Mesterséges intelligencia alapjai jegyzet

Molnár Antal Albert

2020. június 23.

Előszó

Ez a jegyzet a Debreceni Egyetemen a Kádek Tamás által oktatott, *INBPM0418E* tárgykódú *A mesterséges intelligencia alapjai* tárgyhoz nyújt némi segítséget a vizsgára készülő hallgatóknak.

A leírtakért semmilyen felelősséget nem tudok vállalni, hiszen még jómagam is csak ismerkedem a mesterséges intelligencia világával.

A jegyzet nagyrészt az órákon tárgyalt könyvből[1], valamint Várterész Magda előadásaiból[2] merít.

Tartalomjegyzék

1. fejezet

Témakörök, melyeket mélységben ismerni kell

1.1. Ágens szemlélet

1.1.1. Az ágens fogalma

1. Definíció. Ágens. Egy **ágens** (agent) bármi lehet, amit úgy tekinthetünk, mint ami az **érzékelői** (sensors) segítségével érzékeli a **környezetét** (environment), és **beavatkozói** (actuators) segítségével megváltoztatja azt.

Az emberi ágensnek van szeme, füle és egyéb szervei az érzékelésre, és keze, lába, szája és egyéb testrészei a beavatkozásra.

A robotágens kamerákat és infravörös távolsági keresőket használ érzékelőként, és különféle motorokat beavatkozóként.

A szoftverágens billentyűleütéseket, fájltartalmakat és hálózati adatcsomagokat fogad érzékelőinek bemeneteként, és képernyőn történő kijelzéssel, fájlok írásával, hálózati csomagok küldésével avatkozik be a környezetébe.

Azzal az általános feltételezéssel fogunk élni, hogy minden ágens képes saját akcióinak érzékelésére (de nem mindig látja azok hatását).

2. Definíció. Érzékelés. Az érzékelés (percept) fogalmat használjuk az ágens érzékelő bemeneteinek leírására egy tetszőleges pillanatban.

3. Definíció. Ágens érzékelési sorozata. Egy ágens érzékelési sorozata (percept sequence) az ágens érzékeléseinek teljes története, minden, amit az ágens valaha is érzékelt. Általánosságban, egy adott pillanatban egy ágens cselekvése az addig megfigyelt teljes érzékelési sorozatától függhet. Ha az összes lehetséges érzékelési sorozathoz meg tudjuk határozni az ágens lehetséges cselekvéseit, akkor lényegében mindent elmondtunk az ágensről.

4. Definíció. Ágensfüggvény. Matematikailag megfogalmazva azt mondhatjuk, hogy az ágens viselkedését az ágensfüggvény (agent function) írja le, ami az adott érzékelési sorozatot egy cselekvésre képezi le.

5. Definíció. Ágensprogram. Egy mesterséges ágens belsejében az ágensfüggvényt egy ágensprogram (agent program) valósítja meg.

1. Megjegyzés. Különbség ágensfüggvény és ágensprogram között. Az ágensfüggvény egy absztrakt matematikai leírás, az ágensprogram egy konkrét implementáció, amely az ágens architektúráján működik.

1.1.2. Az ágens jellemzése (teljesítmény, környezet, érzékelők, beavatkozók

6. Definíció. TKBÉ (PEAS) leírás. Egy ágens tervezése során az első lépésnek mindig a feladatkörnyezet lehető legteljesebb meghatározásának kell lennie.

Ágenstípus	**Tel-jesítmény-mérték (Perfor-mance)**	**Környezet (En-vironment)**	**Beavatkozók (Actuators)**	**Érzékelők (Sen-sors)**
Taxisofőr	Biztonságos, gyors, törvényes, kényelmes utazás, maximum haszon	Utak, egyéb forga-lom, gyalogosok, ügyfelek	Kormány, gáz, fék, index, kürt, kijel-ző	Kamerák, hangra-das, sebességmé-rő, GPS, kilomé-teróra, motorérzé-kelők, billentyűzet

7. Definíció. Teljesen megfigyelhető környezet.

Ha az ágens szenzorai minden pillanatban hozzáférést nyújtanak a környezet teljes állapotához, akkor azt mondjuk, hogy a környezet teljesen megfigyelhető.

8. Definíció. Determinisztikus (deterministic) vagy sztochasztikus (stochastic).

Amennyiben a környezet következő állapotát jelenlegi állapota és az ágens által végrehajtott cselekvés teljesen meghatározza, akkor azt mondjuk, hogy a környezet determinisztikus, egyébként sztochasztikus.

9. Definíció. Epizódszerű (episodic) vagy sorozatszerű (sequential) környezet.

Epizódszerű környezetben az ágens tapasztalata elemi "epizódokra" bontható. Minden egyes epizód az ágens észleléseiből és egy cselekvéséből áll. Nagyon fontos, hogy a következő epizód nem függ az előzőben végrehajtott cselekvésektől. Epizódszerű környezetekben az egyes epizódokban az akció kiválasztása csak az aktuális epizódtól függ.

10. Definíció. Statikus (static) vagy dinamikus (dynamic) környezet.

Ha a környezet megváltozhat, amíg az ágens gondolkodik, akkor azt mondjuk, hogy a környezet az ágens számára dinamikus; egyébként statikus.

11. Definíció. Diszkrét (discrete) vagy folytonos (continuous) környezet.

A diszkrét/folytonos felosztás alkalmazható a környezet állapotára, az időkezelés módjára, az ágens észleléseire, valamint cselekvéseire. Például egy diszkrét állapotú környezet, mint amilyen a sakkjáték, véges számú különálló állapottal rendelkezik. A sakkban szintén diszkrét az akciók és cselekvések halmaza. A taxivezetés folytonos állapotú és idejű probléma: a sebesség, a taxi és más járművek helye folytonos értékek egy tartományát járja végig a folytonos időben.

12. Definíció. Egyágenses (single agent) vagy többágenses (multiagent) környezet.

Az egyágenses és többágenses környezetek közötti különbségtétel egyszerűnek tűnhet. Például a keresztrejtvényt megfejtő ágens önmagában nyilvánvalóan egyágenses környezetben van, míg egy sakkozó ágens egy kétágensesben. Vannak azonban kényes kérdések. Először is: leírtuk azt, hogy egy entitás hogyan tekinthető ágensnek, ugyanakkor nem magyaráztuk meg, mely entitások tekintendők ágensnek. Egy A ágensnek (például a taxisofőrnek) egy B objektumot (egy másik járművet) ágensnek kell tekintenie, vagy egyszerűen egy sztochasztikusan viselkedő dolognak, a tengerparti hullámokhoz vagy a szélben szálló falevelekhez hasonlatosan? A választás kulcsa az, hogy vajon B viselkedése legjobban egy A viselkedésétől függő teljesítménymérték maximalizálásával írhatóe le. Például a sakkban a B ellenfél saját teljesítménymértékét próbálja maximalizálni, amely – a sakk szabályainak következtében – A teljesítménymértékét minimalizálja. Így a sakk egy versengő (**competitive**) többágenses környezet. Másrészről, a taxi vezetési környezetben az ütközések elkerülése az összes ágens teljesítménymértékét maximálja, így az részben kooperatív (**cooperative**) többágenses környezet. Emellett részben versengő is, hiszen például csak egy autó tud egy parkolóhelyet elfoglalni. A többágenses környezetekben felmerülő ágenstervezési problémák gyakran egészen mások, mint egyágenses környezetekben. Többágenses környezetekben például a kommunikáció (communication) gyakran racionális viselkedésként bukkan fel; egyes részlegesen megfigyelhető versengő környezetekben a sztochasztikus viselkedés racionális, hiszen így elkerülhetők a megjósolhatóság csapdái.

