Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnikai és Információs Rendszerek Tanszék

Gyógyszer hatóanyag jelöltek keresése és optimalizálása a molekulák terében

Készítette
Kiss Richárd

Konzulens
Sárközy Péter

2020

TARTALOMJEGYZÉK

Absztrakt

Napjainkban az új gyógyszermolekulák kifejlesztésének egyik kulcsfontosságú lépése a molekulajelöltek kiválasztása. A kiválasztott molekulák szintetizálása és tesztelése igen időigényes és költséges feladat, ezért szükséges, hogy egy megfelelő halmazt válasszuk meg. A generatív modellezés lényege, hogy megtanuljunk olyan molekulákat mintavételezni, amik megfelelnek több kikötésnek is. Ez a folyamat automatizálható és több megközelítés is létezik ennek megoldására. A dolgozatomban megvizsgálok és összehasonlítok két state-of-the-art *de novo* módszert az egy- és több-cél szerinti molekula generálásra. A két modell egyike a SeqGAN, ami egy általános szekvencia generálásra alkalmas módszer, valamint a MolDQN, ami a molekula generálást Markov döntési folyamatként modellezi. Megfelelő reprezentációt kell választanunk ahhoz, hogy ezek a modellek fel tudják dolgozni és értelmezni a molekulákat,. A molekulák szekvenciális reprezentációjára a SeqGAN modellhez SMILES-t használtam. A MolDQN számára ugyan lehetne használni szintén ezt, viszont a Morgan Fingerprint-et használva gyorsabb és stabilabb lesz a tanítás (ez a fajta reprezentáció bináris vektorok formájában kódolja a molekulákat). A két modell mellett bemutatok egy hibrid modellt, ami a SeqGAN-ra épül és a MolDQN-nél is használt jutalomfüggvényeket alkalmazza. Végül összehasonlítom ezeket más state-of-the-art módszerekkel olyan jellemzők szerint, mint: validitási arány, egyediségi arány, újdonsági arány és diverzitás, melyek a modellek jóságát számszerűsítik.

Abstract

Nowadays, collecting a good drug candidate set is a key step in every drug discovery process. Since it costs a lot of time and money to synthetise and test these candidates, it is necessary to use a proper set of molecules. In terms of generative modelling, the goal is to learn to sample molecules in the intersection of multiple property constraints. This can be automated and there are lots of approaches to this problem. In my dissertation I will review and compare two state-of-the-art de novo methods in the field of single- and multi-objective molecule generation. The two models are SeqGAN, which is a model that is capable to generate any kinds of series data and MolDQN which formulates the problem as a Markov Decision Process. Both use different representation of the molecules. For the SeqGAN model we need to use some kind of sequential representation of the molecules, for which I used SMILES. The other model could use this as well, however I decided to use Morgan fingerprints, which describes the molecules in a binary vector format. Using this makes learning faster, and more stable. Furthermore, I will introduce a hybrid model, which is based on the SeqGAN architecture and uses the reward functions that are introduced in the MolDQN paper. I will compare these models with other state-of-the-art methods using metrics that quantify the goodness of the models, like validity, uniqueness, novelty and diversity.

1. Bevezetés

Minden gyógyszerkutatási folyamat egy meghatározó része a megvizsgálandó molekulák halmazának (jelölt halmaz) megválasztása. A halmaz mérete jelentősen befolyásolhatja a kutatás költségeit és időtartamát is. Egy túlzottan nagy halmaz minden elemének szintetizálása és tesztelése túl sok erőforrást vehet igénybe, míg, ha a halmaz túl kicsi, akkor potenciálisan ígéretes molekulákat kihagyhatunk.

A jelöltek megválasztásának automatizálására sok megközelítésből léteznek algoritmusok. Ebben a dolgozatbat két state-of-the-art módszert és pár egyszerűbb megközelítést fogok megvizsgálni és összehasonlítani.

Az egyik state-of-the-art módszer az általános szekvencia generálására alkalmas SeqGAN [2], a másik pedig a MolDQN [3], ami egy tisztán megerősítéses tanulás alapú modell.

A dolgozatomban ismertetett modellekhez saját implementációt készítettem, minden mérést ezeken végeztem.

1.1. Molekula jellemzők

Ahhoz, hogy egy jelölthalmaz, és a benne levő molekulák jóságát minősíthessük, valamilyen módon számszerűsítenünk kell a molekulák jóságát, ezeket hívjuk *targeteknek*. *Targetnek* tetszőleges molekula jellemzőt (vagy jellemzők kombinációját) megválaszthatunk, a dolgozatom során a következőket jellemzőket használtam:

- **QED*** – (Quantitative Estimate of Drug-likeness) azt adja meg, hogy mennyire „gyógyszerszerű” egy adott molekula egy [0,1] intervallumon. (Nagyobb érték jobb)
- **SAS*** – (Synthetic Accessibility Score) azt jellemzi, hogy milyen nehezen szintetizálható egy adott molekula egy pozitív valós számmal. (Kisebb érték jobb)
- **LogP*** – egy oldhatósági mutató. (Az 1-octanol-ban és vízben való oldhatóságok logaritmusának különbsége)
- **Cycle Count** – a molekulagráfban található körök száma
- **Molecular Weight** – molekula tömege
- **Tanimoto similarity (továbbiakban SIM)** – két molekula közötti hasonlósági érték a [0,1] intervallumon a molekulák kémiai struktúrái alapján. Általában akkor nevezünk hasonlónak két molekulát, ha ez az érték nagyobb, mint 0.85 [13] (Nagyobb érték nagyobb hasonlóságot jelent)

Fontos megjegyezni, hogy a *-al jelölt jellemzőket közelítő módszerekkel számítjuk, így a jelölthalmaz kereső módszereink csak annyira lehetnek jók, mint ezek a közelítések. Ha ezek a közelítések rosszak, vagy hibásan olyan targetet választunk, ami nem a valóságnak –céljainknak– megfelelő (lásd 4. fejezet), akkor értelmetlenné válhat az optimalizálás.

1.1.1. Célfüggvények (R)

Az alábbi felsorolásban összegyűjtöttem, hogy milyen célfüggvényeket vizsgáltam meg és hogy milyen megfontolás van ezek mögött. (A dolgozatban *C*-vel jelölöm a célfüggvényt)

- *C(mol) = 5 * QED_mol – SAS_mol*: a függvényben a pozitív együtthatós *QED* tag azokat a molekulákat jutalmazza, amik minél gyógyszerszerűbbek, a *SAS* büntetőtag azokat bünteti, amiket nehéz szintetizálni. Összességében olyan molekulákra ad magas értékeket ez a függvény, amik gyógyszerszerűek és könnyű előállítani őket.
- *C(mol) = LogP_mol – SAS_mol – CycleCount_mol:* a szakirodalomban gyakran előforduló benchmark target. Olyan molekulákra ad magas értékeket, amiket könnyű előállítani, kevés gyűrű van bennük és jól oldódnak zsírban.
- *C(mol) = w * QED + (1-w) * SIM(m0, mol):* ezzel a függvénnyel olyan molekulákat kereshetünk, amik egy már ismert hatóanyaghoz (*m0*) hasonlóak, de magasabb QED értékkel rendelkeznek. A *w* paraméterrel szabályozható, hogy a hasonlóságot, vagy a gyógyszerszerűséget részesítjük előnyben.

1.2. Módszerek jóságának számszerűsítése

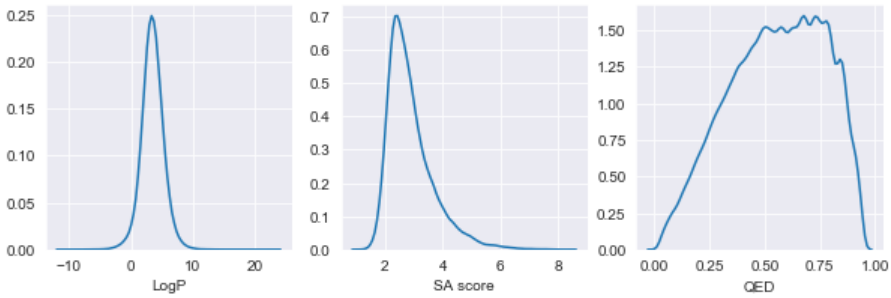
Ahhoz, hogy a jelöltek keresésére használt modelleket összehasonlíthassuk, szükségünk van pár olyan jellemzőre, ami ezen modellek jóságát számszerűsíti:

- *Validity* – megadja, hogy a modell milyen arányban generál *érvényes* szavakat, ahol a számítás módja az érvényes szavak száma osztva generált szavak számával.
- *Az optimalizálandó target/targetek szerint maximum értékek.*
- *Módszer futási ideje.*
- *Módszer futása közben felhasznált memória.*

Ahol a modellek kimenetére SMILES (lásd 1.4. fejezet) szavak listájaként tekintünk és a *generált szavak* a modellből mintavételezett kimenetek, az *érvényes szavak* azon szavak, amik megfelelnek a SMILES szintaxisnak és az RDKit [10] segítségével molekula objektummá parse-olhatók.

1.3. Az adathalmaz

A SeqGAN modell tanításához szükség van egy adathalmazra. Én a *guacamol* molekula adathalmazt választottam, amiben kb. 1.6 millió molekula található SMILES formátumban. Az adathalmaz arra is alkalmas, hogy az optimalizálandó függvény szerint megkeressük benne a legjobb N darab molekulát, azok jóságát össze tudjuk hasonlítani a mi modelleink által előállított jelölthalmazokkal és így képet alkothatunk arról, hogy menynyire „jó” egy modell.



