Budapesti Corvinus Egyetem

Gazdálkodástudományi Kar

Számítástudományi Tanszék

Klaszterezés SOM algoritmus segítségével

Készítette: Fellner Anna Sára

Gazdaságinformatikus Szak

Budapest, 2019

Szakszeminárium vezető: Mohácsi László

Tartalomjegyzék

[1. Bevezetés 2](#_Toc482060810)

[2. Elméleti áttekintés 3](#_Toc482060811)

[2.1. Szövegbányászat 3](#_Toc482060812)

[2.1.1. A szövegbányászat célja 3](#_Toc482060813)

[2.1.2. Szövegbányászati eljárások 5](#_Toc482060814)

[2.2. Neurális hálózatok 6](#_Toc482060815)

[2.2.1. Biológiai neurális hálózatok 6](#_Toc482060816)

[2.2.2. Mesterséges neurális hálózatok 9](#_Toc482060817)

[2.3. Az önszerveződő térképek 12](#_Toc482060818)

[2.3.1. Az önszerveződő térképek elvi működése 12](#_Toc482060819)

[2.3.2. Az önszerveződő térképek sajátos tulajdonságai 13](#_Toc482060820)

[2.3.3. Az önszerveződő térképek absztrakt matematikai definíciója 13](#_Toc482060821)

[2.3.4. Felhasználási területek 14](#_Toc482060822)

[3. Gyakorlati probléma megoldása 15](#_Toc482060823)

[3.1. A program által megoldandó probléma leírása 15](#_Toc482060824)

[3.2. Nyelv és fejlesztői környezet megválasztása 15](#_Toc482060825)

[3.3. A program általános felépítése 16](#_Toc482060826)

[3.4. Színek klaszterezése 16](#_Toc482060827)

[3.4.1. Az algoritmus függvényei és paraméterezési lehetőségei 16](#_Toc482060828)

[3.4.2. Az implementált SOM algoritmus végrehajtása 22](#_Toc482060829)

[3.4.3. Eredmények összehasonlítása különböző paraméterezés esetén 24](#_Toc482060830)

[3.4. Szavak klaszterezése 28](#_Toc482060831)

[4. Összegzés 29](#_Toc482060832)

[5. Irodalomjegyzék 30](#_Toc482060833)

# 1. Bevezetés

Jelen szakdolgozat célkitűzése a neurális hálók családjába tartozó Self-Organizing Map nevezetű algoritmus implementálása, valamint különböző típusú bemeneti adatokkal való tesztelése volt. Az önszervező térképek tanulási algoritmusa néhány szempontból meglehetősen sajátos. A széles körben elterjedt neurális hálózatok többsége általában több rétegbe szervezett, eltérő funkcionalitással bíró neuronokból épül fel. A SOM ezzel szemben akár egyetlen neuron réteggel is képes különféle klaszterezési feladatok elvégzésre, illetve magas dimenziószámú bemeneti értékek látványos megjelenítésére a két- vagy háromdimenziós térben.

Az algoritmus nem felügyelt tanulást alkalmaz, részben emiatt nincs explicit eredménye, ezért a validálása rendkívül nehéz. Ráadásul a paraméterek (szomszédsági függvény, neuronok száma, iterációk száma, stb.) lehetséges értékeinek sokféle kombinációja tovább nehezítheti az optimális kimenet elérését. Eredetileg egy meglehetősen ambiciózus feladat megoldására, természetes szövegben előforduló szavak klaszterezésre szerettem volna felhasználni. Mivel ebben az esetben a hasonlóságot a szövegkörnyezet határozza meg, nincs triviális metrikai kapcsolat a szimbólumok között, ami rendkívül megnehezíti a bemeneti adatok megfelelő transzformálását.

Ezen okokból szükségesnek tűnt az algoritmust egyszerűbben ellenőrizhető és látványosabb kimenetet generáló problémák megoldásán is tesztelni. Két klaszterezési probléma megoldását valósítottam meg eredményesen. Az egyik a színek RGB kód alapján való csoportosítása, a másik pedig a klasszikusnak számító Írisz-adathalmaz klaszterezése. A feladat megoldása során egy általános célú programozási nyelvet, a .Net keretrendszerhez tartozó C#-ot használtam. Habár több, főleg matematikai problémák megoldására alkalmazott nyelvhez is elérhető SOM algoritmushoz készített library, érdekesebbnek tartottam az algoritmust az alapoktól felépíteni, valamint azt is vizsgálni szerettem volna, hogy személyi számítógépen milyen teljesítményre képes.

# 2. Elméleti áttekintés

## 2.1. Szövegbányászat

### 2.1.1. A szövegbányászat célja

Az a**datok** jellegük szerint háromféle kategóriába sorolhatóak: **strukturált**, **gyengén strukturált** és **strukturálatlan**. Strukturált adatok alatt általában az adatbázisokban tárolt adatokat értjük, ebben az esetben a tárolásra szolgáló adatstruktúra információt szolgáltat az adat szemantikájára vonatkozóan. Szabad formátumú szöveges dokumentumok (pl. e-mailek, tudományos publikációk) esetén az adatstruktúra nem utal az adatok szemantikájára, ezért ezeket strukturálatlan adatoknak nevezzük. Gyengén strukturált adatra az XML állományok szolgálhatnak például, mivel ilyenkor bizonyos szemantikus vagy szerkezeti információk is rendelkezésre állnak. (Tikk, 2007)

Az **adatbányászat** egy olyan folyamat, mely során az adatbázisokban tárolt nagy mennyiségű adat vizsgálata, modellezése után olyan implicit és rejtett információk, összefüggések, mintázatok és szabályszerűségek nyerhetőek ki, amelyek a gyakorlatban is jól hasznosíthatók. (Fajszi – Cser – Fehér, 2010) Ehhez a módszerhez értelemszerűen strukturált adatokra van szükség.

A gyengén strukturált és strukturálatlan adatok elemzésére külön szakterület alakult ki. **Szövegbányászat** alatt olyan szöveges adatokon végzett feldolgozási és elemzési tevékenységet értünk, melynek célja a dokumentumokban rejtetten meglévő új összefüggések feltárása, azonosítása és elemzése. (Tikk, 2007) Az 1. számú táblázat foglalja össze az adatbányászat és a szövegbányászat legfontosabb jellemzőit.

1. táblázat. Az adatbányászat és a szövegbányászat összehasonlítása (Tikk, 2007, 21. old. alapján)

|  |  |  |
| --- | --- | --- |
|  | **Adatbányászat** | **Szövegbányászat** |
| **elemzés tárgya** | numerikus és kategorikus | szabad formátumú szöveges dokumentum |
| **adatok jellege** | strukturált | strukturálatlan, gyengén strukturált |
| **adatok tárolási helye** | (relációs) adatbázis | dokumentumgyűjtemény |
| **feladat** | összefüggések feltárása, jövőbeni szituációk előrejelzése | szövegelemzés, információkinyerés, osztályozás, csoportosítás, összegzéskészítés, vizualizálás, kereséstámogatás |
| **jellemző módszerek** | neurális hálózatok, döntési fák, statisztikai modellek, klaszteranalízis, idősorok elemzése | dokumentumindexelés, felügyelt és felügyelet nélküli gépi tanulók, számítógépes nyelvészeti eszközök, ontológiák |

Az első lépés a szövegbányászat folyamata során a **dokumentumok** **előfeldolgozása**. Ennek célja, hogy a szöveges adatokat olyan formára hozzuk, amelyen a későbbiekben hatékonyan alkalmazhatjuk a tényleges **szövegbányászati eljárásokat**. Ez a forma legtöbbször valamilyen **numerikus reprezentáció**, de lehetséges adatbázis alapú vagy strukturált szöveges formátum is. Az előfeldolgozásnál általában szükség van a **nyelvtechnológia** eszközeire is, ilyen például a szótövezés vagy a stop szavak szűrése. A szövegbányászati eljárások végrehajtása után kapott eredményeket érdemes **információkezelő rendszerben** eltárolni a későbbi hatékony **tudás** kinyerés végett. (Fajszi et. al., 2010)

Untitled Diagram.png

1. ábra. A szövegbányászat általános modellje (Fajszi et. al., 2010, 270. old. alapján)