1.2. Állapottér reprezentáció

1.2.1. Az állapottér fogalma

Az állapottér-reprezentáció az egyik leggyakrabban használt reprezentációs mód ami egy probléma formális megadására szolgál.

13. Definíció. Probléma. Egy **probléma** (problem) formális megragadásához az alábbi négy komponensre van szükség:

- **kiinduló állapot**: amiből az ágens kezdi a cselekvéseit,
- **cselekvések halmaza**: ágens rendelkezésére álló lehetséges cselekvések,
- **állapotátmenet-függvény**: visszaadja a rendezett ⟨ cselekvés, utódállapot ⟩ párok halmazát (Egy alternatív megfogalmazás az **operátor**ok egy halmaza, amelyeket egy állapotra alkalmazva lehet az utódállapotokat generálni.) lásd még: *14. Definíció*
- **állapottér**: A kezdeti állapot és az állapotátmenet-függvény együttesen implicit módon definiálják a probléma **állapotteré**t: azon állapotok halmazát, amelyek a kiinduló állapotból elérhetők.

14. Definíció. Operátor alkalmazási előfeltétel teszt. Ahhoz, hogy meggyőződjünk arról, hogy egy adott állapotra egy adott operátor alkalmazható, előbb meg kell vizsgálnunk, hogy az állapotra alkalmazható-e az operátor. Ezt a vizsgálódást operátor alkalmazási előfeltétel tesztnek nevezzük.

15. Definíció. Célteszt. A célteszt ellenőrzi a célfeltételek teljesülését egy adott állapotban.

1.2.2. Az állapottérgráf

16. Definíció. Állapottérgráf. Az állapottér egy gráfot alkot, amelynek csomópontjai az állapotok és a csomópontok közötti élek a cselekvések.

17. Definíció. Állapottér útja. Az **állapottér egy útja** az állapotok egy sorozata, amely állapotokat a cselekvések egy sorozata köt össze.

18. Definíció. Állapottér-reprezentációs gráf bonyolultsága. Egy állapottér-reprezentált probléma megoldásának sikerét jelentősen befolyásolja a reprezentációs gráf bonyolultsága:

- a csúcsok száma,
- az egy csúcsból kiinduló élek száma,
- a hurkok és körök száma és hossza.

Ezért célszerű minden lehetséges egyszerűsítést végrehajtani. Lehetséges egyszerűsítések:

- a csúcsok számának csökkentése — ügyes reprezentációval az állapottér kisebb méretű lehet;
- az egy csúcsból kiinduló élek számának csökkentése — az operátorok értelmezési tartományának alkalmas megválasztásával érhető el;
- a reprezentációs gráf fává alakítása — a hurkokat, illetve köröket „kiegyenesítjük”

Az állapottérgráf jellemzése

b: Az **elágazási tényező** (branching factor) a tetszőleges állapotból közvetlenül elérhető állapotok maximális száma.

$$\max\{\text{card}\{b : a \Rightarrow b\} : a \in \mathcal{A})\}.$$

d : A **legsekélyebb megoldás** a legkevesebb operátoralkalmazás segítségével elérhető célállapot eléréshez szükséges operátoralkalmazások száma. (A legrövidebb megoldás hossza. A célállapotok minimális mélysége.) A legkisebb olyan i amely esetén van olyan állapotsorozat, hogy

$$a_0 \Rightarrow a_1, \quad a_1 \Rightarrow a_2 \quad \ldots \quad a_{i-1} \Rightarrow a_i, \quad a_i \in \mathcal{C}.$$

m : A **csomópontok maximális mélysége**. A legnagyobb olyan i, amely esetén van olyan állapotsorozat, hogy

$$a_0 \Rightarrow a_1, \quad a_1 \Rightarrow a_2 \quad \dots \quad a_{i-1} \Rightarrow a_i.$$

1.2.3. Költség és heurisztika fogalmak

19. Definíció. Lépésköltség. Az x állapotból az y állapotba vezető cs cselekvés **lépésköltség**e (step cost) legyen $lk(x, cs, y)$. Tételezzük fel, hogy a lépésköltség nemnegatív.

20. Definíció. Útköltség-függvény. Egy **útköltség-függvény**, egy olyan függvény amely az állapottér minden útjához hozzárendel egy költséget.

21. Definíció. Megoldás. Egy út, amely a kiinduló állapotból egy célállapotba vezet.

22. Definíció. Optimális megoldás. A legkisebb útköltségű megoldás.

1.3. Megoldáskereső algoritmusok

1.3.1. Fakereső algoritmusok

Az általános fakeresési algoritmus informális leírása:

Algorithm 1: Általános fakeresési algoritmus informális leírása

```
1  function Fa-Kereses(probléma, stratégia):
2      a probléma kezdeti állapotából kiindulva inicializáld a keresési fát
3      loop do
4          if nincs kifejtendő csomópont then
5              return kudarc
6          end if
7          a stratégiának megfelelően válassz ki kifejtésre egy levélcsomópontot
8          if a csomópont célállapotot tartalmaz then
9              return a hozzá tartozó megoldás
10         else
11             fejtsd ki a csomópontot és az eredményül kapott csomópontokat, és add a keresési fához
12         end if
13     end loop
14 end function
```

23. Definíció. Perem.

A legenerált, kifejtésre váró csomópontokat külön nyilvántartjuk, ez a **perem**. Ezt a gyakorlatban általában egy várakozási sorként (queue) szokás implementálni.

24. Definíció. Az algoritmusok hatékonyságának értékelése.

A problémamegoldó algoritmus kimenete vagy kudarc, vagy egy megoldás (egyes algoritmusok végtelen hurokba kerülhetnek és soha nem térnek vissza válasszal). Mi az algoritmusok hatékonyságát négyféle módon fogjuk értékelni:

- **Teljesség (completeness)**: az algoritmus generáltan megtalál egy megoldást, amennyiben létezik megoldás?
- **Optimalitást (optimality)**: a stratégia megtalálja az optimális megoldást?
- **Időigény (time complexity)**: mennyi ideig tart egy megoldás megtalálása?
- **Tárigény (space complexity)**: a keresés elvégzéséhez mennyi memóriára van szükség?

1.3.2. Gráfkereső algoritmusok

Eddig figyelmen kívül hagytuk a keresés egyik legfontosabb megoldandó problémáját: a már korábban egy másik úton megtalált és kifejtett állapotok ismételt kifejtéséből adódó időpazarlást. Néhány probléma esetén ezen lehetőség soha nem merül fel, mert az állapottér egy fa, és minden állapotba csak egyetlen módon lehet eljutni. A 8-királynő probléma hatékony megfogalmazása – amikor minden új királynőt a bal szélső szabad oszlopba helyezzük – nagyrészt épp ennek köszönheti hatékonyságát, vagyis hogy minden egyes állapotba csak egyetlen úton lehet eljutni. Ha a 8-királynő problémát úgy fogalmazzuk meg, hogy egy királynőt bármelyik oszlopban el lehet helyezni, akkor az n királynőt tartalmazó minden állapotot $n!$ különböző úton el lehet érni.