1. ábra - A Guacamol adathalmazban található molekulák eloszlásai LogP, SA score és QED szerint.

1.4. Molekula reprezentációk

Ahhoz, hogy az algoritmusok tanulni és keresni tudjanak a molekulák terében, olyan reprezentációkra van szükségünk, amelyeket fel tudnak dolgozni. A SeqGAN számára szekvenciális reprezentációként a SMILES-t választottam.

Egy molekula SMILES reprezentációja a molekuláris gráfból úgy kapható meg, hogy a gráfban található körök felbontása után DFS-el (Depth First Search, mélységi kereséssel) bejárjuk azt, és az atomokat egy listába fűzzük. Az atomokat és kötéseket ASCII karakterekként kódoljuk.

A MolDQN modellnél a molekulák Morgan Fingerprint-jét használtam. Ez egy olyan bináris vektor reprezentációja a molekuláknak, ami a molekula szerkezetét és kémiai tulajdonságait tömöríti. A vektor méretét mi választhatjuk meg, én a munkám során 1024 hosszú vektorokat használtam.

1.5. Markov Döntési Folyamat (MDP) és Megerősítéses Tanulás (RL)

Mivel a dolgozatomban vizsgált modellek Markov döntési folyamatokat és megerősítéses tanulást használ, itt röviden összefoglalom a lényegesebb, a dolgozatban felhasznált fogalmakat.

1.5.1. Markov Döntési Folyamat [1]

Az MDP-t a következők alkotják:

- **Állapotok: *S*** – a lehetséges állapotok halmaza (*s*-el jelöljük a jelenlegi állapotot)
- **Cselekvések: *A*** – a lehetséges cselekvések halmaza. *A(s)*-el jelöljük az *s* állapotban választható cselekvéseket.
- **Jutalom:** egy függvény, ami állapotokhoz és/vagy cselekvésekhez vagy tranzíciókhoz rendel egy valós számot: *R(s), R(s, a), R(s, a, s')*
- **Környezeti modell:** meghatározza, hogy egy adott állapotban adott cselekvés választására milyen állapotba kerülünk. A környezeti modell lehet determinisztikus, vagy sztochasztikus:
    - Determinisztikus környezeti modell esetén a tranzíciók egyértelműek, minden *(s, a)* párhoz egy *s'* állapotot rendel (*s* és *s'* lehet azonos).

    - Szochasztikus környezeti modellnél az átmeneti függvény a következő állapot valószínűségi eloszlása feltételezve a jelenlegi állapotot és a választott cselekvést: *P(s' | s,a)*, az átmeneteket ebből mintavételezve kaphatjuk meg.
- **Policy (π):** egy megoldás az MDP-re. A policy cselekvést rendel az állapotokhoz. A Markov Döntési Folyamatoknál tipikusan egy olyan policy megtalálása a feladat, ami maximalizálja az összes jutalom várható értékét: $J = E[\sum_{t=0}^{\infty} \gamma^t R(s_t, s_{t+1})]$, ahol gamma a leszámoltatási tényező[1] (0,1]. Azt a policy-t, ami maximalizálja *J*-t, optimális policy-nek hívjuk és π*-al jelöljük.
- **Trajektória:** egymást követő állapotok listája.

1.5.2. Molekula építés, mint Markov Döntési Folyamat [3]

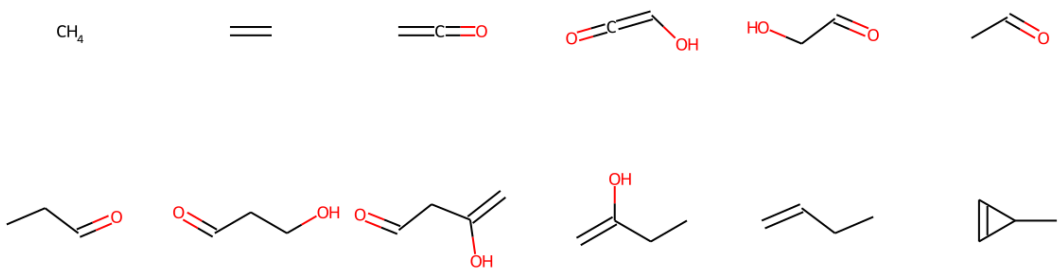
- Az állapotok halmaza (*S*) az összes valid[2] molekula.
- Három elemi műveletet definiál a cikk, amikből képezhető az összes *s* állapotból elérhető következő állapot:
    - Kötés hozzáadása – egy N-es kötés N+1-es kötéssé alakítása
    - Kötés eltávolítása – egy N-es kötés N-1-es kötéssé alakítása
    - Atom hozzáadása – egy atom hozzáadása a jelenlegi molekulához

  Egy kötés lehet 0-s, 1-es, 2-es vagy 3-as kötés. 0-s kötést a kötés eltávolításával érhetünk el, ez megfelel egy atom eltávolításának. Olyan kötés nullázása nem engedélyezett, amivel a molekula több, nem csak egy atomot tartalmazó komponensre szakadna.

  Ezeket a műveleteket az összes lehetséges módon használva képezzük az *s* állapotból (jelenlegi molekulából) a következő lehetséges állapotok halmazát. Az MDP-beli cselekvések (és számuk) a jelenlegi (*s*) állapottól függenek. A cselekvések hatására 1 valószínűséggel kerülünk át a hozzá tartozó *s'* állapotba.

  Fontos megjegyezni, hogy ha egy érvényes molekulából indulunk ki, akkor ezekkel a műveletekkel csak érvényes molekulákba juthatunk – így az ezt használó modellek validity értéke 100% lesz.
- Az *s* állapotból elérhető állapotokhoz vezető cselekvések halmazához (*A(s)*) hozzávesszük azt a cselekvést, ami *s* állapotba vezet vissza, ez a „megállás" cselekvés. Ennek választása befejezi a trajektóriát.
- Mivel az állapotátmenetek valószínűsége 1 minden cselekvéshez, ezért *C*-t definiálhatjuk a cselekvésekre is a következő képpen:
    - $C(a) = C(s'), a \in A(s)$ és a cselekvés hatására *s'* állapotba kerülünk
- A jutalomfüggvény akármilyen target lehet.

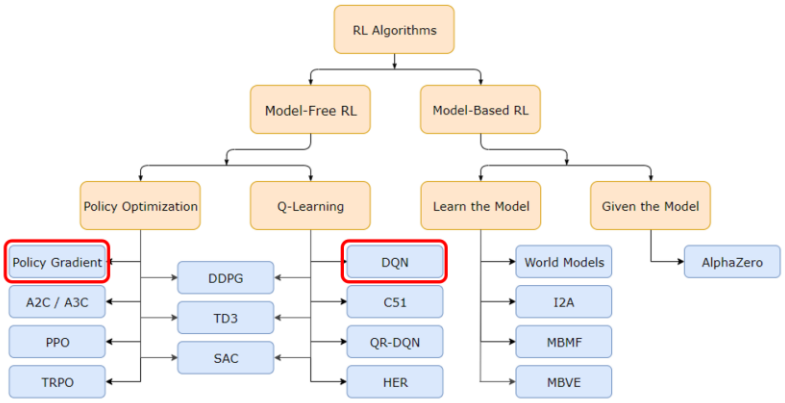
[1] A leszámolási tényezőnek két fontos szerepe van. Első, hogy a végtelen sor konvergenciáját biztosítsa (esetünkben ez nem releváns, mert a trajektóriák véges hosszúak). A második az, hogy a γ által a későbbi jutalmak kevésbé lesznek értékesek számunkra, mint amiket hamarabb megkapunk.
[2] RDKit-tel parse-olható molekula objektummá



2. ábra - Példa egy véletlenszerű trajektóriára

1.5.3. Megerősítéses Tanulás

A Megerősítéses Tanulás a Mesterséges Intelligenciának (MI) egyik ága, ami Markov Döntési Folyamatokra keres optimális megoldást. Azokat az algoritmusokat, amik direkten nem férnek hozzá a környezeti modellhez, modellmentes (Model-Free) algoritmusoknak nevezzük. A dolgozatban megvizsgált két modell (SeqGAN és MolDQN) ilyen, modellmentes algoritmusokat alkalmaz, névszerint a *Policy Gradientet* [6] és a *DQN*-t [4].



3. ábra - Megerősítéses Tanulás algoritmusok taxonómiája [14]

2. Modellek

A dolgozatban ismertetett algoritmusokhoz készített saját implementációk Python 3.7 [12] nyelven készültek.

Felhasznált python csomagok:

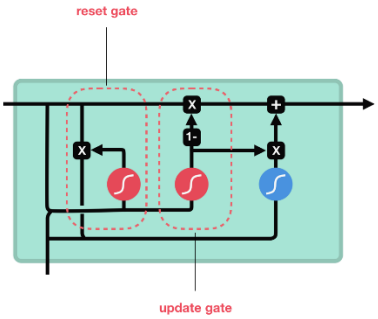
- RDKit [10]: Molekulák kezelése, jellemzők, targetek számítása
- Numpy [16]: Mátrixműveletek végzése
- Pytorch [11]: Neurális hálózatok és gradiens eljárás

2.1. SeqGAN

A SeqGAN modell egy általános szekvencia generálására alkalmas modell. A GAN-okhoz hasonlóan ez is két neurális hálózatból áll, egy *generátorból*, és egy *diszkriminátorból*. A molekulák generálását a *generátor* hálóval végezzük, a *diszkriminator*t tanításhoz használjuk.