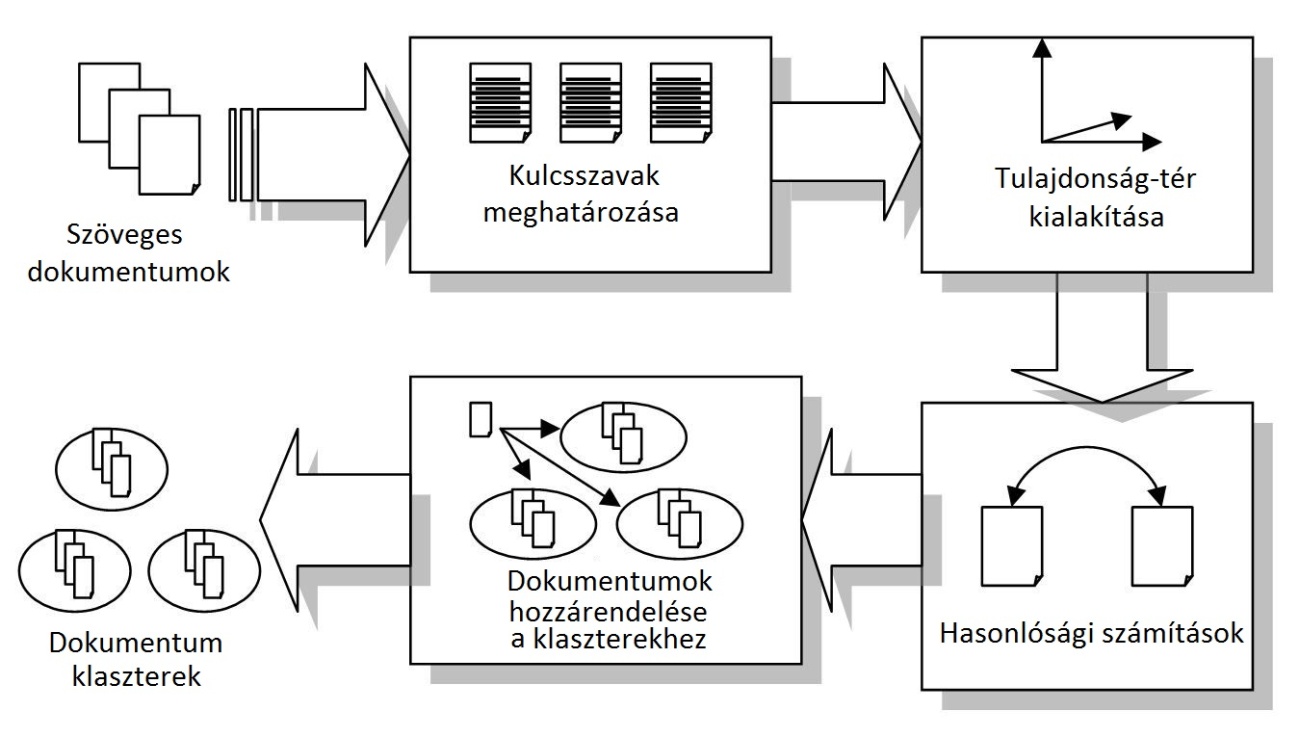
### 2.1.2. Szövegbányászati eljárások

Az **információkinyerés** célja egy előre definiált feladat szempontjából releváns tények kigyűjtése szöveges dokumentumokból, azaz strukturálatlan szövegekből strukturált információ előállítása. Az eljárás néhány felhasználási területe: orvosi zárójelentések, publikációk feldolgozása gyógyszerkutatási célból; sajtóban megjelenő hírek elemzése sajtófigyelő szolgáltatások esetén; katonai hírszerzési dokumentumokból rendszerezett információgyűjtés analitikusok számára. (Fajszi et. al., 2010) A közösségi média tartalmak ilyenfajta elemzése az elmúlt néhány évben került előtérbe, amely a rövid és „zajos” szövegek miatt különösen nagy kihívásokat tartogat. (Piskorski, – Yangarber, 2013)

A **szövegosztályozás** célja szöveges dokumentumok ellátása előre meghatározott tematikus kategóriacímkékkel. Ezáltal lehetővé válik a dokumentumok tartalom szerinti automatikus rendszerezése, amely az egyik legalapvetőbb szövegbányászati feladat. Az osztályozó algoritmus tanítódokumentumok alapján készíti el az egyes kategóriacímkék modelljeit, és ezek segítségével próbálja az ismeretlen dokumentumok kategóriáját meghatározni, azaz *felügyelt gépi tanulást* alkalmaz. (Fajszi et. al., 2010) Az eljárást számos területen alkalmazzák: szabadalmak kategorizálása, kéretlen levelek szűrése, dokumentumok nyelvének meghatározása, többértelmű szavak egyértelműsítése. (Tikk, 2007)

A **csoportosítás** (vagy más néven **klaszterezés**) a dokumentumok rendszerezésének egy másik módja. Ebben az esetben a cél elkülönülő dokumentum csoportok kialakítása, oly módon, hogy az egy csoportba tartozó dokumentumok minél hasonlóbbak, az eltérő csoportban lévők pedig minél különbözőbbek legyenek. Ilyenkor nem áll rendelkezésre tanítókörnyezet, így a csoportok a dokumentumok jellemzői alapján alakíthatók ki *nem felügyelt gépi tanulás* segítségével. (Tikk, 2007)

A klaszterezés használható a szövegosztályozási eredményének javításhoz, szövegek automatikus kivonatolásának előkészítéséhez, internetes keresőmotorok hatékonyságának növeléséhez. (Yuan-Chao Liu – Ming Liu – Xiao-Long Wang, 2012)



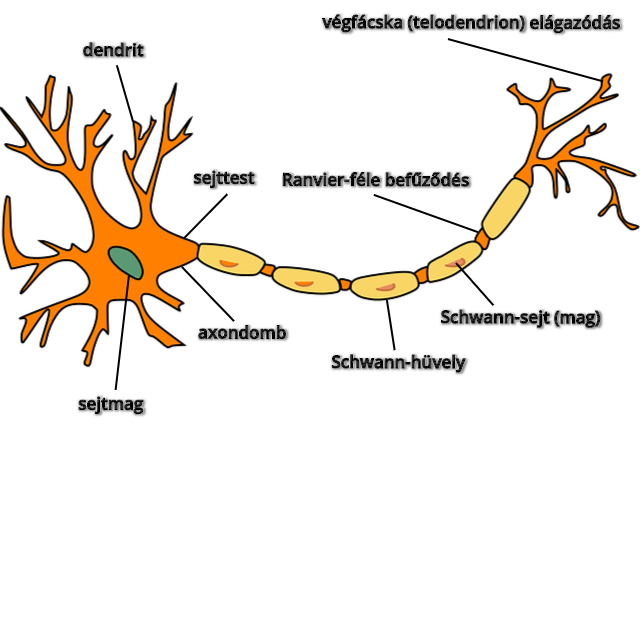
2. ábra. A dokumentum-klaszterezés általános modellje (Yuan-Chao Liu et. al., 2012 alapján)

A **kivonatolás** célja a dokumentumok tartalmának automatikus összefoglalása oly módon, hogy az eredeti szöveg leginkább jellemző részeit tartalmazza. Az egyik jellemző működési séma mondatkiválasztáson alapul, amelynek során a mondatokhoz fontosságuk alapján pontértéket rendelünk, és a legmagasabb pontszámú mondatok kerülnek bele a kivonatba. A kivonatolás technikája felhasználható internetes keresés és többnyelvű információkinyerés támogatására, összehasonlító táblázatok készítésére, illetve biográfiai profilok készítésére. (Tikk, 2007)

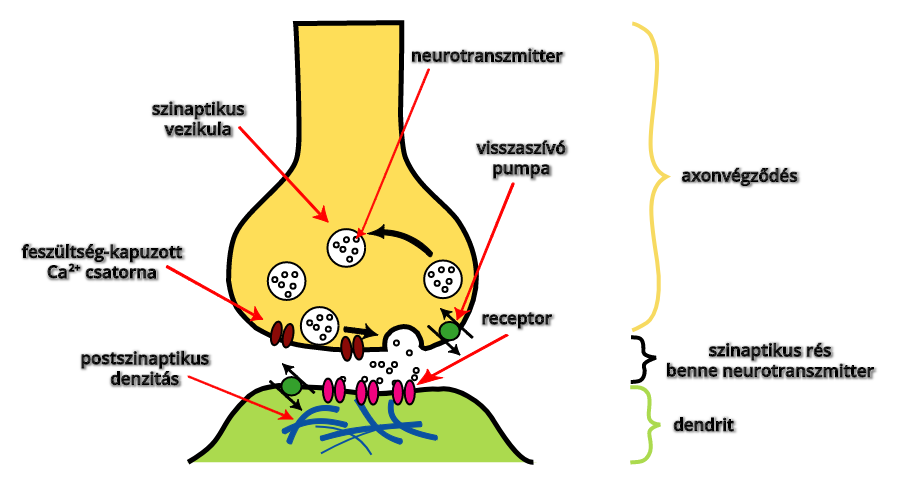
## 2.2. Neurális hálózatok

### 2.2.1. Biológiai neurális hálózatok

A **biológiai** (vagy más néven természetes) **neurális hálózatokat** egymással összeköttetésben lévő idegsejtek alkotják. Az **idegsejtek** nyúlványokkal kapcsolódnak egymáshoz, melyeknek két típusa van. A **dendritek** az ingerületek felvételéért felelősek, melyet a sejttesthez továbbítanak. Az **axon** feladata a sejttesttől származó ingerületek továbbítása.