Számos problémánál azonban elkerülhetetlenek a megismételt állapotok. Ezek közé tartozik az összes olyan probléma, amelyben az operátorok reverzíbilisek. Többek között ebbe a csoportba tartoznak az útkeresési problémák és a csúszó lapka fejtörő játékok. Az ezen problémákhoz tartozó keresési fák végtelenek, de ha a megismételt állapotok egy részét levágjuk, akkor a keresési fát véges méretűre vághatjuk, ezáltal a keresési fának csak az állapottér gráfot kifeszítő részét generálva.

Az ismétlődő állapotok tehát a megoldható problémákat megoldhatatlan problémákká alakítják, amennyiben az algoritmus nem képes ezeket az állapotokat detektálni. A detektálás általában azt jelenti, hogy az új kifejtendő csomópontot a már kifejtett csomópontokkal hasonlítjuk össze. Egyezés esetén az adott csomóponthoz az algoritmus két utat talált, és valamelyiket eldobhatja.

Algorithm 2: Az általános gráfkereső algoritmus

```
1  function Graf-Kereses(probléma, perem) returns egy megoldás vagy kudarc:
2      zárt lista ← egy üres halmaz
3      perem ← Beszúr(Csomópontot-Létrehoz(Kiinduló-állapot[probléma]), perem)
4      loop do
5          if Üres?(perem) then return kudarc
6          csomópont ← Vedd-Az-Első-Elemet(perem)
7          if Cél-Teszt[probléma](Állapot[csomópont]) then return Megoldás(csomópont)
8          if Állapot[csomópont] nem eleme a zárt listának then
9              adjuk hozzá az Állapot[csomópont]-ot a zárt listához
10             perem ← Beszúr-Mind(Kifejt(csomópont, probléma), perem)
11         end if
12     end loop
13 end function
```

2. Megjegyzés. A zártak listáját érdemes egy hash táblával implementálni, hogy hatékonyan tudjuk ellenőrizni a már meglátogatott csomópontokat.

Jellemzés

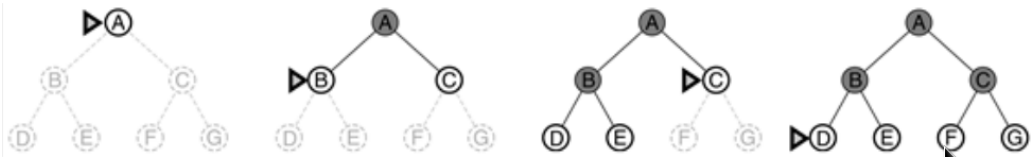
- **Időbonyolultság**: $1 + b + b^2 + \ldots + b^{d+1} - b \in O(b^{d+1})$
- **Tárbonyolultság**: $1 + b + b^2 + \ldots + b^{d+1} - b \in O(b^{d+1})$
- **Teljesség**: teljes, ha b véges,
- **Optimalitás**: optimális, ha minden költség egységnyi.

1.3.3. Szélességi kereső

A **szélességi keresés (breadth-first search)** egy egyszerű keresési stratégia, ahol először a gyökércsomópontot fejtjük ki, majd a következő lépésben az összes gyökércsomópontból generált csomópontot, majd azok követőit, stb.

Általánosságban **a keresési stratégia minden adott mélységű csomópontot hamarabb fejt ki, mielőtt bármelyik, egy szinttel lejjebbi csomópontot kifejtené**.

A szélességi keresést meg lehet valósítani a FA-KERESÉS algoritmussal egy olyan üres peremmel, amely egy először-be-először-ki (first-in-first-out – FIFO) sor, biztosítva ezzel, hogy a korábban meglátogatott csomópontokat az algoritmus korábban fejti ki.



1.1. ábra. Szélességi keresés egy egyszerű bináris fában

Jellemzés

- **Időbonyolultság**: $1 + b + b^2 + \ldots + b^{d+1} - b \in O(b^{d+1})$
- **Tárbonyolultság**: $1 + b + b^2 + \ldots + b^{d+1} - b \in O(b^{d+1})$
- **Teljesség**: teljes, ha b véges,
- **Optimalitás**: optimális, ha minden költség egységnyi.

1.3.4. Mélységi kereső

A **mélységi keresés (depth-first search)** mindig a keresési fa aktuális peremében a legmélyebben fekvő csomópontot fejti ki. A keresés azonnal a fa legmélyebb szintjére jut el, ahol a csomópontoknak már nincsenek követőik. Kifejtésüket követően kikerülnek a peremből és a keresés "visszalép" ahhoz a következő legmélyebben fekvő csomóponthoz, amelynek vannak még ki nem fejtett követői.

Ez a stratégia egy olyan FA-KERESÉS függvénnyel implementálható, amelynek a sorbaállító függvénye az *utolsónak-be-elsőnek-ki (last-in-first-out, LIFO)*, más néven verem. A mélységi keresést szokás a FA-KERESÉS függvény alternatívájaként egy rekurzív függvénnyel is implementálni, amely a gyermekcsomópontokkal meghívja önmagát.

A mélységi keresés nagyon szerény tárigényű. Csak egyetlen, a gyökércsomóponttól egy levélcsomópontig vezető utat kell tárolnia, kiegészítve az út minden egyes csomópontja melletti kifejtetlen csomópontokkal. Egy kifejtett csomópont el is hagyható a memóriából, feltéve, hogy az összes leszármazottja meg lett vizsgálva. Egy b elágazási tényezőjű és m maximális mélységű állapottér esetén a mélységi keresés tárigénye $b \cdot m + 1$.

Jellemzés (fakeresőként)

- **Időbonyolultság**: $1 + b + b^2 + \ldots + b^m \in O(b^m)$
- **Tárbonyolultság**: $1 + b \cdot m \in O(b \cdot m)$, feltéve, hogy minden olyan csomópontot elhagyunk, amely összes leszármazottja meg lett vizsgálva
- **Teljesség**: csak véges körmentes gráfban teljes
- **Optimalitás**: nem garantál optimális megoldásokat

1.3.5. Visszalépéses kereső

A mélységi keresés **visszalépéses keresésnek (backtracking search)** nevezett változata még kevesebb memóriát használ. A visszalépéses keresés az összes követő helyett egyidejűleg csak egy követőt generál. Minden részben kifejtett csomópont emlékszik, melyik követője jön a legközelebb. Ily módon csak $O(m)$ memóriára van szükség, $O(b \cdot m)$ helyett. A visszalépéses keresés még egy memória- (és idő-) spóroló trükkhöz folyamodik. Az ötlet a követő csomópont generálása az aktuális állapot módosításával, anélkül hogy az állapotot átmásolnánk. Ezzel a memóriaszükséglet egy állapotra és $O(m)$ cselekvésre redukálódik. Ahhoz, hogy az ötlet működjön, amikor visszalépünk, hogy a következő követőt generáljuk, mindegyik módosítást vissza kell tudnunk csinálni. Nagy állapottérrel rendelkező problémák esetén, mint például robot-összeszerelés esetén, az ilyen módszerek lényegesek a sikerességhez.