2.1.1. Rekurrens hálózat

Mivel szekvenciák generálásáról van szó, szükséges valamilyen rekurrens elem beépítése a modellbe, hogy ne csak a pillanatnyi, hanem az eddigi kontextus alapján tudjon döntéseket (karaktereket) választani. Én a modellhez a GRU-t (Gated Recurrent Unit) [5] választottam. Ez egy kevésbé komplex fajtája a rekurrens celláknak. Azért ezt választottam, mivel működése során ez kevesebb tenzor műveletet végez, mint egy másik gyakran használt rekurrens cella, az LSTM [5], ezáltal valamivel gyorsabb.

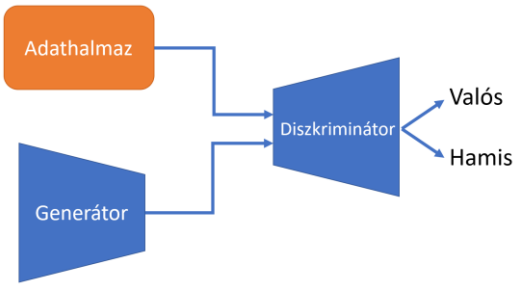


4. ábra - GRU cella [15]

Ahogy a képen is láthatjuk, a cellának két kapuja van, a reset kapu és az update kapu. Az update kapu feladata, hogy eldöntse hogy milyen információt engedjen át a cella és mit „nullázzon” ki. A reset kapu azt szabályozza, hogy a cella milyen régi információt vegyen figyelembe, azaz milyen mértékben „felejtsen”.

2.1.2. GAN architektúra

A GAN-ok (Generative Adversarial Network) a generatív modellek[3] közé tartoznak, melyeket jellemzően nem felügyelt tanítással tanítunk. Ezek a modellek általában adatok eloszlását tanulják, majd ezekből mintavételezve a tanítóadatokhoz hasonló eloszlású mintákat kaphatunk. A GAN-ok általában két részből állnak, egy *generátorból* és egy *diszkriminátorból*. A *diszkriminátor* feladata, hogy egy a bemenetére érkező mintáról eldöntse, hogy az a tanító adathalmazból, vagy a *generátortól* származik. A *generátor* feladata, hogy megtévessze a *diszkriminátort*, olyan mintát állítson elő, hogy arról a *diszkriminátor* ne tudja megállapítani, hogy nem a tanító halmazból való. A két hálózat együttes tanítását nevezzük *versengő (adversarial)* tanításnak.



5. ábra – Általáno Generatív Adversarial Network architektúra

2.1.3. Tanítás

Az együttes modell tanításának két fázisa van, amikor a *generátort* tanítjuk és amikor a *diszkriminátort*.

A *diszkriminátor* tanításakor véletlenszerű mintákat választunk az adathalmazból (valós adatok) és a *generátor* hálózatot is mintavételezzük („hamis” adatok). A *diszkriminátor* elvárt kimenete a valós adatokra 1, a hamisokra 0. A hálózatot gradienssüllyesztéssel tanítjuk, és veszteségfüggvényként bináris keresztentrópiát használunk.

A *generátor* tanításánál lehetőség lenne arra, hogy direkten a *diszrkiminátort* felhasználva, szintén bináris keresztentrópia veszteségfüggvény szerint tanítsuk úgy, hogy az elvárt kimenet a hamis adatokra 1. Ekkor a gradienst vissza tudnánk terjeszteni a *diszkriminátoron* keresztül, majd azt felhasználva szintén valamilyen gradienssüllyesztési algoritmussal taníthatnánk a *generátort*. Ehelyett viszont a SeqGAN egy megerősítéses tanulásból vett módszert alkalmaz, a *Policy Gradientet (PG)*. A *generátorral* egy policy-t tanulunk arra, hogy milyen karaktereket rakjunk egymás után. Ez abból a szempontból nagyon előnyös számunkra, hogy a *PG* módszerrel lecserélhető a *diszkriminátor* kimenete akármilyen más optimalizálandó jutalomfüggvényre.

[3] A generatív modellek közül még említésre méltó az Autoenkóder és Variációs Autoenkóder [8], ezeket is nagyon jól lehet használni molekulajelölt keresési feladatokra. Ezekkel dolgozatomban nem foglalkozok.

Policy Gradient módszer

A *PG* esetében magát a policy-t, az állapot függvényébe, a cselekvések fölötti valószínűségi eloszlást (vagy akár egyből ezek logaritmusát) tanuljuk. A következő veszteségfüggvényt próbáljuk meg maximalizálni:

$$J(\theta) = E[\sum_{t=0}^{T-1} r_{t+1} |\pi_\theta].$$

A $\frac{d}{dx}\log f(x) = \frac{f\prime(x)}{f(x)}$ azonosságot felhasználva (így kerül bele a policy logaritmusa) eljuthatunk a célfüggvény deriváltjához, amit használni fogunk gradiens süllyesztésnél:

$$\nabla_\theta J(\theta) \sim \sum_{t=l}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) * r_t$$

A cikkben a *diszkriminátort* használják jutalomfüggvényként, viszont én kipróbáltam más lehetőségeket is:

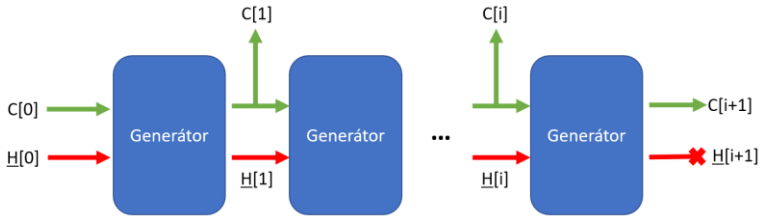
- Ha $r_i = 1$ bármely i-re és nem a *generátorból* mintavételezett szekvenciákat, hanem az adathalmaz egyes szekvenciáit használjuk bemenetként, akkor a hálózatnak azt tanítjuk, hogy egy bemeneti szekvenciára következő karakternek a bemeneti szekvencia következő karakterét válassza. Ebben az esetben lehetne akár keresztentrópia veszteségfüggvényt is használni, viszont *PG*-vel is teljesen jól működik
- Ha r_i -t az optimalizálandó célfüggvények egyikének választjuk meg, akkor azzal a modellnek azt taníthatjuk, hogy a magasabb jutalmat eredményező szekvenciákat válassza a kisebb jutalmúakkal szemben. (Ebben az esetben egy trajektóriának az utolsó állapotára – a kész stringre – számított célfüggvény értéket használjuk.)
- Ha r_i -t a *diszkriminátor* kimeneteként választjuk meg, akkor hasonló hatást érhetünk el, mint amikor $r_i = 1$ bármely i-re.

2.1.4. Generálás

Annak érdekében, hogy a *generátor* hálózat minél diverzebb mintákat generáljon, (felfedezzen) a kimeneti eloszlásból nem a legjobb cselekvést választjuk, hanem mintavételezzük azt. A generálás az alábbiak szerint történik:

1. inicializáljuk a GRU rejtett réteget (H[0]) egy csupa 0 vektorra és válasszunk egy véletlenszerű kezdő karaktert (C[0]) és inicializálunk egy üres kimeneti listát
2. a *generátor* hálózaton átvezetve az előző rejtett vektort (H[i]) és a bemeneti karaktert (C[i]), a kimeneti eloszlásból mintavételezzük a választott karaktert (ez lesz C[i+1]) és hozzáfűzzük a kimeneti listához
3. ha a kimeneti lista utolsó karaktere <*eos*>[4], akkor megállunk, különben a 2 lépéssel folytatjuk.

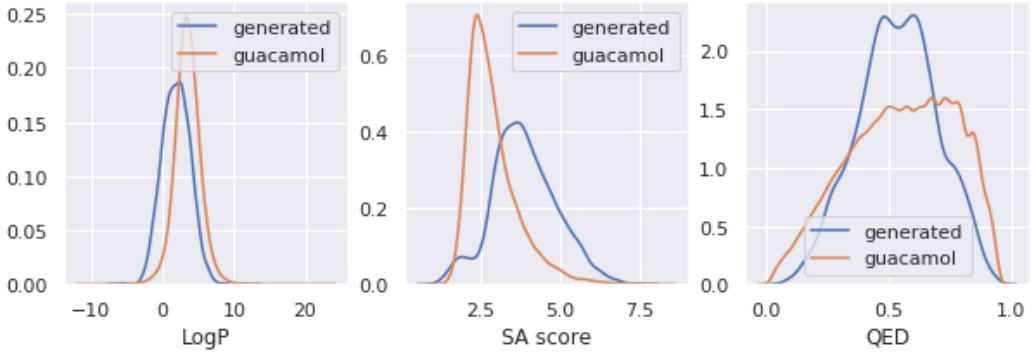
[4] eos – end of string – a szöveg végét jelző token



6. ábra – Szekvencia generálása a *generátor* hálózattal

2.1.5. Tanítás

A modell tanítását követően mintavételeztem egy 10000 méretű molekula halmazt a modellből, majd ezek eloszlását összehasonlítottam a tanító adathalmazéval:



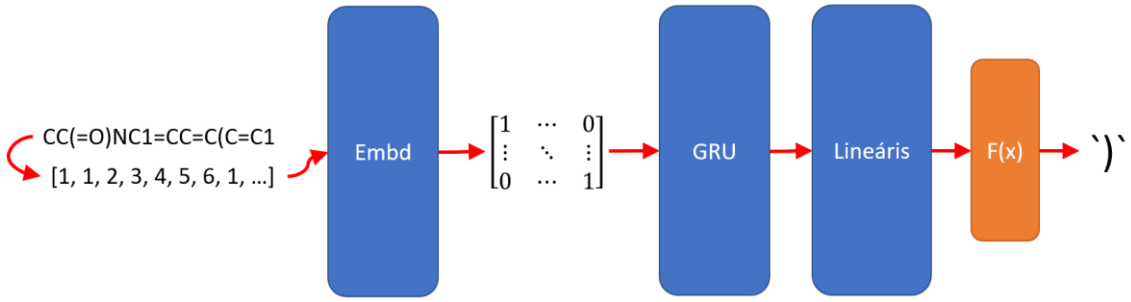
7. ábra – SeqGAN modellel generált molekulák eloszlása a tanító adathalmazhoz (Guacamol) képest, LogP, SA score és QED szerint

A generált SMILES szavak 99%-a nem volt bent a tanító adathalmazban. Ez elsőre meglepően jó aránynak tűnhet, viszont fontos figyelembe venni, hogy egy molekulának több SMILES reprezentációja lehet attól függően, hogy a molekulagráf köreit hol bontjuk fel és hol kezdjük a bejárását. A generált adathalmaz és a Guacamol molekuláinak metszetét meghatározni túlzottan számításigényes, ugyanis minden molekula párra hasonlóságot kellene számítani, ami 10E * 1.6M darab hasonlóság kiszámítását jelentené.