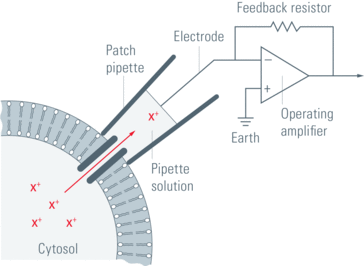
3. ábra. Az idegsejt felépítése (Wikipédia – Idegsejt szócikk alapján)



4. ábra. A szinapszis struktúrája (Wikipédia – Szinapszis szócikk alapján)

Az idegsejtek között a kapcsolat **szinapszisokon** keresztül valósul meg, ahol az ingerület átvitel kémiai jellegű. A 4. ábrán látható gömbök az úgynevezett **vezikulumok**, ezekben foglalnak helyet a **neurotranszmitterek**. Amikor az idegsejt tüzel, akkor a neurotranszmitterek egy része beleürül a **szinaptikus résbe**, mely a két idegsejt között található. A neurotranszmitterek kötődhetnek az ingerelt idegsejt sejtfalába épülő makromolekulákhoz. A bekötött neurotranszmitterek hatására a makromolekulák térbeli szerkezete megváltozik, az **ioncsatornák** kinyílnak vagy összezáródnak, a sejtfal átjárhatósága Na+ illetve Ca+ számára megváltozik. A sejtbe áramló Na+ és Ca+ ionok megváltoztatják a sejt membránpotenciálját. Ha a dendritek felől érkező potenciálváltozás elér egy szintet, a sejt tüzel, azaz egy elektromos impulzus fut végig az axonon, melynek hatására a neurotranszmitterek a szinaptikus résbe ürülnek. A szinaptikus résbe ürült neurotranszmittereknek vissza is kell szívódnia, visszaürülnie a vezikulumokba, különben az idegsejt nem lenne képes újra tüzelni. Ennek a kémiai folyamatnak van egy meghatározott ideje, amely limitálja az idegsejtek működésének maximális sebességét. Egy idegsejtnek több ezer dendritvégződése is lehet, így nagyon sok más idegsejttel lehet kapcsolatban. A szinaptikus résbe beépülhetnek olyan molekulák, amelyek a szinaptikus ingerület átvitelt erősítik vagy gátolják.

Az idegsejtek működésének vizsgálata kétféleképpen történhet: **in vivo** és **in vitro**. In vivo: ebben az esetben a vizsgálat magában az élő szervezetben történik. Az egyik legelterjedtebb technológia erre az úgynevezett **patch clamp** módszer. A vizsgálat során egy elektrolittal feltöltött mikropipettát tapasztanak az idegsejt membránjára. A sejt ioncsatornáján keresztül mérhetővé válik annak tüzelése, a sejtmembránon mért feszültségváltozás erősítés után hangszórón meg is hallgatható, ekkor minden tüzelés egy-egy kattogó hangnak felel meg. Ebből következtetni lehet arra, hogy az élő szervezetben milyen gyakran következnek be a tüzelések egy idegsejt esetében. (Veitinger, 2011) Érdekes, hogy a sejtek ingerlés hiányában spontán is tüzelnek időnként. In vitro: ebben az esetben egy vagy több sejtből álló sejtpreparátumot készítenek üveglapra, melyet tápoldaton tartanak életben. Ekkor mérhetővé válnak a sejtek közötti szinaptikus kapcsolatok, az elektrokémiai impulzus terjedése a sejt különböző részeiben, a jelterjedés paraméterei közti összefüggések.



5. ábra. A patch clamp módszer (Veitinger, 2011)

### 2.2.2. Mesterséges neurális hálózatok

A **mesterséges neurális hálózatok** koncepciójának megalkotását ez a biológiai rendszer ihlette, mivel a természetes neurális hálózatok képesek olyan bonyolult feladatok megoldására, melyre szinte lehetetlen hatékonyan működő hagyományos algoritmust kidolgozni. Ilyen feladat például az alak- vagy a beszédfelismerés.

Felmerül azonban a kérdés, hogy az agyműködés szimulálása egyáltalán lehetséges-e Neumann-elvű számítógépeken. Az **emberi agy** nagyjából **100 milliárd idegsejt**ből áll. A neocortex esetében – mely szorosan kapcsolódik a magasabb szintű kognitív folyamatokhoz – az idegsejtek száma nőknél 19 milliárd, férfiaknál 23 milliárd. (Platek – Keenan – Shackelford, 2007, 139. old.) Egy idegsejt átlagosan **7000 szinaptikus kapcsolat**tal rendelkezik. (Drachman, 2005) Az idegsejtek a tüzelési sebessége korlátozott, viszont időben **párhuzamos**an működnek. Habár a mai processzormagok már gigahertzes nagyságrendű órajellel is rendelkezhetnek, továbbra is csak **soros**an képesek az utasításokat elvégezni. A többmagos processzorok (valamint az olyan speciális technológiák, mint az NVIDIA által fejlesztett CUDA) sem képesek megközelíteni azt a szintű párhuzamosítást, amelyre az emberi agy képes. A mesterséges neurális hálózatok tehát messze vannak attól, hogy pontosan modellezzék az emberi agy működését, mégis sikeresen alkalmazzák annak bizonyos aspektusait. A mesterséges neuronok közötti **súlyok** például az idegsejtek között létrejövő **szinapszisok**at próbálják modellezni.

A mesterséges neurális hálózatok egyszerű számítási műveleteket használnak bonyolult problémák megoldására. Tipikusan ilyenek a rosszul definiált matematikai, nemlineáris, sztochasztikus problémák. A hagyományos algoritmusok bonyolult egyenletek halmazát használják célzottan egy, és csakis egy feladat megoldására, ezzel szemben a neurális hálózati megoldások (bizonyos keretek között) könnyen adaptálhatóak különböző feladatok megoldására. (Graupe, 2007)

A neurális hálózatok működése tipikusan két fázisra osztható:

* a **tanulási fázis**ban a hálózat kialakítása történik meg a rendelkezésre álló adatok alapján,
* az **előhívási fázis**ban pedig a hálózat az adatokból kinyert információ alapján képessé válik a tőle elvárt leképezés megvalósítására. (Altrichter - Horváth - Pataki - Strausz - Takács - Valyon, 2006)

#### A neuron

A mesterséges neurális hálózatok mesterséges neuronokból épülnek fel, melyek szoros összeköttetésben állnak egymással. A mesterséges **neuron**ok a biológia neuronokhoz hasonlóan egy vagy több bemenettel, és egy kimenettel rendelkeznek. A bemenetekből a neuron egy nemlineáris függvény alkalmazásával hozza létre a kimenetet. Ezt a függvényt szokás **aktivációs** vagy **transzfer** **függvény**nek nevezni. A neuronok rendelkezhetnek lokális memóriával is, ebben az esetben a bemenetek mellett a memóriában tárolt érték is hatással lehet a kimenetre.

C:\Users\Sára\Desktop\Untitled Diagram.png

6. ábra. A neuron általános felépítése

#### Topológia

A neuronok összeköttetési rendszere, valamint a bemenetek és kimenetek helye határozza meg az adott neurális hálózat topológiáját. A neuronok általában nincsenek egyszerre kapcsolatban az összes többi neuronnal, csupán azok egy részhalmazával. Példaként: a rétegekbe szervezett neuronok egyik lehetséges csoportosítási módja topológia szempontból:

* **bemeneti neuronok**: egy bemenetű, egy kimenetű, jelfeldolgozást nem végző neuronok, bemenetük a hálózat bemenete;
* **kimeneti neuronok**: kimenetük a hálózat kimenete, azaz a környezet felé továbbítják a kívánt információt;
* **rejtett neuronok**: bemeneteik és kimeneteik kizárólag más neuronokhoz kapcsolódnak.

A fentiek alapján a neuronok **réteg**ekbe szervezhetőek. A rétegekbe szervezett hálónak rendelkeznie kell legalább egy bemeneti és egy kimeneti réteggel, melyek között tetszőleges számú rejtett réteg lehet. (Horváth et. al., 2006)

#### Tanulási módszerek

A neurális hálózatok csoportosíthatóak az általuk alkalmazott tanulási algoritmus alapján. Alapvetően kétféle módszert különböztethető meg.

* **Felügyelt tanulás** esetén rendelkezésre állnak összetartozó bemeneti és kimeneti adatok. Ezek felhasználásával a hálózat meghatározza azt a függvényt, amely képes a minta adatok által képviselt leképezés létrehozására.
* **Nem felügyelt tanulás** esetén a bemeneti adatokhoz nincsenek ismert, hozzájuk tartozó kimeneti adatok. Ebben az esetben a hálózatnak a bemeneti értékek alapján kell azok hasonlóságát, korrelációját meghatároznia, illetve azt, hogy hogyan alakíthatóak ki belőlük kategóriák, csoportok. Így tehát az elvárt kimenet, a háló által megvalósítandó leképezés nem definiálható előre. (Horváth et. al., 2006)

A **kompetitív tanulás** a nem felügyelt tanulási módszerek közé sorolható, ekkor minden bemeneti értékhez egy győztes neuron kerül kiválasztásra. Ilyenkor csak a győztes neuron (vagy a győztes neuron környezetében található neuronok egy csoportja) fog adaptálódni az adott kimenethez. Ez a tanulási elv elsősorban a klaszterek kialakítása során használatos.

#### Felhasználási területek

A különböző tanulási stratégiákhoz jellemzően más felhasználási területek kapcsolódnak. A **felügyelt tanítású** hálózatok fontosabb alkalmazási területei:

* egy- és többdimenziós jelfeldolgozás (képfeldolgozás, beszédfeldolgozás, stb.)
* felismerési problémák: képek, karakterek, szövegek felismerése, beszéd, beszélő felismerése
* rendszer-identifikáció és vezérlés,
* robotika,
* orvosi és műszaki diagnosztika,
* pénzügyi és közgazdasági folyamatok modellezése, jellemzők becslése. (Horváth et. al., 2006, 207. old.)

**Nem felügyelt tanítás** esetén ezek a főbb felhasználási területek:

* a bemenetre kerülő minták közötti hasonlóság megállapítása,
* a bemeneti mintatérben csoportok, klaszterek kialakítása,
* adattömörítés, főkomponens analízis,
* független komponensek meghatározása. (Horváth et. al., 2006, 293. old.)