A mélységi keresés hátrányos tulajdonsága, hogy egy rossz választással egy hosszú (akár végtelen) út mentén lefelé elakadhat, miközben például egy más döntés elvezetne a gyökérhez közeli megoldáshoz. A legrosszabb esetben a mélységi keresés a keresési fában az összes $O(b^m)$ csomópontot generálni fogja, ahol m a csomópontok maximális mélysége. Jegyezzük meg, hogy m sokkal nagyobb lehet, mint d (a legsekélyebb megoldás mélysége), és korlátlan fák esetén értéke végtelen.

3. Megjegyzés. A visszalépéses kereső használatával a mélységi keresés tárbonyolultsága csökkenthető tovább.

Jellemzés

- **Időbonyolultság**: $1 + b + b^2 + \ldots + b^m \in O(b^m)$
- **Tárbonyolultság**: $1 + m \in O(m)$
- **Teljesség**: csak véges körmentes gráfban teljes
- **Optimalitás**: nem garantál optimális megoldásokat

1.3.6. Egyenletes költségű (optimális kereső)

A szélességi keresés optimális, ha minden lépés költsége azonos, mert mindig a legsekélyebb ki nem fejtett csomópontot fejti ki. Egyszerű általánosítással egy olyan algoritmust találhatunk ki, amely tetszőleges lépésköltség mellett optimális. Az **egyenletes költségű keresés (uniform cost search)** mindig a legkisebb útköltségű n csomópontot fejti ki először, nem pedig a legkisebb mélységű csomópontot. Egyszerűen belátható, hogy a szélességi keresés is egyenletes költségű keresés, amennyiben minden lépésköltség azonos.

Az egyenletes költségű keresés nem foglalkozik azzal, hogy *hány* lépésből áll egy bizonyos út, hanem csak az összköltségükkel törődik. Emiatt mindig végtelen hurokba kerül, ha egy csomópont kifejtése zérus költségű cselekvéshez és ugyanahhoz az állapothoz való visszatérést eredményez (például a NOOP cselekvés).

A teljességet csak úgy garantálhatjuk, hogy minden lépés költsége egy kis pozitív e konstansnál nagyobb, vagy azzal egyenlő. Ez a feltétel egyben az optimalitás elégséges feltétele is. Ez azt jelenti, hogy egy út költsége az út mentén mindig növekszik. Ebből a tulajdonságból látszik, hogy az algoritmus a csomópontokat mindig a növekvő útköltség függvényében fejti ki. Azaz az első kifejtésre kiválasztott célcsomópont egyben az optimális megoldás is (emlékezzünk arra, hogy a FA-KERESÉS a célállapottesztet csak a kifejtésre megválasztott csomópontokra alkalmazza).

1.3.7. Legjobbat először kereső

Informált keresési módszer.

A **mohó legjobbat-először keresés (greedy best-first search)** azt a csomópontot fejti ki a következő lépésben, amelyiknek az állapotát a legközelebbinek ítéli a célállapothoz, abból kiindulva, hogy így gyorsan megtalálja a megoldást. A csomópontokat az algoritmus tehát az $f(n) = h(n)$ heurisztikus függvénnyel értékeli ki.

Jellemzés

- **Időbonyolultság**: $O(b^m)$
- **Tárbonyolultság**: $O(b^m)$
- **Teljesség**: nem teljes
- **Optimalitás**: nem optimális

4. Megjegyzés. Az idő- és tárbonyolultság nagyban függ a heurisztikus függvény minőségétől.

1.3.8. Az A* algoritmus

Informált keresési módszer.

A legjobbat-először keresés leginkább ismert változata az A* keresés. A csomópontokat úgy értékeli ki, hogy összekombinálja $g(n)$ értékét – az aktuális csomópontig megtett út költsége – és $h(n)$ értékét – vagyis az adott csomóponttól a célhoz vezető út költségének becslőjét:

$$f(n) = g(n) + h(n).$$

Mivel $g(n)$ megadja a kiinduló csomóponttól az n csomópontig számított útköltséget, és h(n) az n csomóponttól a célcsomópontba vezető legolcsóbb költségű út költségének becslője, így az alábbi összefüggést kapjuk:

$$f(n) = \text{a legolcsóbb, az n csomóponton keresztül vezető megoldás becsült költsége.}$$

Így amennyiben a legolcsóbb megoldást keressük, ésszerű először a legkisebb $g(n) + h(n)$ értékkel rendelkező csomópontot kifejteni. Ezen stratégia kellemes tulajdonsága, hogy ez a stratégia több mint ésszerű: amennyiben a h függvény eleget tesz bizonyos feltételeknek, az A* keresés teljes és optimális.

Jellemzés

- **Időbonyolultság**: $O(b^m)$
- **Tárbonyolultság**: $O(b^m)$
- **Teljesség**: teljes
- **Optimalitás**: optimális, ha $h(n)$ konzisztens (vagyis monoton), valamint elfogadható heurisztika, azaz soha nem becsül felül (ez következik a konzisztenciából)

1.4. Kétszemélyes játékok

1.4.1. A játékok reprezentációja

A továbbiakban a

- teljesen megfigyelhető,
- véges,
- determinisztikus,
- kétszemélyes,
- zérusösszegű

játékokkal foglalkozunk.

25. Definíció. Játék reprezentációja. Egy játék reprezentációja megadható a

$$\langle \mathcal{B}, b_0, \mathcal{J}, \mathcal{V}, \hat{v}, \mathcal{L} \rangle$$

rendezett hatossal, ahol:

- $\mathcal{B}$: a játékállások halmaza,
- b_0: a kezdőállás, ahol $b_0 \in \mathcal{B}$,
- $\mathcal{J}$: a játékosok halmaza, ahol $\operatorname{card} \mathcal{J} = 2$,
- $\mathcal{V}$: a végállások halmaza, ahol $\mathcal{V} \subseteq \mathcal{B}$,
- $\hat{v}$: egy $\mathcal{V} \to \{-1, 0, 1\}$ függvény,
- $\mathcal{L}$: a lépések halmaza.

26. Definíció. Nyertes. Végállásban a $\hat{v}$ függvény határozza meg, hogy melyik játékos nyert:

$$\hat{v}(b) = \begin{cases} 1 & \text{ha } b \text{ állásban a következő játékos nyer} \\ 0 & \text{ha } b \text{ állásban a következő játékos veszít} \\ -1 & \text{egyébként (döntetlen esetén)} \end{cases}.$$

27. Definíció. Lépés. Minden $l \in \mathcal{L}$ egy $\mathcal{B} \to \mathcal{B}$ parciális függvény.

28. Definíció. Játék állapottere. Legyen $\langle \mathcal{B}, b_0, \mathcal{J}, \mathcal{V}, \hat{v}, \mathcal{L} \rangle$ egy játék reprezentációja. Ekkor a **játék állapottere** $\langle \mathcal{A}, a_0, \mathcal{C}, \mathcal{O} \rangle$ definiálható a következőképpen:

- $\mathcal{A} = \mathcal{B} \times \mathcal{J}$;
- $a_0 = \langle b_0, j_0 \rangle$, ahol $j_0 \in \mathcal{J}$ a kezdőjátékos;
- $\mathcal{C} = \{\langle b, j \rangle : \langle b, j \rangle \in A \wedge b \in \mathcal{V}\}$
- $\mathcal{O} = \{o_l : l \in \mathcal{L}\}$, ahol $o_l : \mathcal{A} \to \mathcal{A}$, úgy hogy
  - $\mathrm{dom}(o_l) = \{\langle b, j \rangle : b \in \mathrm{dom}(l)\}$
  - $o_l(\langle b, j \rangle) = \langle l(b), j' \rangle$
  - $j' \in (\mathcal{J} \setminus \{j\})$

29. Definíció. Közvetlen elérhetőség. Az $a \in \mathcal{A}$ állapotból az $a' \in \mathcal{A}$ állapot **közvetlenül elérhető**, ha van olyan $o_l \in \mathcal{O}$, amely esetén

$$a \in \mathrm{dom}(o_l) \text{ és } o_l(a) = a'$$

és ezt $a \Rightarrow a'$ alakban jelöljük.