2.1.6. Implementáció

Az implementációmban a *generátor* hálózat a következő elemekből áll:

- Az első egy *Embedding* réteg, ami a bemeneti karakterekből egy *one-hot* vektort állít elő (a *one-hot* vektor dimenzióinak a száma egyenlő a tanító adathalmazban használt karakterek számával, ahol minden dimenzió egy-egy karakternek felel meg),
- Egy GRU réteg, aminél a rejtett vektor mérete 256,
- Egy lineáris réteg, ami a GRU rejtett rétegét leképezi egy akkora vektorra, mint amit az Embedding réteg használ,
- Egy softmax aktiváció, amivel egy valószínűségi eloszlást állítok elő a kimeneti rétegből



8. ábra - Generátor architektúra – itt F(x) a softmax függvény alkalmazását és a mintavételezést jelenti az így kapott eloszlásból

A *diszkriminátor* hálózat a szinte teljesen ugyanaz, mint a *generátor*, azzal a kükönbséggel, hogy az lineáris réteg kimenete egy 1 méretű vektor (skalár) és a hálózat legvégén egy sigmoid aktiváció áll. A hálózat kimenete annak a valószínűsége, hogy a bemeneti minta valós (a modell által tanult paramétereket feltételezve).

Tanításnál bevezettem dropout rétegeket is mindkét hálózatba, hogy minél általánosabb modellt tanuljon, viszont azt tapasztaltam, hogy nem hajlamos a modell a túltanulásra (mivel kicsi a komplexitása), így az összehasonlításnál a dropout nélküli modellt használtam.

A következő fejezetekben bemutatott modellek mind a korábban ismertett (1.5.2. alfejezet) molekula építési MDP-t használják.

2.2. Random Walk

Ez egy igen egyszerű, referencia módszer. Egy tetszőleges molekulából kiindulva, minden cselekvést egyforma valószínűséggel választva járja be a molekulatér egy részét.

A felhasznált memória úgy csökkenthető, ha egy M méretű rendezett tömbbe szúrjuk be az új molekulákat és ha a tömbbeli legrosszabb molekulánál kisebb jutalom értékű a jelenlegi molekula, akkor eldobjuk.

RANDOM WALK

---

input:
N: bejárandó trajektóriák száma
M: kívánt jelölthalmaz mérete
k: bejárandó trajektóriák hossza
C: célfüggvény
output:
P: jelölthalmaz

1 *P* := üres, *M* méretű tömb.
2 *s* := S_0
3 *l* := 0 // trajektória lépéseinek száma
4 *a*-t egyenletes eloszlás szerint mintavételezzük *A(s)*-ből
5 *s'* := *a*-hoz tartozó állapot
6 *l* := *l* + *1*
7 **if** *l* = *k* **or** *s'* = *s* **then**
8 *P*-be beszúrjuk *s'*-t úgy, hogy *P* rendezve legyen *C* szerint

```
 9      goto 2.
10  else:
11      s := s'
12      goto 4.
13  endif
14  end
```

2.3. Naív mohó policy

Ez a policy mindig azt a cselekvést választja, ami maximalizálja a célfüggvényt.

$$\mathrm{a} = \underset{\mathrm{a} \in \mathrm{A}(s)}{\operatorname{argmax}}\{C(\mathrm{a})\}$$

2.4. Mohó-szerű sztochasztikus algoritmus

Ezt az algoritmust a Random Walk-ra épít, azzal a kiegészítéssel, hogy a jobb cselekvéseket nagyobb valószínűséggel választja.

Minden lehetséges következő állapotra kiértékeli a célfüggvényt, majd ezeket egy valószínűségi eloszlássá alakítja úgy, hogy átvezeti az összes értéket egy softmax függvényen. A következő lépésben ezen eloszlásból mintavételezzük a választandó cselekvést. A softmax-ból adódóan az eloszlásban a magasabb célfüggvény értékű cselekvéshez tartozó valószínűség nagyobb - azaz nagyobb valószínűséggel választja azt a cselekvést, ami magasabb jutalmat eredményez.

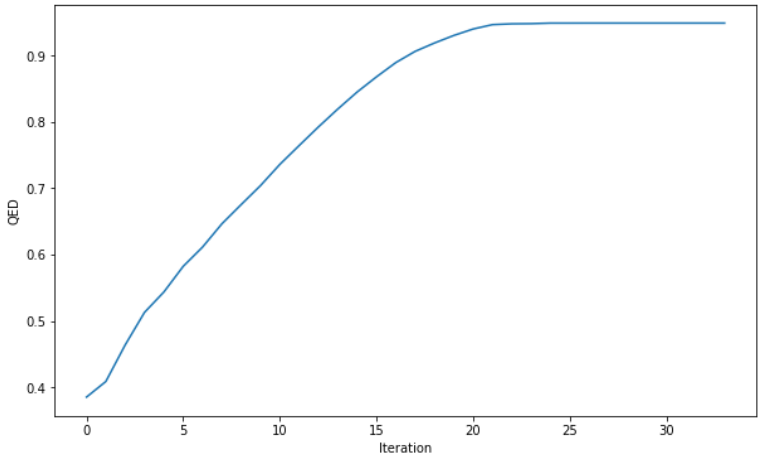
MOHÓ SZERŰ SZTOCHASZTIKUS

```
    input:
        N: bejárandó trajektóriák száma
        M: kívánt jelölthalmaz mérete
        k: bejárandó trajektóriák hossza
        S0: kezdeti állapot
        C: célfüggvény
    output:
        P: jelölthalmaz
 1  P := üres, M méretű tömb
 2  s := S0
 3  R := softmax({C(a) | a ∈ A(s)}) // multinomiális eloszlás a cselekvések felett
 4  a ~ R, s' := az a-hoz tartozó cselekvés // a cselekvést mintavételezzük R-ből és elvégezzük a tranzíciót
 5  if eddigi lépések száma = k or s' = s then
 6      P-be beszúrjuk s'-t úgy, hogy P rendezve legyen C szerint
 7      if nem jártunk még be N darab trajektóriát then
 8          goto 2.
 9      endif
10      goto 15.
11  else:
```

12 *s := s'*
13 **goto** 3.
14 **endif**
15 **end**

2.5. Top-M mohó algoritmus

Ez a policy hasonlóan működik, mint a sima mohó policy, azzal a különbséggel, hogy egyszerre nem egy, hanem M állapotban van. Minden iterációban egy halmazba rakja az összes jelenlegi állapotból egy lépéssel elérhető állapotot, a halmazhoz hozzáveszi a jelenlegi állapotokat és ezek közül kiválasztja a célfüggvény szerint legjobb M darabot. Az algoritmus az előre meghatározott N iteráció után áll le, vagy ha a jelölthalmaz két egymás utáni iterációban nem változik.



9. ábra - Jelölthalmaz *C (=QED)* szerinti legjobb molekulája minden iterációban, az algoritmus 34 iteráció után konvergált.

TOP-M MOHÓ

---

input:
N: iterációk maximális száma
M: kívánt jelölthalmaz mérete
S_0: kezdeti állapot
C: célfüggvény
output:
P: jelölthalmaz

1 *P* := üres, *M* méretű tömb
2 *P[0]* := S_0
3 *P'* := az összes *P*-beli molekulából egy elemi művelettel (1.5.11.5.2) elérhető molekula és *P* elemei
4 rendezzük *P'*-t *C* szerint csökkenő sorrendbe és az *M.* elem utáni tagokat eldobjuk
5 **if** *P' = P* **then goto** 8. **endif**
6 *P := P'*
7 **goto** 3.
8 **end**

2.6. MolDQN

Ez a modell Q-tanulást használ a policy indirekt megtanulására.

Q-tanulás

A Q-függvény a kumulatív jutalom várható értéke az adott policy-vel megkapható trajektóriák felett, feltéve az állapotot és a választott cselekvést:

$$Q^{\pi}(s,a) = E_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a]$$

Az optimális Q-függvényt *Q**-al jelöljük:

$$Q^{*}(s,a) = \max_{\pi} E_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a]$$

Q-tanulásnál a Q* függvényt szeretnénk megtanulni, ebből indirekten megismerhetjük az optimális policy-t, mégpedig úgy, hogy minden *s* állapotban azt az *a** cselekvést választjuk, melyre:

$$a^{*} = \underset{a}{\operatorname{argmax}}\, Q^{*}(s,a)$$

Q-tanulás legegyszerűbb formája a Q-tábla, aminek a sorai az állapotok, oszlopai a cselekvések. A táblázatot iteratívan frissítjük a környezetből mintavételezett információk alapján:

$$Q^{\text{Új}}(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha(r_t + \gamma \max_{a} Q(s_{t+1},a) - Q(s_t,a_t))$$

ahol α a tanulási ráta, és γ a leszámolási tényező. A pirossal jelölt tagot *TD* targetnek hívjuk, az egész zárójelben levő kifejezést pedig *Temporal Difference*-nek *(TD)*.