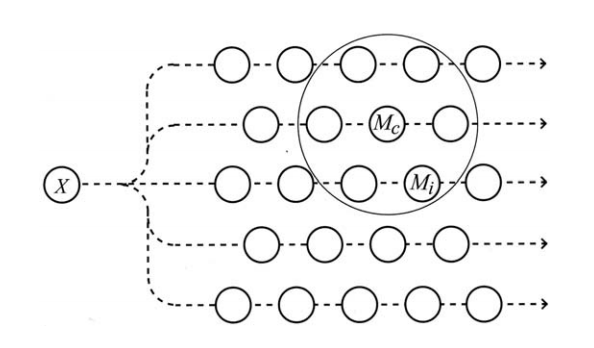
## 2.3. Az önszerveződő térképek

Az **önszerveződő térkép** (angolul **Self-Organizing Map**, röviden **SOM**) egy olyan adatelemzési módszer, melynek célja, hogy a magas dimenziószámú adathalmazok elemei között fennálló hasonlóságot vizualizálja. (Kohonen, 2014) Az algoritmus az 1980-as években került kidolgozásra Teuvo Kohonen által, így Kohonen-térkép néven is szokás hivatkozni rá.

Az önszerveződő térkép felfogható egy **mesterséges neurális hálózat**ként. A hálózatot alkotó neuronok egy **rácson** helyezkednek el, ez a rács általában két- vagy háromdimenziós. Topológiáját tekintve a hálózat **egyetlen neuron réteg**ből áll. Minden bemenet kapcsolódik a hálózat összes neuronjához. Mivel nincs külön kimeneti réteg, a hálózat minden egyes neuronja egyben kimeneti neuron is. (Horváth et. al., 2006) A háló tanítására használt algoritmus a **nem felügyelt tanulás**i módszerek közé tartozik, valamint érvényesül a **kompetitív tanulás** elve is.

### 2.3.1. Az önszerveződő térképek elvi működése

Az önszerveződő térkép alapvető működése a 4. ábra alapján a következőképpen írható le: ***X*** egy **bemeneti érték**, amely minden ***Mi*****modell**nek (neuronnak) bemenetként szolgál. Válasszunk ki egy ***Mc***modellt úgy, hogy az a lehető legjobban hasonlítson az *X* értékére, ez lesz a **győztes modell**. Az összes *Mc* adott környezetében található modellt módosítsuk úgy, hogy jobban hasonlítsanak az *X* értékre, mint a többi modell. Ahogy egyre több bemeneti érték kerül ily módon feldolgozásra, a modellek egyes részhalmazai úgy fognak egyre inkább adaptálódni az egyes bemeneti értékekhez. (Kohonen, 2014)



7. ábra. Az önszerveződő térkép elvi működésének modellje (Kohonen, 2014, 15. old)

### 2.3.2. Az önszerveződő térképek sajátos tulajdonságai

Az önszerveződő térképek hasonlítanak a **nemlineáris leképezést** megvalósító módszerekhez (ilyen a többdimenziós skálázás vagy a Sammon-leképezés), melyek lényege, hogy magas dimenziószámú bemeneti vektorokat képeznek le egy kétdimenziós euklideszi síkban úgy, hogy a leképezés megőrzi a magas dimenziószámú térben lévő vektorok egymástól való távolságát.

A SOM algoritmus azonban a leképezéshez nem euklideszi síkot használ, hanem **csomópontok** **hálózatát**. Ezen kívül a **bemeneti értékeket** nem közvetlenül képezi le, hanem modellek (neuronok) segítségével a **lokális átlagukat** reprezentálja. Az önszerveződő térképek **súlyvektorokat** használnak a különböző bemenetek összehasonlításához és a modellek kialakításához. A **modellek pozíciója** a rácsban **sosem változik**, ehelyett a hozzájuk tartozó súlyvektorokat módosítjuk a bemeneti értékeknek megfelelően. (Kohonen, 2014)

### 2.3.3. Az önszerveződő térképek absztrakt matematikai definíciója

Legyen a bemeneti adat {**x**(*t*)} n dimenziós valós euklideszi vektorok sorozata, ahol a *t* egész szám határozza meg a sorozat indexét. Legyen az {**m**i(*t*)} n dimenziós valós vektorok sorozata az **m**i modellek egymás után számított becslése, ahol *i* az **m**i modellhez tartozó csomópont térbeli indexe. A SOM algoritmus alapfeltevése, hogy a következő tanulási szabály követésével az eredmények konvergensek lesznek, és létrehozzák a kívánt csoportosítást a modellek által:

,(1)

ahol *h*ci(*t*) az úgynevezett szomszédsági függvény, *c* pedig a nyertes neuron térbeli indexe. A nyertes **m**c(*t*) modell a lehető legkisebb euklideszi távolságra helyezkedik el az x(*t*) bemeneti értéktől:

. (2)

Vagyis először az **x**(*t*) bemeneti értékhez kiválasztjuk a nyertes neuront a (2) egyenlet alapján, majd az (1) egyenlet alapján módosítjuk a nyertes neuron és az ahhoz térbeli indexük alapján közel álló neuronok vektorát, oly módon, hogy az jobban hasonlítson a bemeneti vektorhoz. (Kohonen, 2014, 21. old.)

A szomszédsági függvény megválasztása központi szerepet játszik az algoritmus önszerveződési képességét illetően. Mivel a függvénynek többféle változata és paraméterezése is lehetséges, ezért ennek a függvénynek a részletesebb kifejtésére a gyakorlati probléma megoldásánál kerül sor.

### 2.3.4. Felhasználási területek

A SOM algoritmust rendkívül sokféle területen alkalmazzák:

* feltáró adatelemzés,
* szövegek statisztikai analízise és rendezése, (Kohonen, 2014)
* ipari elemzések, (Kohonen – Oja – Simula – Visa –Kangas, 1996),
* telekommunikáció, (Bella – Eloff – Olivier, 2009),
* ügyfélkör elemzés, (Tulankar – Kshirsagar – Wajgi, 2012),
* orvosbiológiai elemzések és alkalmazások, (Xu – Wong, Chin, 2014),
* pénzügyi alkalmazások, (Deboeck – Kohonen, 1998),
* bűnügyi profilalkotás,
* galaxisok kategorizálása, (Miller – Coe, 1995)
* dokumentumok rendszerezése, (Chandrashekar – Shoba, 2009)
* beszédfelismerés.

# 3. Gyakorlati probléma megoldása

## 3.1. A program által megoldandó probléma leírása

Célom egy olyan program létrehozása volt, mely **implementálja a SOM algoritmust**, amit három különböző probléma megoldásán teszteltem. Mindhárom feladat esetében a többdimenziós bemeneti adatokat egy kétdimenziós rácson szerettem volna csoportosítva vizualizálni, amely lényegében **klaszterezés** segítségével valósul meg az algoritmus által. A szóklaszterezési probléma esetében a hasonlóság alapja a **szavak kontextusa**, azaz hogy egy adott szó mely más szavak közelében található meg.

Az implementálás során kihívást jelentett, hogy a SOM algoritmusnak nincsen explicit eredménye, ezért a kimenetet nem lehet pontosan validálni, továbbá a feladat szemantikus volta miatt sokkal bonyolultabb megfelelő bemenetet generálni.

A fentiek miatt úgy döntöttem, hogy alapvetően metrikus bemeneti adatokra is alkalmazom az algoritmust, melyek látványos kimenetet generálnak. Az egyik probléma a **színek klaszterezése**, ahol az **RGB kódok** miatt adott a metrikus bemenet. így sokkal könnyebben alkalmazható rá az önszerveződő térképek készítése. Ebben az esetben a 3 komponensű vektorokkal reprezentált színek képződnek le a kétdimenziós térben, mégpedig oly módon, hogy az egymáshoz hasonlító színek egymás mellé kerülnek. Ezen kívül a klasszikus **Írisz-adathalmaz** három különböző faját képeztem le négy fizikai jellemzőjük alapján.

Az **önszerveződő szemantikus térkép** létrehozásához szükséges algoritmus implementálása sok kihívással járt. Az általános SOM algoritmus megvalósításához szükséges matematikai ismereteken felül az elemezni kívánt szöveges adatsor speciális jellege miatt (a számoknál a távolság fogalma szinte magától értetődő, míg ugyanez a szavakra már nem igaz) olyan problémák is megoldásra vártak, mint például a számításokhoz használt vektorok dimenziószámának csökkentése, vagy az input adatként használt „nyers” szöveg előfeldolgozása.

## 3.2. Nyelv és fejlesztői környezet megválasztása

A fejlesztésre számos nyelv és fejlesztői környezet alkalmas lehet, azonban ezek közül kiemelkedik a MATLAB a hozzá elérhető SOM Toolbox függvénykönyvtár miatt, amely nevéhez híven számos beépített függvénnyel támogatja az algoritmus megvalósítását. Végül egy általános célú programozási nyelvmellett döntöttem, mivel egyrészt érdekelt az algoritmus személyi számítógépeken elérhető teljesítménye, másrészt pedig az alapoktól szerettem volna azt felépíteni a mélyebb megértés érdekében. A fejlesztői környezet és a programozási nyelv kiválasztásakor a korábban ezekkel az eszközökkel szerzett gyakorlatom miatt a felhasználóbarát **Visual Studiora** és a **C# programozási nyelvre** esett a választásom.

## 3.3. A program általános felépítése

A program összesen két fontos **felhasználói felületből** áll. Az egyik a színek klaszterezésének paraméterezésére és megjelenítésére szolgál, míg a másik ugyanezt valósítja meg az íriszekkel.