30. Definíció. Elérhetőség. $Aza \in \mathcal{A}$ állapot az $a' \in \mathcal{A}$ állapot **elérhető** ($a \Rightarrow^* a'$), ha

- $a = a'$, vagy
- van olyan $a_1, a_2, \ldots, a_k$ állapotsorozat, hogy $a_1 = a, a_k = a'$ továbbá

$$a_i \Rightarrow a_{i+1} \text{ minden } i \in \{1, \ldots, k-1\} \text{ esetén.}$$

1.4.2. A játékfa

31. Definíció. Játékfa. Legyen $\langle \mathcal{B}, b_0, \mathcal{J}, \mathcal{V}, \hat{v}, \mathcal{L} \rangle$ egy játék reprezentációja és $j_0 \in \mathcal{J}$ a kezdőjátékos. Ekkor a játékfa olyan fa, melynek csúcsaihoz a játék állapotait rendeljük:

- a fa gyökere a $\langle a_0, j_0 \rangle$ állapottal címkézett csúcs,
- a fa levélelemei olyan $\langle b, j \rangle$ állapottal címkézett csúcsok, ahol $b \in \mathcal{V}$
- a fa $\langle b, j \rangle$ állapottal címkézett nem levélcsúcsának gyermekeit olyan $\langle b', j' \rangle$ címkéjű csúcsok alkotják, ahol $\langle b, j \rangle \Rightarrow \langle b', j' \rangle$

5. Megjegyzés. A játékfa a játék állapotterének gráfját fává egyenesíti ki, egy-egy állapot a fában több csúcs címkéjeként is szerepelhet.

1.4.3. Nyerő stratégia

32. Definíció. Játszma. Egy **játszma** egy olyan lépéssorozat, amely a $\langle b_0, j_0 \rangle$ kezdőállapotból valamely $\langle b, j \rangle \in \mathcal{C}$ célállapotba vezet. (Egy a gyökértől valamely levélcsúcsig vezető út a játékfában).

33. Definíció. Stratégia. Játékterv, ami minden olyan állásban, amikor ő következik, megmondja a játékosnak, hogy mit lépjen.

A $j \in \mathcal{J}$ játékos **stratégiája** olyan $S_j : \{\langle b, j \rangle : \langle b, j \rangle \in \mathcal{A}\} \to \mathcal{O}$ leképezés (döntési terv), amely j számára előírja, hogy a játék azon állapotaiban, ahol j a lépni következő játékos, a megtehető lépések közül melyiket lépje meg.

34. Definíció. Nyerő stratégia.

A $j \in \mathcal{J}$ játékos egy S_j stratégiája **nyerő stratégia**, ha minden a stratégia mellett lejátszható játszmában j nyer.

Ha döntetlen állhat elő – $0 \notin \operatorname{rng}(\hat{v})$ –, akkor az egyik játékos rendelkezik nyerő stratégiával. A nyerő stratégiával rendelkező játékos $w(a_0, j_0)$:

$$w(b,j) = \begin{cases} j & \text{ha } b \in \mathcal{V} \text{ és } \hat{v}(b) = 1 \\ j' & \text{ha } b \in \mathcal{V} \text{ és } \hat{v}(b) = -1 \\ j & \text{ha van olyan } \langle b', j' \rangle \text{ hogy } \langle b, j \rangle \to \langle b', j' \rangle \, sw(b', j') = j \\ j' & \text{egyébként,} \end{cases}$$

ahol $j' \in (\mathcal{J} \setminus \{j\})$.

6. Megjegyzés. Ha a játékban van döntetlen végállás, akkor az egyik játékosnak van garantáltan nem vesztő stratégiája.

1.5. Lépésajánló algoritmusok

35. Definíció. Hasznosságfügvény.

A **hasznosságfügvény** egy becslés, de elávrjuk, hogy végállásban pontos legyen. Formálisan:

$$h : \mathcal{A} \to [-m, m] \text{ ahol } m \in \mathbb{R}^+ \text{ továbbá ha } b \in \mathcal{V} \text{ akkor } h(b,j) = m \cdot \hat{v}(b).$$

Ha $h(b,j) > h(b',j)$ akkor a b állás a j (lépni következő) játékos számára kedvezőbb, mint a b' állás.

Legyen h egy hasznosságfüggvény, ekkor h_i az $i \in \mathcal{J}$ játékos hasznosságfüggvénye, ha

$$h_i(a,j) = \begin{cases} h(a,j) & \text{ha } i = j \\ -h(a,j) & \text{egyébként.} \end{cases}$$

1.5.1. MinMax módszer

Az $b \in \mathcal{B}$ állás hasznossága $t \in \mathcal{J}$ támogatott játékos számára ha $j \in \mathcal{J}$ játékos következik lépni, $k \in \mathbb{N}$ mélységi korláttal rekurzívan a következőképpen számítható:

$$f(b,j,t,k) = \begin{cases} h_t(b,j) & \text{ha } b \in \mathcal{V}, \\ h_t(b,j) & \text{ha } k = 0, \\ \min\{f(b',j',t,k-1) : \langle b,j \rangle \Rightarrow \langle b',j' \rangle\} & \text{ha } t \neq j, \\ \max\{f(b',j',t,k-1) : \langle b,j \rangle \Rightarrow \langle b',j' \rangle\} & \text{ha } t = j. \end{cases}$$

1.5.2. NegaMax módszer

Az $b \in \mathcal{B}$ állás hasznossága $j \in \mathcal{J}$ (lépni következő) játékos számára $k \in \mathbb{N}$ mélységi korláttal rekurzívan a következőképpen számítható:

$$\hat{f}(b,j,k) = \begin{cases} h(b,j) & \text{ha } b \in \mathcal{V}, \\ h(b,j) & \text{ha } k = 0, \\ max\{-\hat{f}(b',j',k-1) : \langle b,j \rangle \Rightarrow \langle b',j' \rangle\} & \text{egyébként.} \end{cases}$$

1.5.3. Alfa-béta nyesés

A gyakorlatban a MinMax algoritmus javított változatát szokás használni, az alfa-béta nyesést.

36. Definíció. Az ajánlott lépés. A $b \in \mathcal{B}$ állásban **ajánlott lépés** a $j \in \mathcal{J}$ (lépni következő) játékos számára olyan $l \in \mathcal{L}$ lépés, amely esetén

- $b \in \operatorname{dom} l$,
- $o_l(\langle b, j\rangle) = \langle b', j'\rangle$, és
- $f(b, j, j, k) = f(b', j', j, k-1)$.

