon kell vigyázni az adat-

bázisok szétválasztásánál. Nyilvánvaló, hogy abban az esetben, ha a tanuló adatbázisban nem szerepel (vagy túlságosan alulreprezentált) a 3-as, akkor a tesztadatbázisban ezt a számot gyakran rosszul fogja a hálózat megtippelni.

Megjegyezzük, hogy a tanulás során lehetőség van arra, hogy a tanuló-adatbázist is kettévágjuk. Az algoritmus ekkor az adatok egy részén csak tanul, másik részén csak tesztel, és magát a tanulást csak akkor állítja le, ha a tanuló és tesztelő fázis közel azonos eredményt ad. Ebben az esetben, ha helyesen választottuk ketté a saját adatbázisunkat, csökkenthetjük a túltanulás veszélyét.

Egyéb algoritmusok

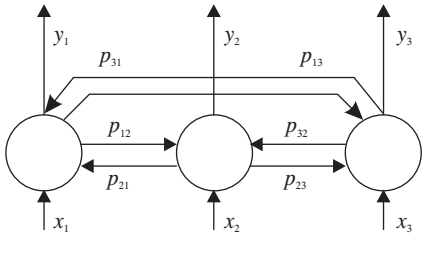
Rengeteg további algoritmus épül az előzőekben bemutatott két evolúciós módszerre. Ezek az algoritmusok állandóan változnak, újak alakulnak ki, illetve kapcsolódnak össze. Ebben a pontban néhány olyan további algoritmust mutatunk be röviden, amelyek az előrejelző modellezés szempontjából fontosak lehetnek.

Simulated annealing. Az eljárás a mechanikából származtatható (*Kirkpatrick és szerzőtársai* [1983]). A módszer ötletét az az eljárás adta, amikor a fémet erőteljes felmelegítés után fokozatosan hűtik le. Az anyagban található atomok a nagy hőmérséklet hatására nagyobb energiával rendelkeznek, és szabadabban rendeződnek el. Ahogy a hőmérséklet csökken, úgy csökken az atomok energiája, és fokozatosan a lehető legszilárdabb állapot alakul ki. Látható, hogy a módszert optimalizációs feladatok esetében alkalmazzák. Számunkra azért érdekes, mert egyfelől önmagában alternatívát ad a genetikus algoritmussal szemben, másrészt egyes alkalmazások összekapcsolják a két módszert, és igyekeznek mindkét módszer előnyeit kiaknázni.

Optimalizáló neurális hálók. Mint említettük, a neurális hálózatot igen sok más feladatban is alkalmazhatjuk, így optimalizálásra is. A *10. ábra* az optimalizáló neurális hálók egy lehetséges felépítését: a *Hopfield-hálót* mutatja be (*Hopfield–Tank* [1985]).

A Hopfield-hálózat esetében nem a súlyok alakulnak ki a tanulás folyamán, hanem az egyes y értékek kerülnek egyre közelebb optimális értékükhöz. A súlyokat (p) és az induló x értékeket a probléma alapján analitikusan meg lehet határozni. Ezután alkalmazva a hálót, megkapjuk az első periódus y értékeit. A következő periódusban ezek lesznek a bementi változók (x), és így kapjuk a második periódus eredményét. A folyamat az optimális megoldáshoz konvergál. Sok esetben azonban a Hopfield-háló lokális optimumban áll meg, ezért sokszor a neurális optimalizáló módszerek egy további verzióját, a Boltzmann-gépet (*Boltzmann machine*) alkalmazzák, amely tulajdonképpen a simulated annealingnek és a Hopfield-hálónak az összekapcsolása (lásd *Aarts–Korst* [1989]).

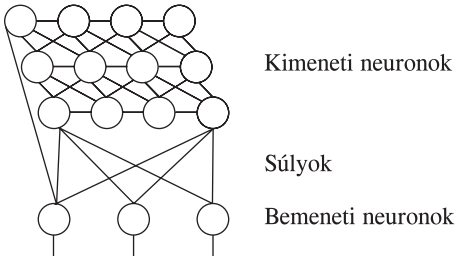
10. ábra
A Hopfield-háló felépítése



Kohonen-hálózat. Ezt a neurális hálózatot *Kohonen* [1989] fejlesztette ki, és olyan esetekben alkalmazzák, amikor nagyszámú és különböző tulajdonságú egyedet kell előre nem ismert tényezők alapján szegmentálni. A hálózatban csak input- és outputneuronok vannak, mint ahogy azt a *11. ábra* mutatja.

Az ábrán látható hálózat egy 4×3-as négyzetrács pontjaiba helyezi el az egyes egyedeket. Amennyiben két egyed ugyanabba – vagy egymáshoz közel lévő – neuronba érkezik, abban az esetben a bemeneti neuronokba beadott tulajdonságaik alapján hasonlónak, amennyiben pedig távol vannak egymástól, akkor különbözőnek mondhatók. Ezért az egyes rácspontok (outputneuronok) klaszтereknek felelnek meg. (Kohonen-háló előrejelzési modellezésben való alkalmazását lásd például: *Benedek* [1999*a*].)