A feladat megoldása során próbáltam objektum orientált megközelítésben létrehozni az osztály struktúrát. Az algoritmus matematikai leírásából következik, hogy a be- és kimeneti adatok a legegyszerűbb módon vektorokkal reprezentálhatóak. Mivel a C#-ban nincs beépített N-dimenziós vektor osztály, ezért ehhez egy külső **Math.NET Numerics** nevű libraryt használtam, amely nagyban megkönnyítette az algoritmus implementálását a számos beépített vektor metódus által.

A **bemeneti adatok** kezelésére két absztrakt osztályt hoztam létre. Az **InputDataItem** osztály egy-egy input adatot reprezentál, míg az **InputDataSet** a bemeneti inputok egész halmazát tartalmazza. Az **InputDataItem** osztály egyetlen publikus InputVector propertyvel és az ahhoz tartozó getter/setter metódusokkal rendelkezik. Ezt öröklik a leszármazott **ColorInputDataItem** és a **FlowerInputDataItem** osztályok, melyek a két konkrét feladathoz tartozó megvalósítások. Az **InputDataSet** absztrakt osztály az input vektorok listáján kívül rendelkezik még két hasznos propertyvel (az inputok számával és a vektorok dimenziószámával), valamint SelectRandomInputItem() metódussal, amely egy véletlenszerű inputot ad vissza. Az **InputDataSet** osztályt szintén két feladat specifikus osztály implementálja. Míg a **ColorInputDataSet** osztály nem tartalmaz további implementációt, a **FlowerInputDataSet** felelős azért, hogy az íriszek bemeneti adatait parszolja egy csv kiterjesztésű fájlból.

A bemeneti adatok kezelésére egy **input osztály**t hoztam létre, míg a kimeneti adatoknál mindkét problémához két önálló, de közös őstől származó **neuron osztály** került implementálásra. Ezek ugyan hasonlóak egymáshoz, de az egyszerű vizuális megjelenítés érdekében mégis az önálló osztályok használata mellett döntöttem. A szemantikus probléma megoldásánál az **előfeldolgozás**i lépésekhez több (az input osztályt is beleszámítva összesen 4) osztályt is létrehoztam, mivel úgy gondoltam, hogy ezek jelentősen növelik a kód átláthatóságát. Ezen kívül felhasználtam még egy úgynevezett  **függvény osztály**t is, amely néhány, a programban gyakran használt statikus függvényt tartalmaz.

## 3.4. Színek klaszterezése

### 3.4.1. Az algoritmus függvényei és paraméterezési lehetőségei

#### Input adatok létrehozása

A SOM algoritmus indítása előtt az első lépés praktikusan a bemeneti adatok meghatározása, mivel ezen adatok **dimenziószám**a fogja meghatározni a neurális háló csomópontjainak dimenziószámát is. Színek csoportosítása esetén ez meglehetősen egyszerű feladat. Jelen esetben 8 darab színt használunk input adatként, melyek reprezentálásra egy-egy 3 dimenziós vektor szolgál, melyek komponensei az adott szín RGB kódjának komponenseivel lesznek egyenlők. Az input adatok tetszés szerint lehetnek egy előre meghatározott „szín-tömb” elemei, de akár véletlenszerűen is le lehet generálni őket a program által. Ezek egy vektorokat tartalmazó tömbben kerülnek eltárolásra, ahogy az az 1. kódrészletben is látszik:

public static Vector[] CreateColorInput(bool fixedcolor)

{

Vector[] Inputs = new Vector[Constants.inputNumber];

for (int i = 0; i < Constants.inputNumber; i++)

{

Inputs[i] = new Vector(Constants.vectorDimension);

if (fixedcolor == false)

{

RandomColors(Inputs, i);

}

}

if (fixedcolor == true)

{

Inputs[0].Components = new double[] { 128, 0, 128 };

Inputs[1].Components = new double[] { 186, 85, 211 };

Inputs[2].Components = new double[] { 148, 0, 211 };

Inputs[3].Components = new double[] { 255, 255, 0 };

Inputs[4].Components = new double[] { 255, 215, 0 };

Inputs[5].Components = new double[] { 128, 0, 0 };

Inputs[6].Components = new double[] { 178, 34, 34 };

Inputs[7].Components = new double[] { 255, 0, 0 };

}

return Inputs;

}

1. kódrészlet

#### A neurális háló tervezése

Második lépés a neurális háló megtervezése. Az egyik fő eldöntendő kérdés, hogy **mennyi neuronra** van szükség az elérni kívánt térkép megvalósításához. Ezt sok esetben szinte lehetetlen előre megjósolni, ezért ilyenkor próbálgatással lehet megtalálni az ideális méretet. Általánosságban elmondható azonban, hogy a várható klaszterek számával arányosan érdemes a neuronok számát megválasztani.

A másik kérdés a neuronok **elhelyezkedés**ének a megválasztása. A legelterjedtebb esetben a neuronok egy **kétdimenziós négyzetes vagy hexagonális rács**on foglalnak helyet egymástól egyenlő távolságra. A függőleges és a vízszintes **oldalak arányá**nak meghatározása szintén fontos lehet. A legjobb, ha ezek aránya nagyjából megegyezik a bemeneti adatok eloszlásának arányával az első két főkomponensük alapján. (Kohonen, 2014) Színek esetén a rács relatív egyszerűen kialakítható, ugyanis egy 40×40-es négyzet alakú rács megfelelő 8 szín megjelenítéséhez.

A 2. kódrészletben megfigyelhető, hogy a rács (angolul lattice) tulajdonképpen egy kétdimenziós neuron tömbből áll. Az, hogy a tömb kétdimenziós, jelentősen megkönnyíti a neuronok ábrázolását a kimeneten:

public ColorNeuron[,] CreateLattice(int x, int y, int dimension)

{

ColorNeuron[,] lattice = new ColorNeuron[x, y];

for (int i = 0; i < x; i++)

{

for (int j = 0; j < y; j++)

{

ColorNeuron n = new ColorNeuron(i, j, dimension);

lattice[i, j] = n;

outputPanel.Invoke(new MethodInvoker(delegate

{

outputPanel.Controls.Add(n);

}));

}

}

return lattice;

}

2. kódrészlet

A neuronok létrehozásakor inicializálni kell a hozzájuk tartozó n dimenziós vektort. Az **inicializálás** általában kétféleképpen történik: egyszerűbb esetben a vektor komponenseit **véletlen értékek**kel töltjük fel, így a kezdeti állapot is véletlenszerű lesz. Ennél egyre elterjedtebb azonban a **főkomponens-analízis alapú inicializálás** (Principal Component Initialization), amelynek segítségével az algoritmus konvergenciájának sebessége jelentősen felgyorsítható. (Akinduko – Mirkes – Gorban, 2015) Utóbbinak azonban meglehetősen bonyolult a matematikai megvalósítása, így a programban a véletlenszerű értékekkel való inicializálás mellett döntöttem, ahogy ez a 3. és 4. kódrészletben is megfigyelhető:

public ColorNeuron(int x, int y, int dimension)

{

Weights = new Vector(dimension);

Weights.Randomize();

\_x = x;

\_y = y;

this.Top = (ColorNeuron.size + ColorNeuron.spacing) \* y;

this.Left = (ColorNeuron.size + ColorNeuron.spacing) \* x;

this.Height = ColorNeuron.size;

this.Width = ColorNeuron.size;

this.BackColor = Color.Black;

}

3. kódrészlet

public void Randomize()

{

for (int i = 0; i < \_components.Length; i++)

{

\_components[i] = rnd.NextDouble();

}

}

4. kódrészlet

#### A szomszédsági függvény kiválasztása

A **szomszédsági függvény** megválasztása az algoritmus egyik legkritikusabb pontja. Ez határozza meg, hogy az egyes neuronok a győztes neurontól való távolságuk függvényében mennyire adaptálódnak az adott bemeneti értékhez. Ennek többféle változata is létezik, de közös jellemzőjük, hogy csak a győztes neuron esetében veszik fel maximális értéküket (mely egy 1-nél nem nagyobb pozitív szám), a győztes neurontól távolodva pedig értékük monoton csökken. (Horváth et. al., 2006, 299. old.)

A legegyszerűbb szomszédsági függvény az úgynevezett **buborék forma**. Ekkor *h*ci = 1, ha egy meghatározott távolságnál közelebb van a neuron a győztes neuronhoz, minden más esetben *h*ci = 0. (Kohonen, 2014)

Az egyik leggyakoribb választás a **normális eloszlású** (vagy más néven Gauss-eloszlású) **függvény**:

, (3)

ahol α(t) < 1 és σ(t) egy-egy monoton csökkenő skaláris függvénye t-nek. Ezekben az esetekben a monoton csökkenés lehet például hiperbolikus, exponenciális vagy szakaszos lineáris. (Kohonen, 2014)

A normális eloszlású függvény egyik módosított változata a **vágott Gauss-eloszlású függvény** (angolul **cut gaussian**), ennek képlete:

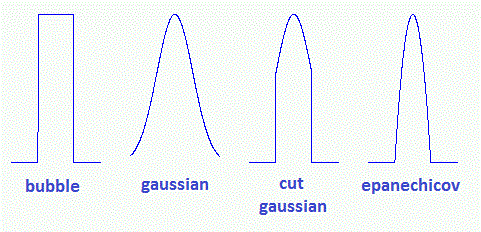
, (4)

(Barbarino – Boinee – De Angelis, 2008).