1.6. Élkonzisztencia algoritmusok

37. Definíció. Legyen $\mathcal{V}$ változók egy tetszőleges halmaza, $\mathcal{D}$ pedig egy olyan leképezés, amely minden $x \in \mathcal{V}$ változóhoz egy $\mathcal{D}(x)$-el jelölt halmazt (tartományt) rendel.

Tetszőleges két különböző $x \in \mathcal{V}$ és $y \in \mathcal{V}$ változó közti bináris kényszer egy $\mathcal{D}(x)$ és $\mathcal{D}(y)$ feletti bináris reláció:

$$R_{x,y} \subseteq \mathcal{D}(x) \times \mathcal{D}(y).$$

Jelöljük $R_{x,y}^{-1}$ formában az $R_{x,y}$ reláció inverzét:

$$R_{x,y}^{-1} = \{(b, a) : (a, b) \in R_{x,y}\}.$$

38. Definíció. Véges, bináris kényszerkielégítési probléma. Egy **véges, bináris kényszerkielégítése probléma** alatt olyan $\langle \mathcal{V}, \mathcal{D}, \mathcal{C}\rangle$ rendezett hármast értünk, ahol

- $\mathcal{V}$ változók egy véges, nemüres halmaza
- $\mathcal{D}(x)$ minden $x \in \mathcal{V}$ változó esetén egy véges halmaz(tartomány)
- $\mathcal{C}$ a $\mathcal{V}$-beli változók feletti bináris kényszerek halmaza, úgy hogy
  - ha $R_{x_y} \in \mathcal{C}$ és $R'_{x,y} \in \mathcal{C}$ akkor $R_{x,y} = R'_{x,y}$.
  - ha $R_{x_y} \in \mathcal{C}$ akkor $R_{y,x} \in \mathcal{C}$ és $R_{y,x} = R_{x,y}^{-1}$.

39. Definíció. Gráf reprezentáció. Egy $\langle \mathcal{V}, \mathcal{D}, \mathcal{C}\rangle$(nem feltétlenül véges) **bináris kényszerkielégítési probléma gráf reprezentációja** egy olyan $\langle N, E\rangle$ pár, ahol:

- $N = \{n_v : v \in \mathcal{V}\}$ alkotja a gráf csúcsait,
- $E = \{(n_x, n_y) : R_{x,y} \in \mathcal{C}\}$ alkotja a gráf irányított éleit.

7. Megjegyzés. Mivel $R_{x,y}$ és $R_{y,x}$ bináris kényszerek ugyanazt fejezik ki, ezért az irányított élek helyett irányítatlan éleket tartalmazó gráfot szokás használni.

Annak, hogy $R_{x,y}$ és $R_{y,x}$ együtt szerepel a kényszerek között, a később bemutatott algoritmusokban lesz jelentősége.

40. Definíció. Nem bináris kényszerek.

Unáris kényszerek: Az unáris kényszerek egyetlen változó értékét korlátoznák. A helyett, hogy ezt kényszerként jelenítenénk meg, kifejezhetjük azzal, hogy a változóhoz rendelt tartomány elemeiként eleve csak az unáris kényszernek megfelelő értékeket választunk.

Magasabb rendű kényszerek: A magasabb rendű kényszerek mindig kiválthatók bináris kényszerekkel újabb változók bevezetése mellett.

41. Definíció. Kényszerek terjesztése. A **kényszerek terjesztése** során egy tekintett változó értékére vonatkozó megszorítás következményeit a vele kényszerek útján kapcsolatban álló változók értékeire vonatkozóan is kiterjesztjük, ezen változók értékeire újabb megszorításokat alkalmazva.

42. Definíció. Élkonzisztencia algoritmusok. Az **élkonzisztencia algoritmusok** feladata, a kényszerek terjesztésének hatékony megvalósítása. Ezen algoritmusok egyaránt alkalmazhatóak a keresés megkezdése előtt a probléma méretét csökkentő előfeldolgozó lépésként, vagy akár a keresés közben is. Ezek közül mi az előbbi lehetőséget vizsgáljuk meg.

1.6.1. AC1

Az AC-1 (gyakorlatban nem használt) algoritmus egyszerű naiv megközelítés segítségével mutatja be a kényszerek terjesztésének ötletét.

43. Definíció. Felülvizsgálat. Az $R_{x,y}$ **kényszer felülvizsgálat**a során a $\langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$ bináris kényszereket tartalmazó véges kényszerkielégítési problémát a $\langle \mathcal{V}, \mathcal{D}', \mathcal{C}' \rangle$ problémával helyettesítjük, ahol

$$\mathcal{D}' = \begin{cases} \{a : a \in \mathcal{D}(x) \wedge b \in D(y) \wedge \langle a, b \rangle \in R_{x,y}\} & \text{ha } z = x \\ \mathcal{D}(z) & \text{egyébként.} \end{cases}.$$

Algorithm 3: AC-1

```
1 function AC1(⟨V, D, C⟩):
2     repeat
3         D' ← D
4         forall R_{x,y} ∈ C do
5             D ← revise(R_{x,y}, ⟨V, D, C⟩)
6         end forall
7     until D = D'
8     return ⟨V, D, C⟩
9 end function
```

1.6.2. AC3

Az AC-3 algoritmus (Mackworth) egy már a gyakorlatban is használható algoritmus, amely egy sorban tartja nyilván azokat a kényszereket, amelyeket felül kell vizsgálni.

Ha $R_{x,y}$ felülvizsgálat során az x változó tartománya megváltozik, akkor minden olyan kényszer ismét bekerül a sorba, amely $R_{y,x}$ alakú.

Az algoritmus akkor fejeződik be, mikor elfogytak a sorból a kényszerek.

Az algoritmus időbonyolultsága $\mathcal{O}(n^2 \cdot d^3)$.

Algorithm 4: AC-3

```
1  function AC3(⟨V, D, C⟩):
2      while W ≠ ∅ do
3          remove an R_{x,y} constraint from W
4          D' ← D
5          D ← revise(R_{x,y}, ⟨V, D, C⟩)
6          if D' ≠ D then
7              if D(x) = ∅ then
8                  return failure
9              else
10                 forall R_{u,w} ∈ {R_{u,w} : R_{u,w} ∈ C ∧ w = x} do
11                     W ← W ∪ {R_{u,w}}
12                 end forall
13             end if
14         end if
15     end while
16     return ⟨V, D, C⟩
17 end function
```

1.6.3. AC4

Az AC-4 algoritmus (Mohr és Henderson) két előre elkészített adatszerkezettel dolgozik:

- minden $R_{x,y}$ kényszer és $v_x \in \mathcal{D}(x)$ értékhez egy számlálót rendelünk:

$$C_{x,v_x,y} = \text{card}\{v_y : v_y \in \mathcal{D}(y) \wedge \langle v_x, v_y\rangle \in R_{x,y}\},$$

- az x változó minden $v_x \in \mathcal{D}(x)$ értékéhez egy halmazt rendelünk:

$$S_{x,v_x} = \{\langle y, v_y\rangle : v_y \in \mathcal{D}(y) \wedge \langle v_y, v_x\rangle \in R_{y,x}\},$$

  amely megmutatja, hogy mely y változók mely $v_y \in \mathcal{D}(y)$ értékeihez számoltuk hozzá v_x-et.