11. ábra
A Kohonen-háló felépítése



Genetikus algoritmus és neurális háló. A neurális hálózat tanulási folyamatát többek között genetikus optimalizációval is meg lehet valósítani. Ekkor a súlyok képviselik a genetikus algoritmus egyedeit és az optimalizálandó függvény az illeszkedés jósága (a várt és az aktuális értékek közti távolság, amelyet minimalizálni kell). Egyes neurálishálómodellek automatikusan tudják változtatni a köztes réteg(ek) struktúráját. Ennek megvalósítása is elképzelhető genetikus algoritmus segítségével.

Irodalom

Aarts, E.–Korst, J. [1989]: Simulated annealing and Boltzmann machines. Wiley, Chichester, Egyesült Királyság.

Baker, J. E. [1985]: Adaptive selection methods for genetic algorithms. Proc. 1st Int. Conf. on Genetic Algorithms and their Applications, Lawrence Erlbaum Associates, Hillsdale, NJ.

Balla Katalin–Benedek Gábor [1999]: Riccati Extended Forms. 6th Colloquium on the Qualitative Theory of Differential Equations, augusztus.

Benedek Gábor [1999*a*]: Mesterséges inteligencia az üzleti életben – Marketingakciók hatékonyságának vizsgálata statisztikai és Data Mining módszerekkel. Vezetéstudomány, november.

Benedek Gábor [1999*b*]: Opcióárazás numerikus módszerekkel. Közgazdasági Szemle, 10. sz.

Bigus, J. P. [1996]: Data mining with neural networks: solving business problems. McGraw-Hill, New York.

Davis, L. [1991]: Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York, NY.

Fogarty, T. C. [1989]: Varying the probability of mutation in the genetic algorithms. Proc. 3rd Int. Conf. on Genetic Algorithms and their Applications, George Mason University.

Fu, L. [1994]: Neural networks in computer intelligence, McGraw-Hill, Inc., Egyesült Államok.

Grefenstette, J. J. [1986]: Optimisation of control parameters for genetic algorithms. IEEE Trans. On Systems, Man and Cybernetics, Vol. SMC-16, No. 1.

HIMANEN, V. [1998] Neural networks in transport applications Megjelent: *Himanen, V.* (szerk.): Ashgate, Aldershot.

HOLLAND, J. H. [1975]: Adaptation in natural and artificial systems, University of Michigan Press, Ann Arbor, MI.

HOLMES, J. N. [1993]: Speech synthesis and recognition. Chapman and Hall, London.

HOPFIELD, J. J.–TANK, D. W. [1985]: „Neural" computation of decisions in optimization problems. Biol. Cybern., Vol. 52.

HULL, J. C. [1993]: Options, Futures, and other Derivative Securities. 2. kiadás, Prentice-Hall International, Inc., Engle Wood Cliffs, New Jersey.

INSTITUTE OF ELECTRICAL AN... [1989]: Neural network based speech recognition systems. (Technical tutorial seminar.) I.E.E.E., Piscataway, N. J.

KIRKPATRIC, S.–GELATT, C. D. JR–VECCHI, M. P. [1983]: Optimization by simulated annealing. Science, Vol. 220.

KOCSIS ÉVA –SZABÓ KATALIN [2000]: A posztmodern vállalat. Oktatási Minisztérium, Budapest.

KOHONEN, T. [1989]: Self-organisation and associative memory. Springer-Verlag, Berlin.

MAMMONE, R. J. (szerk.)[1991]: Neural networks: theory and applications. Academic Press, Boston.

MAREN, A. J–HARST, C. T. [1990]: Handbook of neural computing applications. Academic Press, San Diego.

MICHALEWICZ, Z. [1992]: Genetic algorithms + Data structures = Evolution programs. Springer-Verlag, New York, NY.

PHAM, D. T.–KARABOGA, D. [1998]: Intelligent optimasation Techniques. Springer-Verlag, London.

SCHAFFER, J. D.–CARUANA, R. A.–ESHELMAN, L. J.–DAS, R. [1989]: A study of control parameters affecting on-line performance of genetic algorithms for function optimasation. Proc. 3rd Int. Conf. on Genetic Algorithms and their Applications, George Mason University.

SIMONOVITS ANDRÁS [1998]: Matematikai módszerek a dinamikus közgazdaságtanban. Közgazdasági és Jogi Könyvkiadó, Budapest.

WHITELY, D.–HANSON, T. [1989]: Optimising neural networks using faster, more accurate genetic search. Proc. 3rd Int. Conf. on Genetic Algorithms and their Applications, George Mason University.

WINTER, G. (szerk.) [1995]: Genetic algorithms in engineering and computer science. Wiley, Chichester.