Ismeretes még az úgynevezett **epanechicov függvény**:

, (5)

(SOM implementation in SOM Toolbox, 2005).



8. ábra. A különböző szomszédossági függvények ábrázolása   
(SOM implementation in SOM Toolbox, 2005 alapján)

Az implementálás során egy normális eloszlású szomszédossági függvény használata mellett döntöttem, mivel a szemantikus önszerveződő térképekkel foglalkozó szakirodalom többségében is ezt alkalmazták. Az általam használt függvény a következő:

, (6)

(Ritter – Kohonen, 1989).

Az 5. kódrészletnél látható a képlet megvalósítása. A függvény két paramétere az aktuális két pont közötti euklideszi távolság (distance), valamint az aktuális szórás (standard deviation, a kódban SD).

public static double NeighborhoodFunction(double distance, double SD)

{

double dividend = Math.Pow(distance, 2);

double divisor = Math.Pow(SD, 2);

double NF = Math.Exp(-1 \* (dividend / divisor));

return NF;

}

5. kódrészlet

#### A szomszédossági függvény alakjának változása az iterációk függvényében

A SOM algoritmus esetében a különböző szomszédossági függvények közös jellemzője, hogy az iterációk számának növekedésével párhuzamosan az általuk lefedett neuronok csoportja is **csökken**. (Haykin, 1999) Ennek megfelelően a győztes neuron „hatósugara” folyamatosan csökken, így egyre kevesebb neuronra lesz hatással a tanulási függvény. Ezért a környezeti zsugorodásért a **σ(t) függvény** felel, mely meghatározza az effektív **szélesség**ét a topológiai környezetnek.

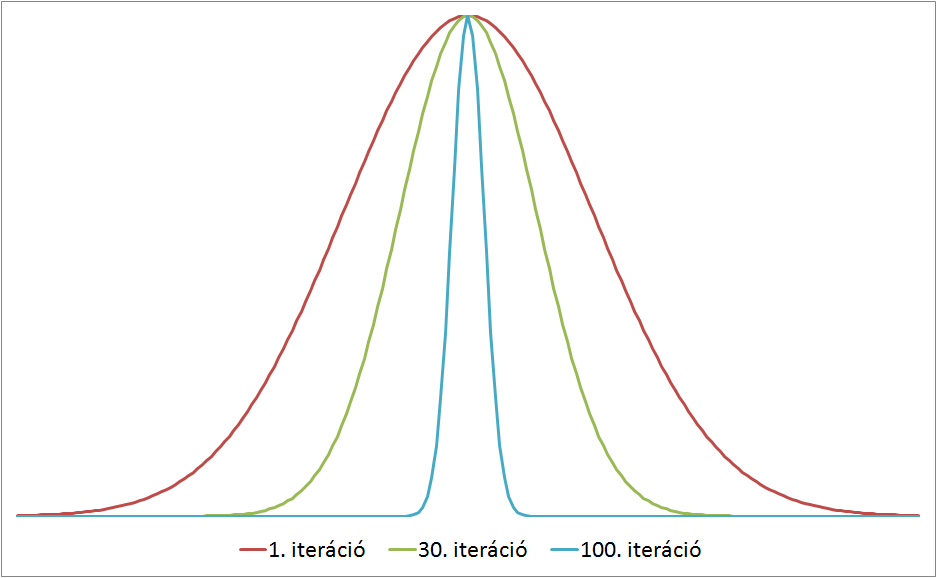
A 8. ábrán látható, hogyan változik a normális eloszlású függvény alakja az iterációk előrehaladtával. A függvényt a következő képlet felhasználásával állítottam elő:

. (7)

σ(t) képlete:

, (8)

(Ritter – Kohonen, 1989), ahol σi (**szórás kezdeti értéke**) és σf (**szórás végleges értéke**) konstansok, melyeknél a következő értékeket használtam: σi = 4 és σf = 0,5. t az iteráció aktuális, tmax pedig az iterációk maximális számát jelöli, jelen esetben ez 100 volt. Az ábrán jól látható, hogyan csökken a Gauss-görbe félértékszélessége az iterációk számának növekedésével.



9. ábra. A szomszédossági függvény alakja 1, 30 és 100 iteráció után.

Természetesen a σ(t) más alakot is felvehet, azzal a feltétellel, hogy monoton csökkenő skaláris függvénye legyen t-nek. A programomban a (8) számú függvényt valósítottam meg. A 6. kódrészlet alapján σi-t az SDI (standard deviation initial), míg σf-t az SDF (standard deviation final) változókkal jelöltem.

public static double StandardDeviation(double SDI, double SDF, int numberOfIterations, int iteration)

{

double exponent = (double)iteration / (double)numberOfIterations;

double base\_ = SDF / SDI;

double SD = SDI \* Math.Pow(base\_, exponent);

return SD;

}

6. kódrészlet

#### A tanulási együttható

A **tanulási együttható** adja meg, hogy a szomszédossági függvény által lefedett neuronok csoportja milyen mértékben adaptálódjon az adott bemeneti adathalmazhoz. Ennek az értékét adja meg az **α(t) függvény**, amely monoton csökkenő. Leggyakoribb formái a lineáris, eltelt idővel fordítottan arányos, és a hatványsoros függvények, ezek képletei:

* lineáris: (9)
* eltelt idővel fordítottan arányos: (10)
* hatványsoros: (11)

(Natita – Wiboonsak – Dusadee, 2016).

A programomban a (10) számú képletet alkalmaztam a tanulási együttható csökkentésére. A tanulási együttható angolul **Learning Coefficient**, innen az LC változó neve. (7. kódrészlet)

LC = LC \* (1 - (iteration / numberOfIterations));

7. kódrészlet

#### Az iterációk számának megválasztása

Az iterációk maximális számát nehéz előre megállapítani, próbálgatással azonban ki lehet tapasztalni. Ha a térkép egy bizonyos mennyiségű iteráció után már nem változik, az azt jelenti, hogy a neurális háló már „abbahagyta” a tanulást. Ekkor érdemes az iterációk számát csökkenteni, hiszen az iterációk száma jelentős hatással van az algoritmus lefutási idejére. 8 darab szín klaszterezéséhez néhány ezer iteráció is elegendőnek bizonyult.

#### A program által használt függvények és paraméterek

A SOM algoritmus egyik fő jellemzője, hogy számtalan módon lehet implementálni. Különböző függvények és paraméterek használata esetén drasztikus eltérések lehetnek a végeredmény használhatóságát illetően. A színek klaszterezése esetén bizonyos paramétereket **konstans**ként kezelek, így az input adatok száma minden esetben 8, ezzel összhangban a neurális háló neuronjainak a száma 160, melyek egy 40×40-es rácson helyezkednek el.

Összefoglalva tehát, a programban a következő függvényeket használtam fel:

**Tanulási függvény**:

, (12)

amely tulajdonképpen az (1) függvény kissé módosított változata. (A tanulási együttható itt nem a szomszédossági függvény része.)

A 8. kódrészletben megfigyelhető, hogy egy iteráció során hogyan változnak meg a rácson elhelyezkedő neuronok súlyvektorai. A program a függvényt több lépésben hajtja végre. A 17. sorban kivonja az aktuális bemenet vektorjából a neuron vektorját. Ezután ennek az eredményét a 18. sorban skalárisan szorozza a szomszédossági függvény értékével, majd ugyanezt teszi a 19. sorban a tanulási együtthatóval. Végül az így kapott eredményt összeadja a neuron eredeti vektorjával. Ezzel be is fejeződik a kiválasztott neuron súlyvektorának adaptálása.

**1** public void UpdateLattice(Vector inputVector, ColorNeuron[,] lattice,

**2** ColorNeuron BMU, int numberOfIterations, int iteration, double LC, double SD)

**3** {

**4** int latticeX = lattice.GetLength(0);

**5** int latticeY = lattice.GetLength(1);

**6**

**7** for (int x = 0; x < latticeX; x++)

**8** {

**9** for (int y = 0; y < latticeY; y++)

**10** {

**11** ColorNeuron currentNeuron = lattice[x, y];

**12** Vector oldVector = currentNeuron.Weights;

**13**

**14** double distance = Function.EuclideanDistance(currentNeuron, BMU);

**15** double NF = Function.NeighborhoodFunction(distance, SD);

**16**

**17** Vector newVector = Vector.VectorSubtraction(inputVector,oldVector);

**18** newVector.ScalarMultiplication(NF);

**19** newVector.ScalarMultiplication(LC);

**20** newVector = Vector.VectorAddition(oldVector, newVector);

**21**

**22** lattice[x, y].Weights.Components = newVector.Components;

**23**

**24** }

**25** }

**26** }

8. kódrészlet

**Szomszédossági függvény**:

, (6)

programbeli implementálását az 5. kódrészlet írja le.

**Szomszédossági függvény szórási értékét meghatározó függvény**:

, (8)

programbeli implementálását a 6. kódrészlet írja le.

**Tanulási együtthatót meghatározó függvény**:

, (10)

programbeli implementálását a 7. kódrészlet írja le.