Jelölje $\mathcal{D}_{-\langle x,v_x\rangle}$ azt a leképezést, amely $\mathcal{D}$ leképezéstől annyiban különbözik, hogy $v_x \notin \mathcal{D}(x)$, vagyis x nem veheti fel v_x értéket:

$$\mathcal{D}_{-\langle x,v_x\rangle}(z) = \begin{cases} \mathcal{D}(x) \setminus \{v_x\} & \text{ha } z = x, \\ \mathcal{D}(z) & \text{egyébként.} \end{cases}$$

Algorithm 5: AC-4 inicializálás

```
1  function AC4-initialize(⟨V, D, C⟩):
2      Compute S
3      Compute C
4      forall ⟨x, v_x⟩ ∈ {⟨x, v_x⟩ : C_{x,v_x,y} = 0} do
5          D ← D_{−⟨x,v_x⟩}
6      end forall
7      return S, C, D
8  end function
```

Algorithm 6: AC-4

```
1  function AC4(⟨V, D, C⟩):
2      S, C, D ← AC4-initialize(⟨V, D, C⟩)
3      W ← {⟨x, v_x⟩ : C_{x,v_x,y} = 0}
4      while W ≠ ∅ do
5          remove an ⟨x, v_x⟩ pair from W
6          forall ⟨y, v_y⟩ ∈ S_{x,v_x} do
7              C_{y,v_y,x} ← C_{y,v_y,x} − 1
8              if C_{y,v_y,x} = 0 and v_y ∈ D_y then
9                  D ← D_{−⟨y,v_y⟩}
10                 insert the ⟨y, v_y⟩ pair into W
11             end if
12         end forall
13     end while
14     return S, C, D
15 end function
```

Az **AC-4 algoritmus** egy váltótó-érték párokat tartalmazó munkahalmaz segítségével végzi el feladatát. Azon értékek, amelyek által C számlálója 0-ra csökken törölni kell a változó tartományából. A munkahalmazba a törtölt értékek kerülnek, mert a tőlük függő értékek számlálóit csökkenteni kell. Ezek gyors felderítésében segít S.

8. Megjegyzés. Az AC-4 algoritmus időbonyolultsága: $\mathcal{O}(n^2 \cdot d^2)$. Ennek ellenére valódi problémák esetében sok esetben az AC-3 algoritmus teljesít jobban (Richard J. Wallace).

1.6.4. Visszalépéses kereső

2. fejezet

Témakörök, melyekre rálátással kell rendelkezni

2.1. Következtetések ítéletlogikában

44. Definíció. Vonzat. Azt mondjuk, hogy az α mondat maga után vonzza a β mondatot, akkor és csakis akkor, ha minden modellben, amelyben α igaz, β szintén igaz. Ezt a következőképp jelöljük:

$$\alpha \models \beta.$$

45. Definíció. Következtetési eljárás helyessége.

Egy következtetési eljárást, amely csak vonzat mondatokat vezet le, helyesnek vagy igazságtartónak nevezzük. A helyesség egy igencsak kívánatos tulajdonság. Egy nem helyes következtetési eljárás kitalál olyan dolgokat ahogy előrehalad, olyan tűk megtalálását jelenti be, amelyek nem is léteznek.

46. Definíció. Következtetési eljárás teljessége. A teljesség tulajdonság szintén kívánatos: **egy következtetési eljárás teljes, ha képes levezetni minden vonzatmondatot**.

Valódi szénakazlak esetében, amelyek véges méretűek, nyilvánvalónak tűnik, hogy szisztematikus kutatással mindig eldönthető, hogy a tű a kazalban van-e. Sok tudásbázis esetében azonban a konzekvenciák szénakazlának mérete végtelen, és így a teljesség egy fontos kérdéssé válik.

2.1.1. Rezolúciókalkulus ítéletlogikában

2.2. Döntési fák

47. Definíció. Információs rendszer. Egy **információs rendszer** alatt egy $\mathcal{I} = \langle S, A, V, f \rangle$ négyest értünk, ahol

- S az objektumok halmaza,
- $A = \{a_1, a_2, \ldots a_n\}$ attribútumok véges nemüres halmaza, ahol minden $a \in A$ attribútomhoz tartozik egy nemüres V_a értékhalmaz (tartomány)
- V az attribútumértékek halmaza: $V = \cup_{i=1}^{k} V_{a_i}$
- $f : S \times A \to V$ függvény úgy, hogy

$$f(s, a) \in V_a \text{ minden } s \in S \text{ és } a \in A \text{ esetén.}$$

48. Definíció. Döntési tábla. **Döntési tábla** alatt olyan $\langle \mathcal{I}, d \rangle$ rendezett párt értünk, ahol

- $\mathcal{I} = \langle S, A, V, f \rangle$ egy információs rendszer,
- $d \in A$ a **döntési attribútum**.

Ekkor az attribútumok $C = A \setminus \{d\}$ részhalmaza a **feltétel attribútumok** (condition attributes) halmaza.

2.2.1. Az ID3 algoritmus

Algorithm 7: ID3

```
1  Function ID3( I, d, S', C', b):
2      if S' = ∅ then
3          return new-node-with-label(b)
4      end if
5      m ← majority(S', d)
6      if A = ∅ or ∀s ∈ S' : (f(s, d) = m) then
7          return new-node-with-label(m)
8      end if
9      a ← select(A)
10     C'' ← C' \ {a}
11     r ← new-node-with-label(a)
12     forall v ∈ V_a do
13         S'' ← {u : u ∈ S' ∧ f(u, a) = v}
14         t ← ID3(I, d, S'', C'', m)
15         add-edge(r, t, v)
16     end forall
17     return r
```

2.3. Valószínűségi következtetés

2.3.1. Bayes hálok, Bayes tétel

2.3.2. Feltételes valószínűség számítása

2.4. Neurális hálók, kitekintés

Néhány kihívás és néhány megoldás mesterséges intelligencia segítségével

- nagy tömegű adat tárolása, visszakeresése
- beszéd előállítása
  - Pocket - demo
- természetes nyelv megértése
  - Siri
  - Google Asssistant
  - Cortana
  - Alexa
- gépi fordítása
  - Google Translate
- alakfelismerés
  - Face Unlock
- robotvezérlés
  - Boston Dynamics