A mi molekula építés Markov Döntési Folyamatunkban az állapotok és a cselekvések tere rendkívül nagy, teljesen kezelhetetlenné válik ezzel a táblázatos módszerrel. A táblázat helyett viszont, ha egy neurális hálózattal közelítjük a Q-függvényt, sokkal egyszerűbb dolgunk van.

A hálózatot itt is gradiens-süllyesztéssel tanítjuk, a *TD targettől* való négyzetes eltérést használva veszteségfüggvényként:

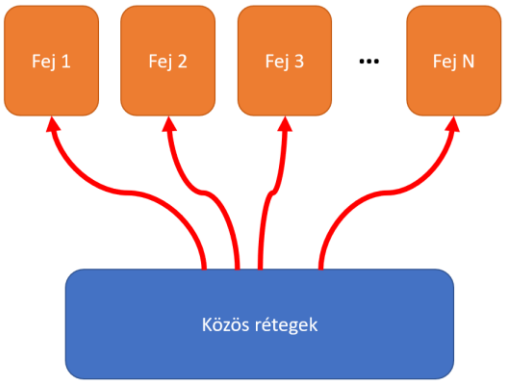
$$\theta_{t+1} \leftarrow \theta_t - \alpha\nabla_{\theta}\left[(r_t + \gamma \max_{a} Q(s_{t+1},a) - Q(s_t,a_t))^2\right]$$

Ha tanításkor ugyanazt a Q-függvényt használjuk a veszteségfüggvény mindkét tagjának (a *TD*-target és Q(s,a)) kiszámításához, akkor a tanítás nagyon instabil lehet, mivel folyamatosan változik a függvény, amit a hálózatnak tanulnia kéne (*TD-target*). Ennek elkerülése érdekében két neurális hálózatot is használunk, egyet a target kiszámítására (target hálózat) és egyet az aktuális Q függvény közelítésére (online hálózat). Tanítás során N iteráció után a target hálózatba bemásoljuk az online hálózat súlyait. Ez stabilabb tanításhoz vezethet azáltal, hogy fixen tartja a target függvényt (legalábbis N ideig).

Két módszert is alkalmazunk annak érdekében, hogy a modellünket felfedezésre késztessük.

Az első ilyen módszer igen egyszerű: minden lépésben ε valószínűséggel véletlenszerű cselekvést választunk. Ezzel szabályozhatjuk a modellünk mohóságát, hogy milyen valószínűséggel válasszon olyan cselekvéseket, ami biztos (ámbár lehet, hogy szuboptimális) jó állapotokba vezetnek. A módszer neve ε-greedy [9].

A *Bootstrapped-DQN* [4] megközelítés komplexebb, viszont sokkal kifinomultabb. Ez a módszer annyival jobb az ε-greedy-nél, hogy a konzisztensen rossz cselekvéseket elkerüli, emellett továbbra is felfedezést biztosít. A megközelítés lényege, hogy nem egy, hanem N darab Q-függvényt tanulunk egyszerre (ezeket fejeknek hívjuk), ezek között véletlenszerűen (akár a legjobban teljesítőket nagyobb valószínűséggel) választunk és azt használjuk a cselekvés kiválasztására. A módszer hatékonyságán javíthat, ha az N darab Q-függvényt approximáló hálózatok osztoznak az első pár rétegen, ezáltal az azokban a rétegekben realizált működést mindegyik tudja használni (nem szükséges újra megtanulni).



10. ábra - Bootstrapped-DQN architektúra

2.6.1. Tanítás

A modell bemenetének a molekulák Morgan fingerprint-jét választottam, emellet megkaphatja a hátralevő lépések számát is. A hátralevő lépések száma azért lehet fontos információ a policy-nk számára, mert lehet, hogy már nem érné el a messzebbi, nagyobb jutalmú molekulát, viszont a hátralevő lépésekbe beleférhet egy kevésbé (de az összes még elérhetőhöz képest jobb) optimumba.

A tanításnál használt ε-greedy módszernél általában nem konstans, hanem „időfüggő” ε függvényt használunk. Az általam használt ε függvény:

$$\varepsilon(t) = \varepsilon(t-1) * \alpha \quad \text{, ahol } \alpha \text{ a lecsengési tényező}$$

ε értékét 1-ről indítom és a tanulás előrehaladtával egyre mohóbbá válik az ágens, nagyobb valószínűséggel cselekszik a tanult policy szerint.

MOL-DQN

---

input:
E: tanítási iterációk száma
M: kívánt jelölthalmaz mérete
k: bejárandó trajektóriák max. hossza
S_0: kezdeti állapot
C: célfüggvény
B: egy tanítási iterációban mintavételezendő trajektóriák száma
Q: Q-függvényt approximáló neurális hálózatok hiperparaméterei
H: bootstrap fejek száma
L_H: bootstrap fejek hossza (rétegekben)
γ: leszámolási tényező
output:
P: jelölthalmaz

1 *P* := üres, *M* méretű tömb
2 *T* := üres lista // replay buffer, korábbi tranzíciók tárolására
3 neurális hálózatok inicializálása *Q*, *H*, L_H alapján (első rétegek közösek)
4 **while** *e* < *E* **do**
5 *b* := 0
6 **while** b < *B* **do**
7 Q_i := véletlenszerűen választott fej
8 *s* := S_0, *z* := *0*, *b* := *b* + 1
9 **if** $p < \varepsilon(e), p \in U(0, 1)$ **then**
10 a := véletlenszerű cselekvés
11 **else**
12 $a := \underset{a \in A(s)}{\operatorname{argmax}} Q_i(s, a)$
13 **endif**
14 *s'* kiválasztása a alapján
15 *(s, a, s', r)* hozzáadása *T*-hez, ahol *r* = *C(s')* vagy *C(s')* – *C(s)*
16 **if** *z* < *k* **and** *s'* ≠ *s* **then goto** 7. **endif**
17 *P*-be beszúrjuk *s'*-t úgy, hogy *P* rendezve legyen *C* szerint
18 **endwhile**
19 *T*-ből kiveszünk egy *(s, a, s', r)* négyest
20 kiválasztunk egy Q_j Q-függvényt
21 kiszámoljuk *TD*-t: $(r + \gamma \max_a Q_j(s', a) - Q_j(s, a))^2$
22 *TD*-n visszaterjesztjük a gradienst és frissítjük Q_j súlyait
23 **endwhile**
24 **end**

3. Összehasonlítás

3.1. Validity

A validity azt adja meg, hogy a jelölthalmazt előállító modell kimenetei milyen arányban érvényes molekulák. Ez az arány határozza meg a már betanított modell mintavételezésének a gyorsaságát illetve mintahatékonyságát. A vizsgált modellek közül egyedül a SeqGAN-nál érdemes erről beszélni, hiszen a többi modell a felépítéséből adódóan 100%-os validity arányt ér el (a molekula építés MDP-ben érvényes molekulából csak érvényes molekulába juthatunk).

A SeqGAN modell validity arányára 0.9% és 1.74% közötti értékeket mértem.

3.2. Futásidő és memóriahsználat

Minden algoritmust ugyanazon a hardware-en futtattam:

- CPU: Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz
- RAM: 252 Gb
- GPU: Nvidia TITAN Xp 12 Gb

A felhasznált memóriát minden algoritmusnál a jelölthalmaz mérete (N) határozza meg. Mivel az algoritmusok egy fix méretű tömböt használnak a jelölteknek, abba rendezetten szúrnak be, a felhasznált memória O(N) méretű marad mindvégig.

A SeqGAN és a MolDQN a használt neurális hálózatokatis a memóriában tárolják, viszont ezeknek a mérete elhanyagolható (és nem is függ N-től) a nagyobb jelölthalmaz méretéhez képest.

A komplexebb modellek futásideje elérheti az egy napot egy erősebb hardware-en, a referencia módszerek futásideje pár perces nagyságrendű. A guacamol adathalmazból kiválasztani az N darab legjobb molekulát valamilyen target szerint párhuzamosítással is egy nap volt (30 szállal próbáltam).

3.3. Legjobb megtalált molekulák

Két táblázatban mutatom be a módszerek teljesítményét 3 target szerint. Az első táblázatban a „naív” (vagy egyszerűbb) módszereket hasonlítom össze, ezeknek a maximumát tekintem baseline-nak minden target szerint.

Modell	**QED**	**5*QED - SA**	**penalized logp**
Random Walk	0.473	0.06	-1.98
Naiv Mohó	0.391	1.42	0.56
Sztochasztikus mohó	**0.732**	**1.56**	**2.11**

Látható, hogy ugyanakkora mintaszám mellett (jelen esetben 1000) a sztochasztikus mohó sokkal jobban teljesít, mint a random walk.

Modell	QED	5*QED-SA	penalized logp
Guacamol (referencia)	0.948	3.32	20.46
Baseline	0.732	1.56	2.11
Top-M mohó	**0.948**	**3.01**	**58.19**
SeqGAN	0.909	2.15	14.62
MolDQN	0.927	1.92	9.81

Azt láthatjuk, hogy az algoritmusok közül a Top-M teljesít messze a legjobban és azáltal, hogy a működése determinisztikus, minden alkalommal meg fogja találni ugyanazt a legmagasabb target-értékkel rendelkező molekulákat, a többi – sztochasztikus - módszerrel ellentétben.