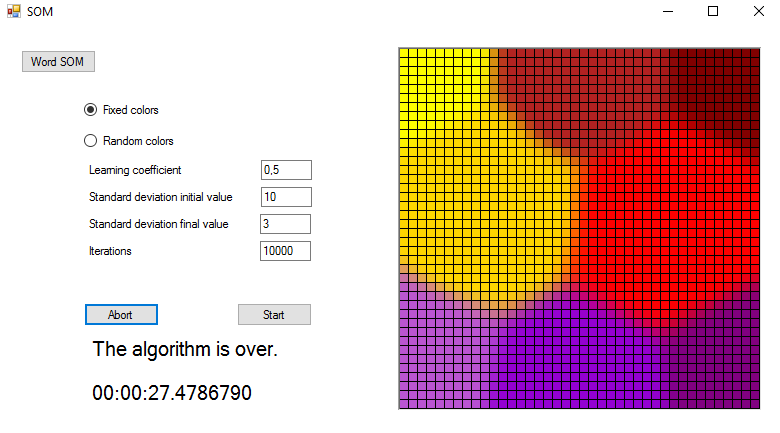
A szabadon megválasztható paramétereket a 2. táblázat foglalja össze:

2. táblázat. Színek klaszterezésére alkalmazott algoritmus paraméterezési lehetőségei a programban

|  |  |  |  |
| --- | --- | --- | --- |
| **magyar megnevezés** | **angol megnevezés** | **képletekben szereplő jelölése** | **programban szereplő jelölése** |
| iterációk száma | number of iterations | *t*max | numberOfIterations |
| tanulási együttható | learning coefficient | α | LC |
| szórás kezdeti értéke | initial value of standard deviation | σi | SDI |
| szórás végleges értéke | final value of standard deviation | σf | SDF |

### 3.4.2. Az implementált SOM algoritmus végrehajtása

#### A program felhasználói felülete (ColorForm)



A felhasználói felület gyakorlatilag két panelből épül fel: bal oldalon a **kontroll panel** (contolPanel), míg jobb oldalon a **kimeneti panel** (outputPanel) található. A kontroll panel segítségével állítható be, hogy milyen paraméterekkel szeretnénk futtatni az algoritmust. A bemeneti adatok beállítására kétféle lehetőség van. **Fixed colors** esetén minden futtatáskor ugyanazt a 8 előre meghatározott színt fogja használni az algoritmus. **Random colors** választása esetén minden futáskor véletlenszerűen generál a program bemeneti színeket.

A szövegdobozokban a következő paraméterek beállítására van lehetőség:

* **Learning coefficient** – tanulási együttható: értéke 0 és 1 lehet.
* **Standard deviation initial value** – szórás kezdeti értéke: 0-nál nagyobb szám, megválasztásánál figyelembe kell venni a neuronok számát.
* **Standard deviation final value** – szórás végleges értéke: 0-nál nagyobb szám, a szórás kezdeti értékénél kisebbnek kell lennie.
* **Iterations** – iterációk száma: tetszőleges egész szám.

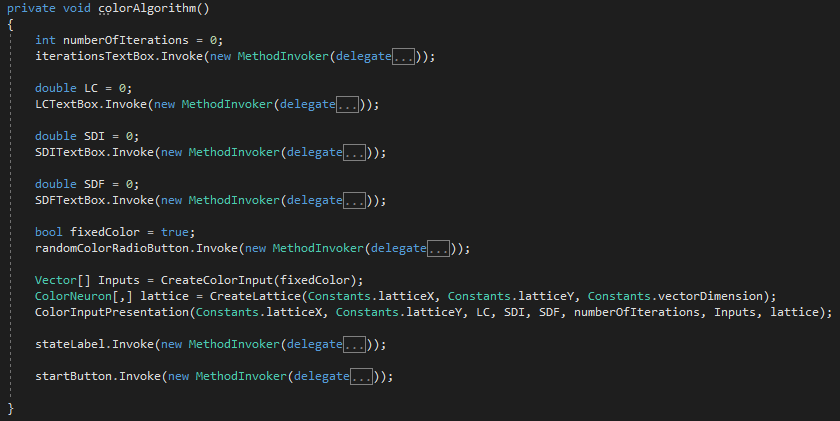
A **Start** gombbal elindíthatjuk az algoritmus futását, míg az **Abort** gomb véglegesen megszakítja azt. A bal felső sarokban található **Word SOM** gombra kattintva megnyílik a szavak klaszterezéséhez használatos űrlap. A felület jobb oldalán a **kimeneti panel** látható, mely az eredmény megjelenítésére lett kialakítva.

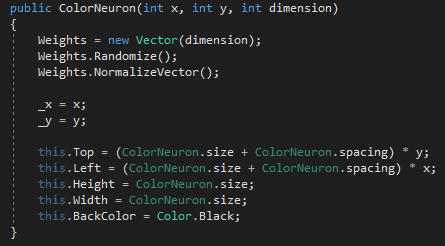
.

A felhasználó felület bal oldalán található a **kontroll panel**, amelynek segítségével beállíthatóak az algoritmus paraméterei. A Start gombbal elindíthatjuk az algoritmus futását, míg az Abort gomb megszakítja azt. A bal felső sarokban található Word SOM gombra kattintva megnyílik a szavak klaszterezéséhez tervezett űrlap. A felület jobb oldalán a **kimeneti panel** látható, mely az eredmény megjelenítésére lett kialakítva.

#### Az algoritmus megvalósítása

Az algoritmus elindítása után egy külön szálon indul el, ez teszi lehetővé, hogy közben az űrlapon található elemeket továbbra is lehessen kezelni. Jelen esetben az űrlapon megadott változó paraméterek kerülnek először beolvasásra. Ezután egy vektor tömböt hozunk létre, ebben tároljuk a bemeneti adatokat, jelen esetben a színek RGB kódjait. (Ez lehet színek egy előre meghatározott csoportja, vagy véletlenszerűen generált.)

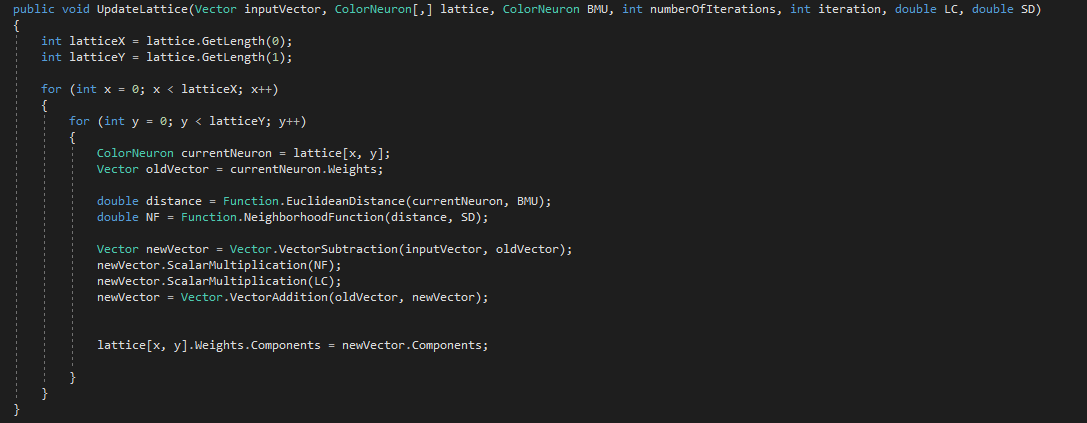




Ezután egy neuronokból álló rácsot hozunk létre, ahol minden egyes neuronhoz egy véletlenszerű egységvektor tartozik.

### 

Innentől kezdődik maga a tanulási folyamat. A 8 bemenetből véletlenszerűen választunk egyet, majd megkeressük azt a neuront, amely súlyvektora leginkább hasonlít az input vektorra, azaz az Euklideszi távolságuk minimális. Erre a nyertes vektorra angolul gyakran szokás **Best Matching Unit**-ként hivatkozni. Meghatározzuk a σ(t) függvény aktuális értékét, ezután pedig adaptáljuk a neuronokat a kiválasztott bemenethez. Minden ezredik iteráció után megjelenik a neuron rács aktuális állapota.



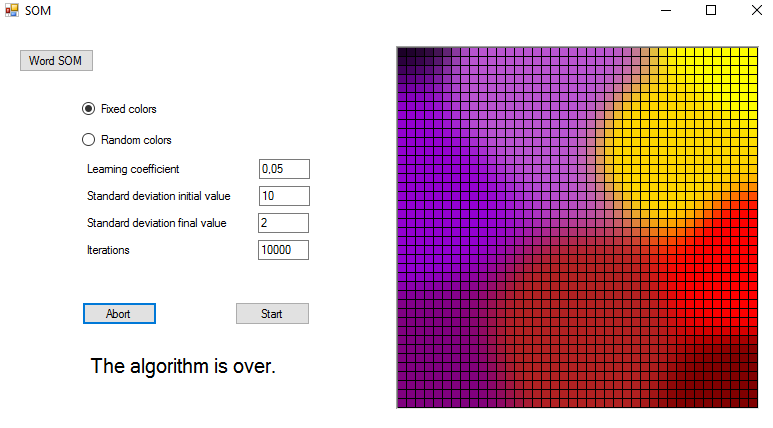
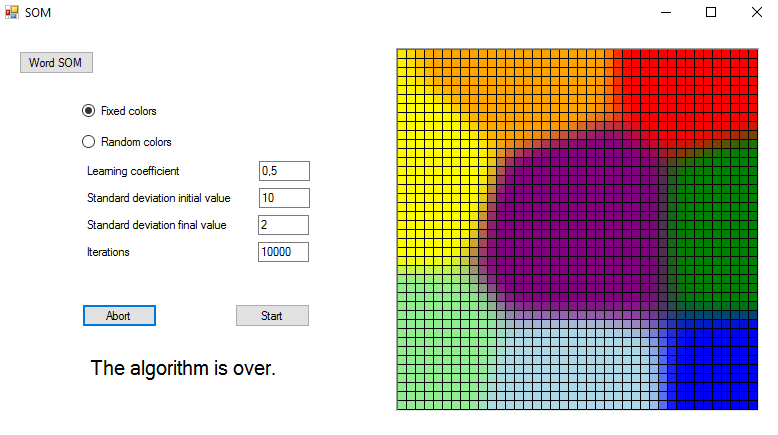
### 3.4.3. Eredmények összehasonlítása különböző paraméterezés esetén