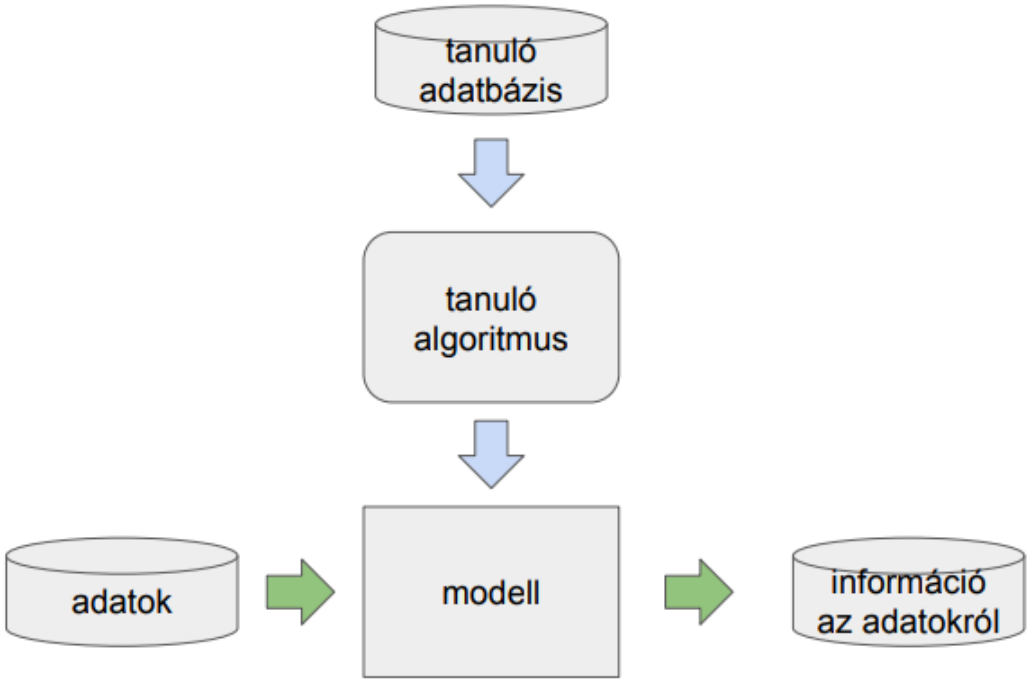
- zeneszerzés
- zenefelismerés
  - Shazam

Példák gépi tanulásra

- spamszűrés
- karakterfelismerés
- fotók címkézése
- ajánló rendszerek
- szociális háló elemzése
- hírek csoportosítása témájuk alapján

Alapfogalmak

- **Tanuló adatbázis:** párok halmaza (pl. email-címke)
- **Modell:** tudás ami alapján címkézhetünk
- **Adatok:** e-mailek
- **Információ az adatokról:** spam/nem spam címkékk



2.1. ábra. Gépi tanulás alapfogalmai

Felügyelt gépi tanulás esetén rendelkezésre állnak a közvetlen információk. Pl.: spam/nem spam, karakterek, stb. (osztályozási feladatok).

Felügyelet nélküli gépi tanulás esetén nincs közvetlen információ. Pl.: csoportosítás/klaszterezés.

49. Definíció. Regresszió.

Az eredményváltozó hogyan függ a más (magyarázó) változók alakulásától.

- lineáris: $\hat{y} = a + bx$
  - legkisebb négyzetek módszere
- nem lineáris: $y = Ae^{Bx}$
- több osztályú problémák

50. Definíció. Osztályozás.

Modell szeparáló egyenesek (hipersíkok).

Módszerek:

- Bayes osztályozók
- Döntési fák
- Support Vector Machines

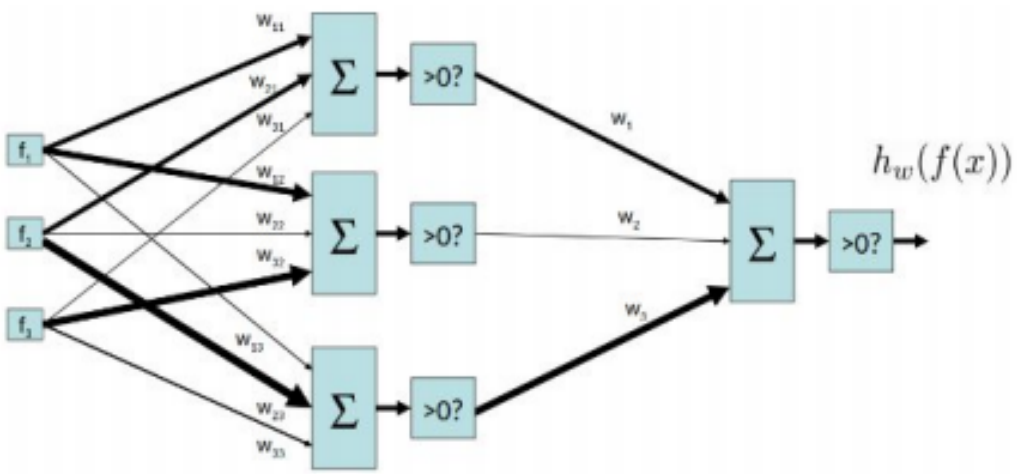
51. Definíció. Klaszterezés.

- Csoportok keresése - felügyelet nélküli tanulása
- Exkluzív (k-közép) vagy átfedő (fuzzy) módszer
- főkomponens analízás (PCA)
- hierarchikus

2.4.1. Neurális hálók és mélytanulás

Mesterséges neuronhálózat

52. Definíció. Neuron: számítási egység.



2.2. ábra. Neuronhálózat

- több x_i bemenet (mindegyikhez w_i súly rendelve)
- egy kimenet
  - izgatottság: $\sum x_i w_i$
  - ha az izgatottság túllép egy határszintet, akkor a neuron tüzel

1. Tétel. Egy kétszintű neuronhálózat elegendő számú neutronnal bármely folytonos függvényt képes megközelíteni tetszőleges pontossággal.

2. Tétel. Mesterséges neuronhálózat szerkezete:

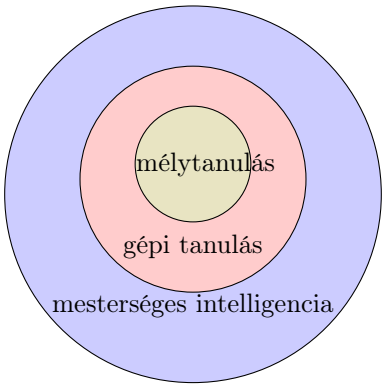
- teljesen kapcsolt réteg:
$$H = XW + b$$
- egyszerű konkurens réteg:
$$H_t = XW_x + b_x + H_{t-1}W_h + b_h$$
- konvolúciós réteg:
$$H = X * W + b.$$
- Aktivációs réteg:
$$H = g(X).$$

Egy neuronhálózat tanítása számolásigényes feladat, a tanítás akár több hét is lehet, az alkalmazás viszont gyors.

Tanítási módszerek:

- hiba-visszaterjesztés
  - az elvárt és kapott kimenet különbségénél megbecsüljük az egyes súlyok hozzájárulását a hibához
    - iránymenti deriváltak meghatározása, gradiens irányban elmozdulás
  - az egyes súlyokat korrigáljuk ennek megfelelően (se túl kicsit, se túl sokat)
  - zajos adatokkal nehezen boldogul
- túltanulás elleni védekezés
  - ritkítjuk a súlymátrixot (több 0 érték)
  - neuronok egy halmazát kirejti a hálózatból

Mélytanulás



2.3. ábra. A mélytanulás és a mesterséges intelligencia kapcsolata

A mélytanulást a következők teszik szükségessé:

- Bonyolultabb feladatoknál a hagyományos módszerek rosszul teljesítenek (a jellemzők kinyerése nehéz munka).
- Amennyiben az input nagyobb méretű, a "keskeny" neuronhálózatok nem elegendőek
  - a neuronok száma exponenciális növelendő
  - a pontosság csapnivaló, a tanulás lassú

Irodalomjegyzék

[1] **Russell** Stuart J, **Norvig** Peter: Mesterséges Intelligencia modern megközelítésben. Panem Kft., 2005.

[2] Várterész Magda: A mesterséges intelligencia alapjai: Az előadások mellé vetített anyag (2011).