Ebben az összehasonlításban csak a jelölthalmazok legjobb 1 molekuláját vetem össze, viszont ebből nem kapunk képet a teljes jelölthalmazok jóságáról.

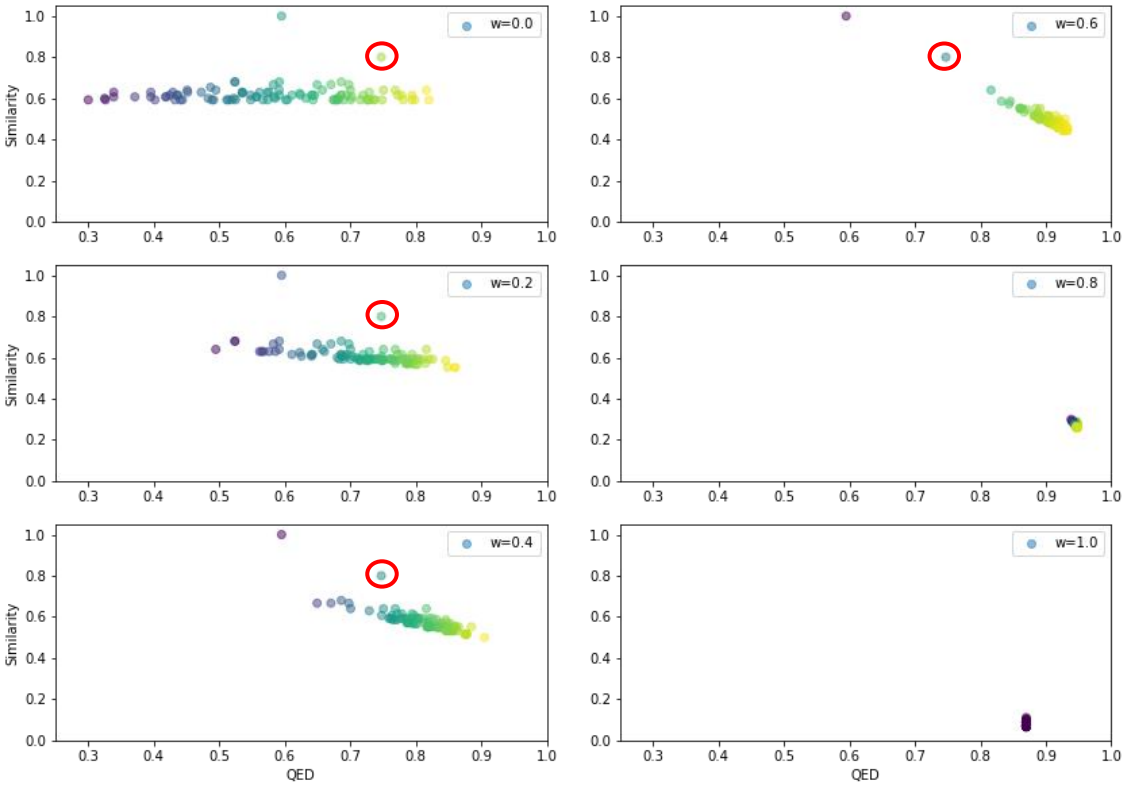
3.4. Jelölthalmazok összehasonlítása

A jelölthalmazok összehasonlítására a *C(mol) = w * QED + (1-w) * SIM(m0, mol)* targetet választottam, ahol *w* értékét interpoláltam 0 és 1 között.

Ebben fejezetben feltételezem, hogy a QED és SA értékek megfelelnek a valóságnak (tehát, hogy ha egy molekula QED értéke magasabb egy másikénál, akkor az a valóságban is nagyobb valószínűséggel jó gyógyszerhatóanyag, illetve, hogy ha az SA értéke kisebb, mint egy másiknak, akkor tényleg könnyebb előállítani).

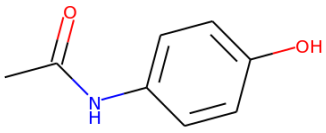
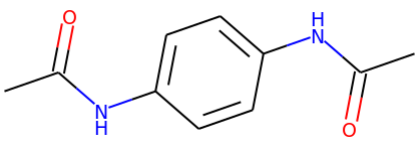
3.4.1. Paracetamol

Az alábbi ábrán láthatjuk, hogy *w* megválasztásával milyen jelölthalmazokat kapunk a Top-M mohó algoritmussal (a jelölthalmaz mérete 100 molekula):



11. ábra - Hasonlóság a paracetamol molekulához és QED értékek *w* függvényében – Top-M mohó algoritmus. Pirossal jelölt molekula a leghasonlóbb a Paracetamolhoz.

Az algoritmusnak sikerült megtalálnia a Paracetamolt minden $w < 0.6$ esetben (az a pont, aminek a Tanimoto hasonlósága – y tengely – 1.0). A pirossal jelölt molekula hasonlít legjobban a Paracetamolhoz, emellett lényegesen jobb is a QED értéke és alacsonyabb a SA értéke (könnyebben előállítható és gyógyszerszerűbb), mint a Paracetamol.

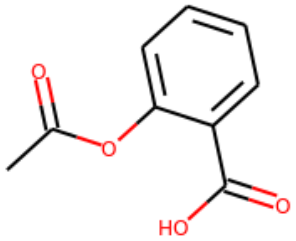
	Paracetamol	**pirossal jelölt molekula**
molekula		
QED	0.595	0.747
SA	1.41	1.37

Látható, hogy *w*-t [0.4;0.6] intervallumból megválasztva lesznek még valamiképp hasonlóak a megtalált molekulák és kapunk egész magas QED értékekets.

3.4.2. Aszpirin

Optimalizálás a *C(mol) = w * QED + (1-w) * SIM(m0, mol)*, targetet használtam, ahol m0 az Aszpirin. Az itt bemutatott ábrákon a színátmenet a molekulák *C* értékeit jelölik, a sötétlilától a sárgáig (sárgább – jobb).

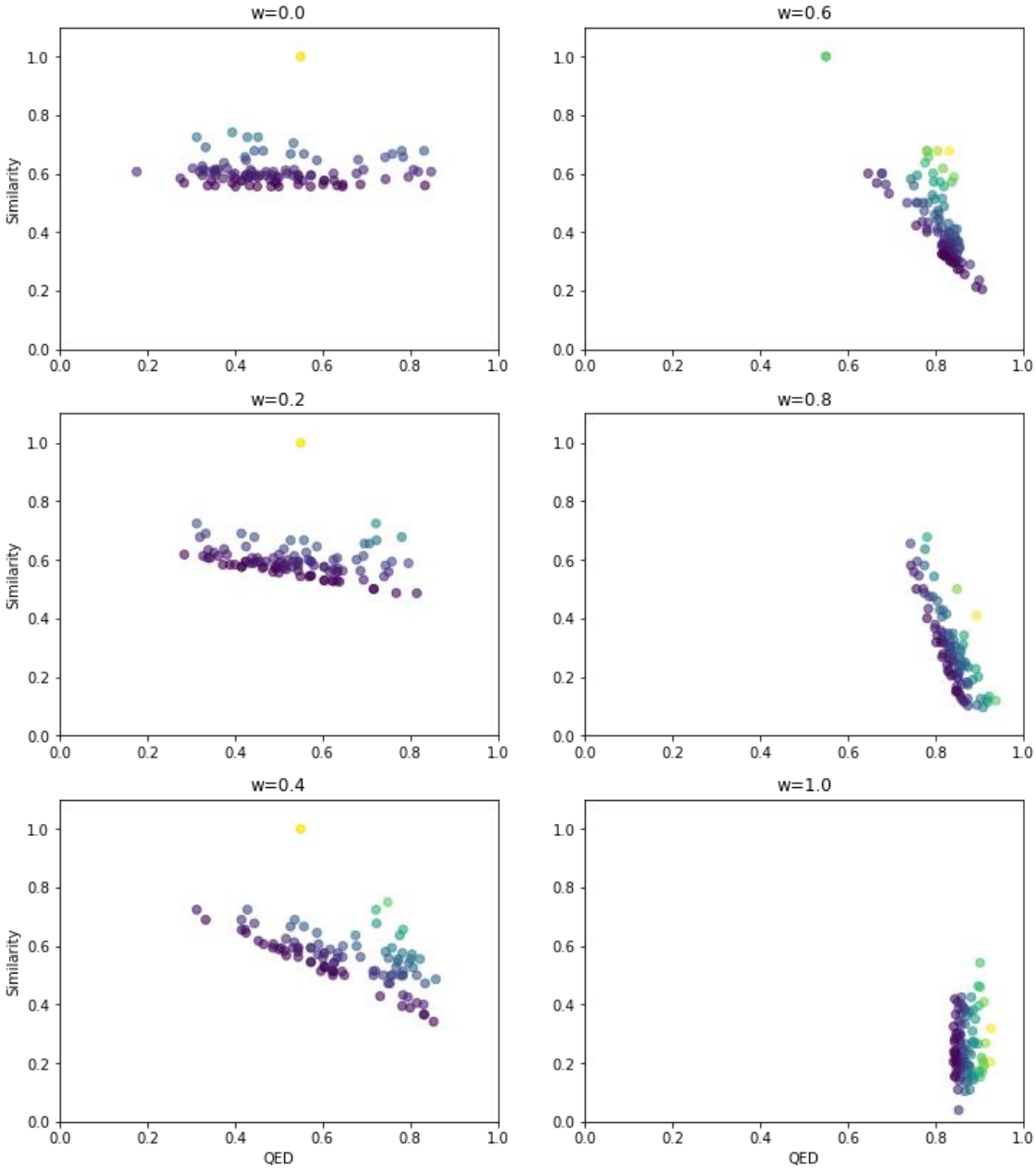
Érdemes megfigyelni a jelölthalmazok szélén (11. 13. 14. és 15. ábrák) az „éles” levágást. ezt a jelenséget az okozza, hogy a jelölthalmazokba csak a legjobb M molekula kerül be. Az egyenes, amit a vágásra illeszthető merőleges arra az irányra, ami felé *C* a legjobban nő (természetesen ez *w* függvényében változik, az egyenes meredekségét ez határozza meg).