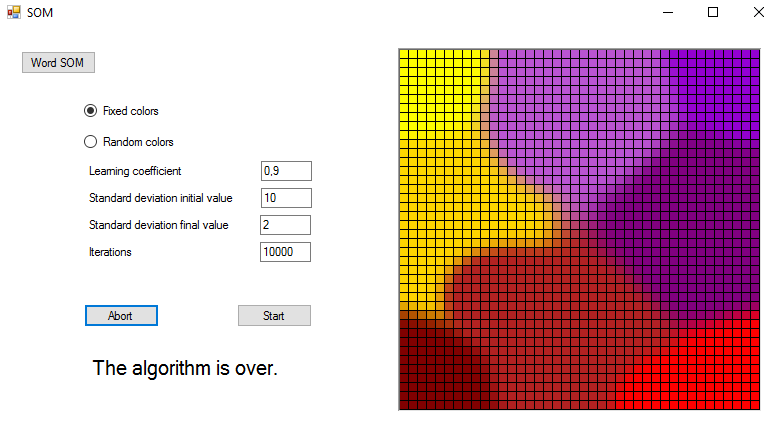
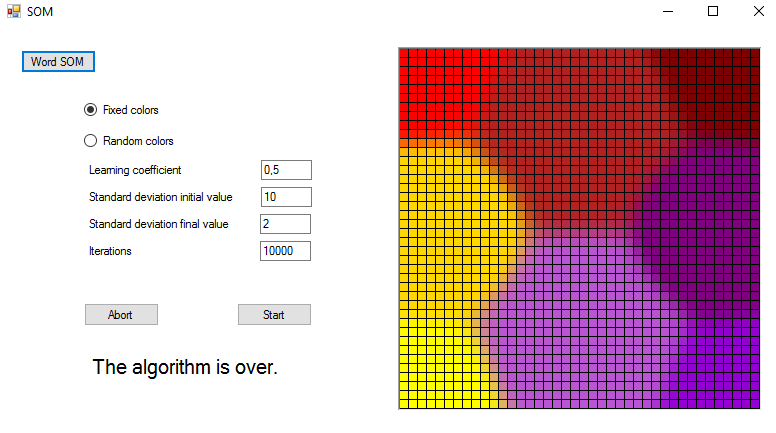
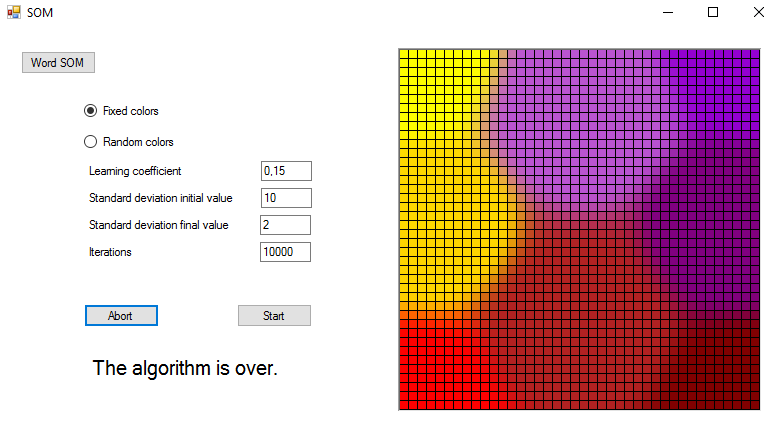
1, különböző input csoportok

2, különböző tanulási együtthatók

3, különböző SD értékek

4, iterációk száma





## 3.4. Szavak klaszterezése

Ahhoz, hogy szöveges dokumentumokból szemantikus önszerveződő térképeket hozhassunk létre mindenképpen szükséges a dokumentumok előfeldolgozása. Ebben az esetben jóval nehezebb a SOM algoritmus által feldolgozható bemeneti értékek létrehozása, mivel valahogy definiálnunk kell a távolság fogalmát…

* szöveg vs. metrika diszkrét szimbólumok
* hogyan lehet reprezentálni a szimbólumokat hogy ne legyen köztük kapcsolat
* kontextus megalkotása
* input megalkotása

Vektor -> Word -> Context -> Average Context -> Input

# 4. Összegzés

# 5. Irodalomjegyzék

Akinduko, A. A. – Mirkes, E. M. – Gorban, A. N. (2015): SOM: Stochastic initialization versus principal components. <http://www.sciencedirect.com/science/article/pii/S0020025515007318>, Letöltés dátuma: 2017. május. 1.

Altrichter Márta – Horváth Gábor – Patak Béla – Strausz György – Takács Gábor – Valyon József (2006): Neurális hálózatok. Panem Könyvkiadó Kft., Budapest.

Barbarino, F. – Boinee, P. – De Angelis A. (2008): Multidimensional data classification with artificial neural networks. <https://arxiv.org/pdf/cs/0412023.pdf>, Letöltés dátuma: 2017. május. 1.

Bella, M. A. B. – Eloff, J. H. P. – Olivier, M. S. (2009): A fraud management system architecture for next-generation networks. Forensic Science International, Vol. 185. No. 1-3, pp. 51-58.

Chandrashekar, B. H. – Shoba, G. (2009): Classification of documents using Kohonen’s self-organizing map. International Journal of Computer Theory and Engineering, Vol. 1. No. 5, pp. 610-613.

Deboeck, G. – Kohonen, T. (1998): Visual explorations in finance with self-organizing maps. Springer, London.

 Drachman, D. (2005): Do we have brain to spare? Neurology, Vol. 64. No. 12.

Fajszi Bulcsú – Cser László – Fehér Tamás (2010): Üzleti haszon az adatok mélyén. Alinea Kiadó, Budapest.

Fan, V. – Wallace, L. – Rich. S. – Zhang. Z. (2006): Tapping the power of text mining. Communications of the ACM, Vol. 49. No. 9., pp. 77-82.

Graupe, D. (2007): Principles of Artificial Neural Networks. World Scientific Publishing Co. Pte. Ltd., Chicago.

Haykin, Simon (1999): Neural networks - A comprehensive foundation. Prentice-Hall, Ontario.

Kohonen, T. – Oja, E. – Simula, O. – Visa, A. – Kangas, J. (1996): Engineering applications of the Self-Organizing Map. Proceedings of the IEEE, Vol. 84. No. 10, pp. 1358-1384.

Kohonen, T. (2014): MATLAB Implementations and Applications of the Self-Organizing Map. Unigrafia Oy, Helsinki.

Miller, A. S. – Coe, M. J. (1995): Star/galaxy classification using Kohonen self-organizing maps. Monthly Notices of the Royal Astronomical Society, Vol. 279. No. 1, pp. 293-300.

Natita, W. – Wiboonsak, W. – Dusadee, S. (2016): Appropriate Learning Rate and Neighborhood Function of Self-organizing Map (SOM) for Specific Humidity Pattern Classification over Southern Thailand. International Journal of Modeling and Optimization, Vol. 6 No. 1.

Piskorski, J. – Yangarber, R. (2013): Information Extraction: Past, Present and Future. In Poibeau, T. – Saggion, H. – Piskorski, J. – Yangarber, R. (eds.): Multi-source, Multilingual Information Extraction and Summarization. Springer Berlin Heidelberg, pp. 23-49.

Platek, S. M. – Keenan, J. P. – Shackelford, T. K. (eds., 2007): Evolutionary Cognitive Neuroscience. The MIT Press, Massachusetts.

Ritter, H. – Kohonen, T. (1989): Self-Organizing Semantic Maps. Biological Cybernetics, Vol. 61., pp. 241-254.

SOM implementation in SOM Toolbox. <http://www.cis.hut.fi/somtoolbox/documentation/somalg.shtml>, Letöltés dátuma: 2017. május 1.

Tikk Domonkos (2007): Szövegbányászat. Typotex, Budapest.

Tulankar, K. – Kshirsagar, M. – Wajgi, R. (2012): Clustering telecom customers using emergent self organizing maps for business probability. International journal of Computer Science and Telecommunications, Vol. 3. No. 1, pp. 256-259.

Veitinger, S. (2011): The Patch-Clamp Technique. <http://www.leica-microsystems.com/science-lab/the-patch-clamp-technique/>, Letöltés dátuma: 2017. május 1.

Xu, M. – Wong, T. C. – Chin, K. S. (2014): A medical procedure-based patient grouping method for an emergency department. Applied Soft Computing, Vol. 14. pp. 31-37.

Yuan-Chao Liu – Ming Liu – Xiao-Long Wang (2012): Application of Self-Organizing Maps in Text Clustering: A Review. <http://cdn.intechopen.com/pdfs-wm/37680.pdf>, Letöltés dátuma: 2017. május. 1.