12. ábra - Aszpirin molekula

3.4.2.1. MolDQN

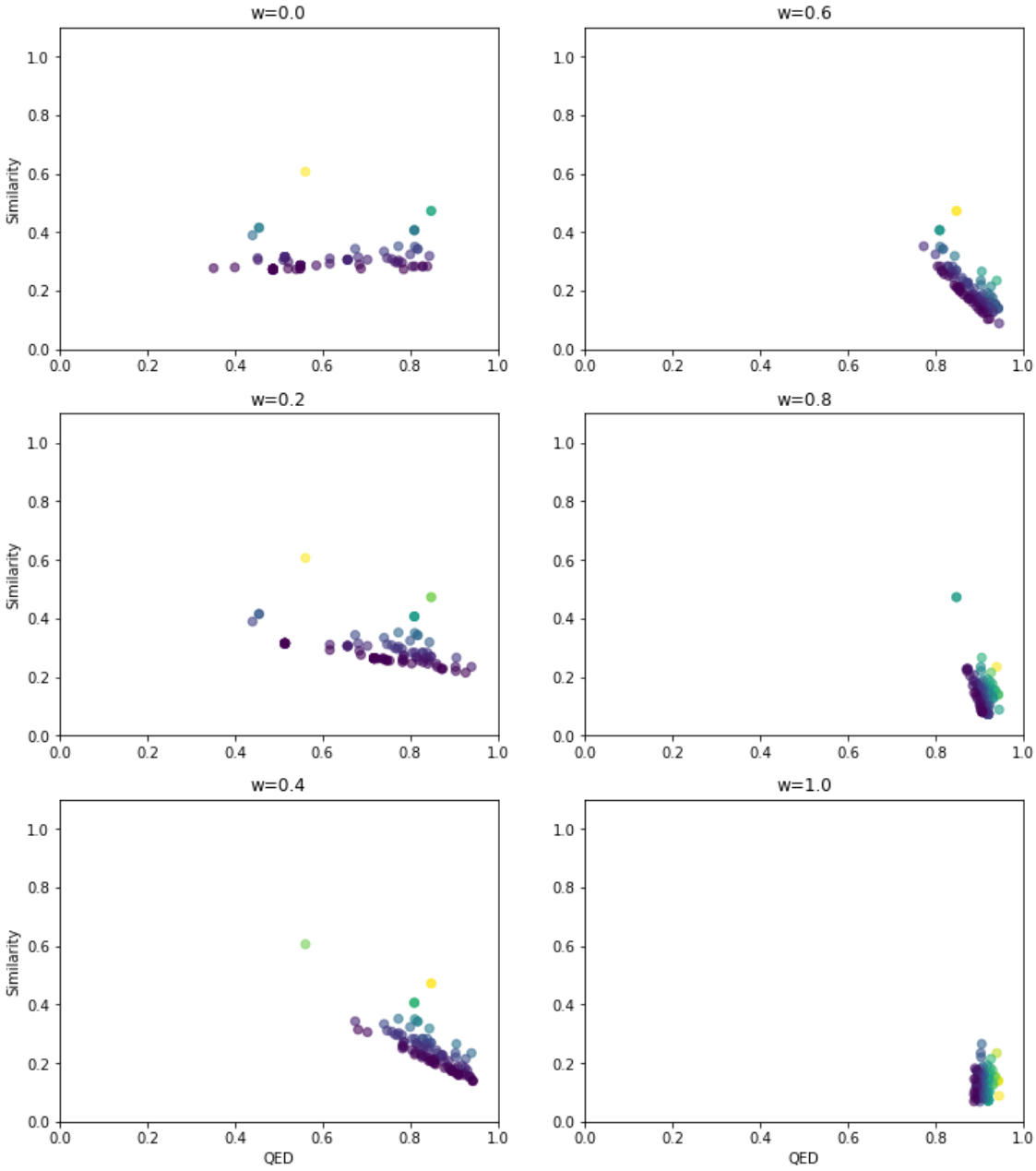
Az ábrán bemutatott jelölthalmazokat a MolDQN modell segítségével állítottam elő, a modellt 1000 trajektórián keresztül tanítottam, és 35 bootstrap fejet használtam.



13. ábra – Hasonlóság és QED szerinti optimalizáció, kettő közötti súlyparaméter interpolációjával – MolDQN modellel előállítva

3.4.2.2. SeqGAN

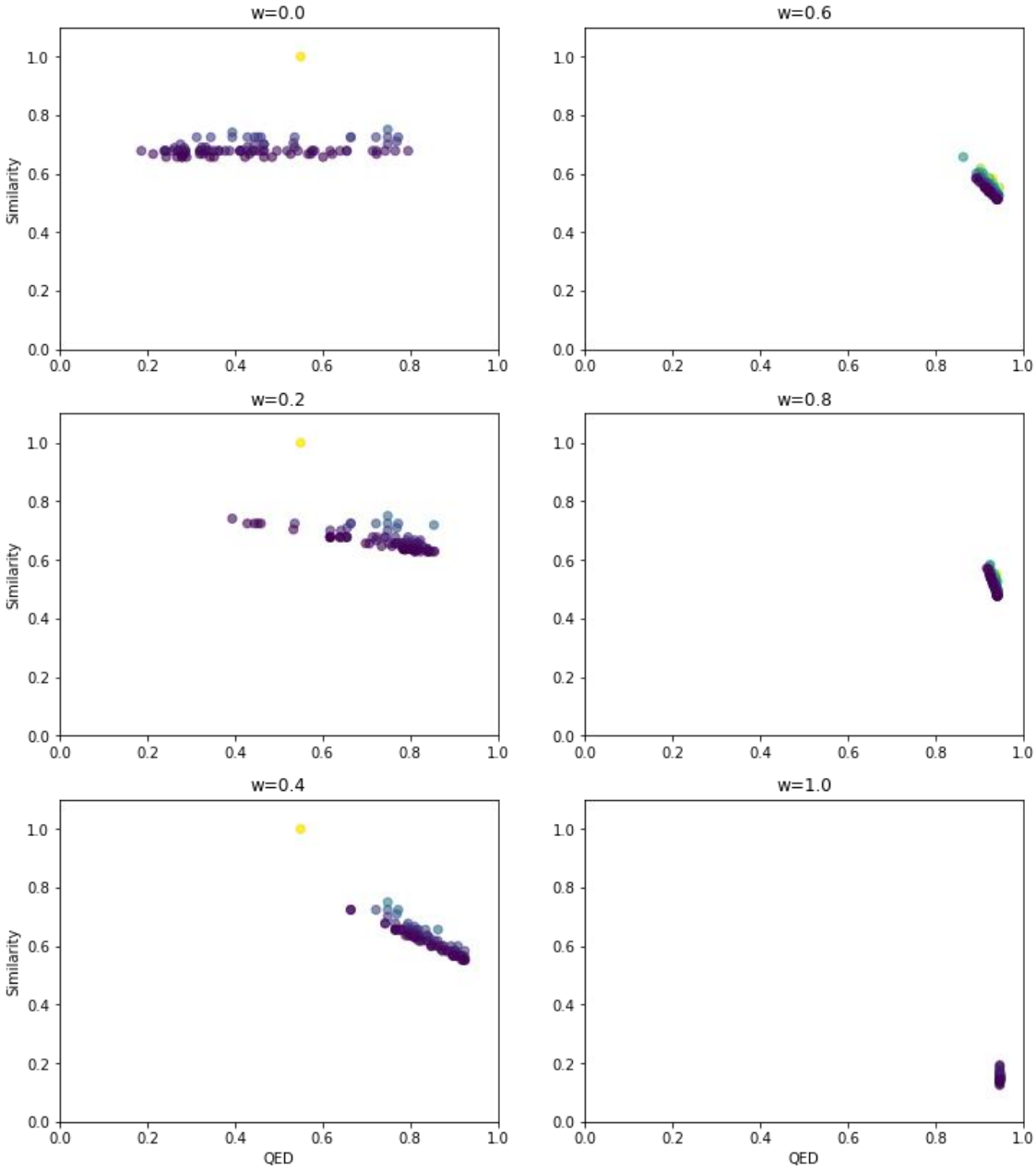
A SeqGAN modellt előtanítással 50000 epochon keresztül tanítottam 128-as batch mérettel, majd ezt követően 1000 epochon keresztül a *C* target szerint.



14. ábra - Hasonlóság és QED szerinti optimalizáció, kettő közötti súlyparaméter interpolációjával – SeqGAN modellel előállítva

Az aszpirin molekulát nem sikerült megtalálnia a SeqGAN-nak a másik két módszerrel ellentétben, viszont QED terén hasonlóan jó eredményeket hozott.

3.4.2.3. Top-M mohó



15. ábra - Hasonlóság és QED szerinti optimalizáció, kettő közötti súlyparaméter interpolációjával – Top-M mohó modellel előállítva

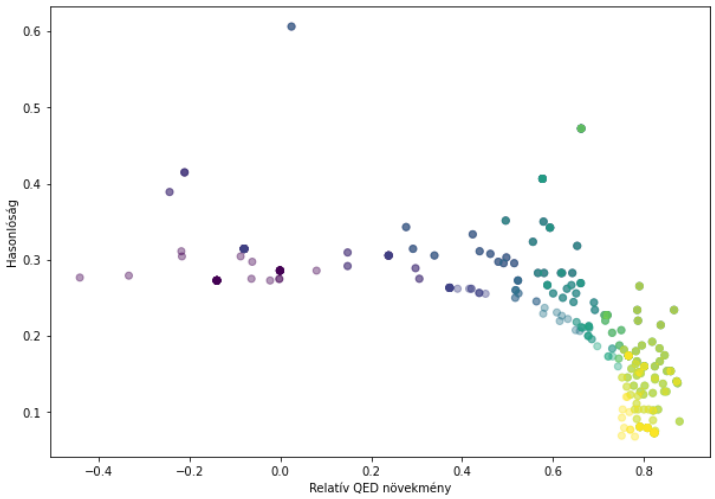
Láthatjuk, hogy ez a módszer sokkal kisebb szórással talált molekulákat, mint az előző kettő, a jelölthalmazban minden molekula nagyon jónak számít a target szerint.

3.4.2.4. Értékelés

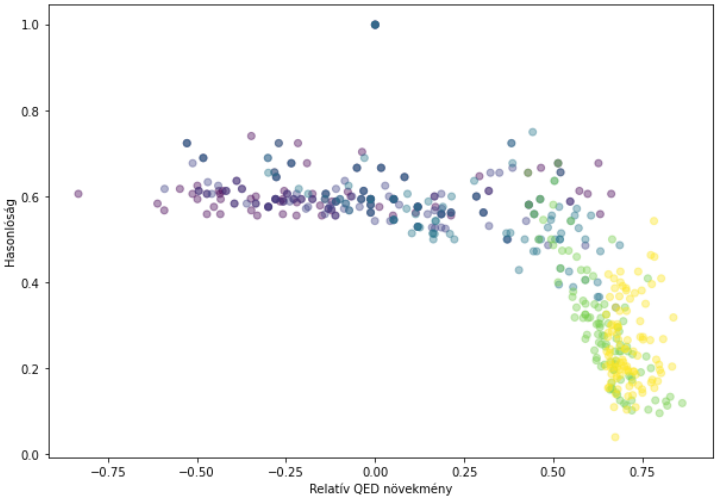
Az jelölthalmazok jóságának összehasonlításhoz bevezetem a Relatív QED Növekmény (imp_{rel}) [3] fogalmát:

$$imp_{rel} = \frac{QED(m) - QED(m_0)}{1 - QED(m_0)}$$

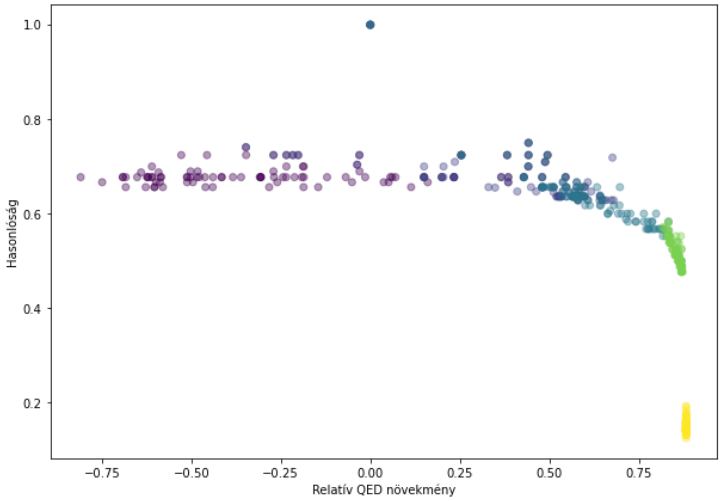
Ez a szám az *m* molekula QED növekménye és a lehetséges legnagyobb QED növekmény arányát adja meg. A pozitív értékek azt jelentik, hogy sikerült javítani a molekulán QED terén, a negatív azt, hogy romlott a QED a m_0-hoz képest. A modelleinknél azt szeretnénk elérni, hogy ez a szám minél nagyobb legyen amellett, hogy a generált molekulák hasonlóak legyenek az m_0 molekulához. Az alábbi ábrákon megmutatom a relatív QED növekményt a molekulák hasonlóságával együtt. Az ábrákon sötétlila-sárga színátmenettel jelölöm az optimalizálásnál használt *w* értékeket (0 a sötétlila, 1 a sárga).



16. ábra – Relatív QED növekmény és Tanimoto hasonlóság összefüggése – SeqGAN modellel előállított jelölthalmazra



17. ábra - Relatív QED növekmény és Tanimoto hasonlóság összefüggése – MolDQN modellel előállított jelölthalmazra



18. ábra - Relatív QED növekmény és Tanimoto hasonlóság összefüggése – Top-M mohó modellel előállított jelölthalmazra

Minden esetben azt tapasztalhatjuk, hogy egy ideig nagyjából vízszintes egyenest követnek a molekulák, majd következik egy „letörés”, attól kezdve rohamosan csökken a hasonlóság az eredeti molekulához. A jelölthalmaz annál jobb, minél később van benne a törés és a vízszintes szakasz minél magasabban van, hiszen ez azt jelenti, hogy hasonlóbb molekulákkal nagyobb javulást értünk el, mint a másik módszerek esetében.

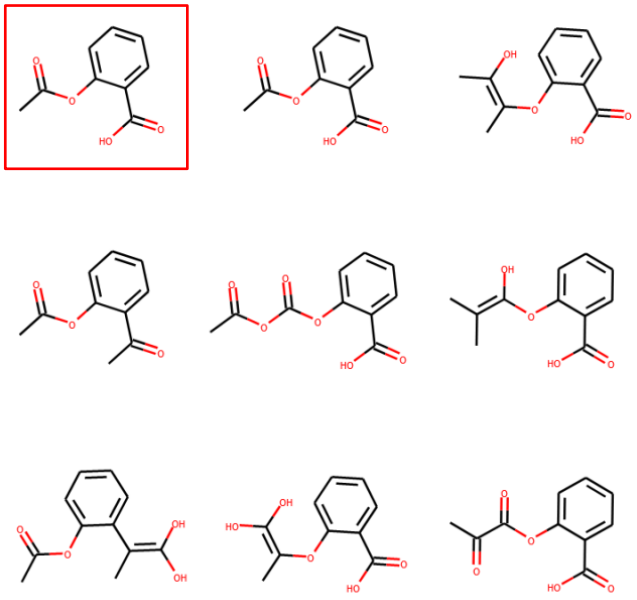
Itt is látható, hogy az 0.5 körüli *w* megválasztásával kaphatjuk a legjobb növekményű, de még hasonló molekulákat.

módszer	**egyenes magassága**	**letörés helye (imp_{rel})**
SeqGAN	0.3	0.5
MolDQN	0.6	0.4
Top-M mohó	**0.7**	**0.6**

A teljes jelölthalmazt illetően is a Top-M mohó algoritmus teljesít a legjobban.

3.4.3. Pár molekula a jelölthalmazból

Az alábbi képen a Top-M mohó algoritmus által megtalált leghasonlóbb 9 molekulát láthatjuk az aszpirinhez:



19. ábra - Leghasonlóbb 9 molekula az aszpirinhez, pirossal keretezett maga az aszpirin

4. A jó target fontossága

4.1. Probléma

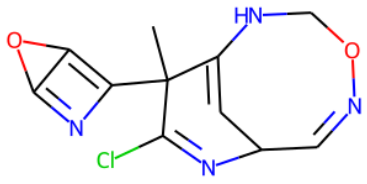
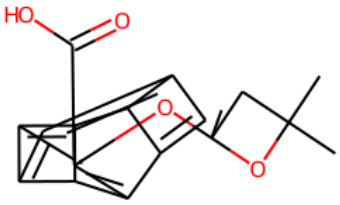
Ahogy azt a 3.3. fejezet második táblázatában is láthattuk, a *penalized logp* targetre való optimalizálásnál a Top-M mohónak sikerült elérnie egy rendkívül magas értéket a többihez képest. Az algoritmus megtalálta a célfüggvényben rejlő hibát: egy olyan molekulát épített, amiben rendkívül hosszú kénláncok vannak (további kénatomok beszúrásával lehet még tovább növelni a target értékét, a keresést leállítottam 100 iteráció után). Kijelenthetjük, hogy a target „nem jó” olyan értelemben, hogy egy ilyen (valóságban értelmetlen/használhatatlan/kezelhetetlen) molekulát jónak minősít.

20. ábra - A legjobb megtalált molekula a *penalized logp* target szerint – Top-M mohó algoritmussal

LogP	**SA**	**Cycle Count**	**penalized logp**
65.18	6.99	0	58.19

A target hibája az SA-t és LogP-t számítást végző kódrészletek okozzák, egyes abnormálisan nagy molekulák „kiakasztják” ezeket a könyvtárakat.

Ez nem csak ennél a targetnél fordulhat elő, sikerült pl. a QED-et úgy maximalizálni, hogy igen érdekes molekulákat találtak:

MolDQN	
Top-M mohó	

Nem szükséges mélyebb kémia ismeret annak megállapításához, hogy ezek a molekulák nagy valószínűséggel nem állíthatóak elő, illetve instabilak.

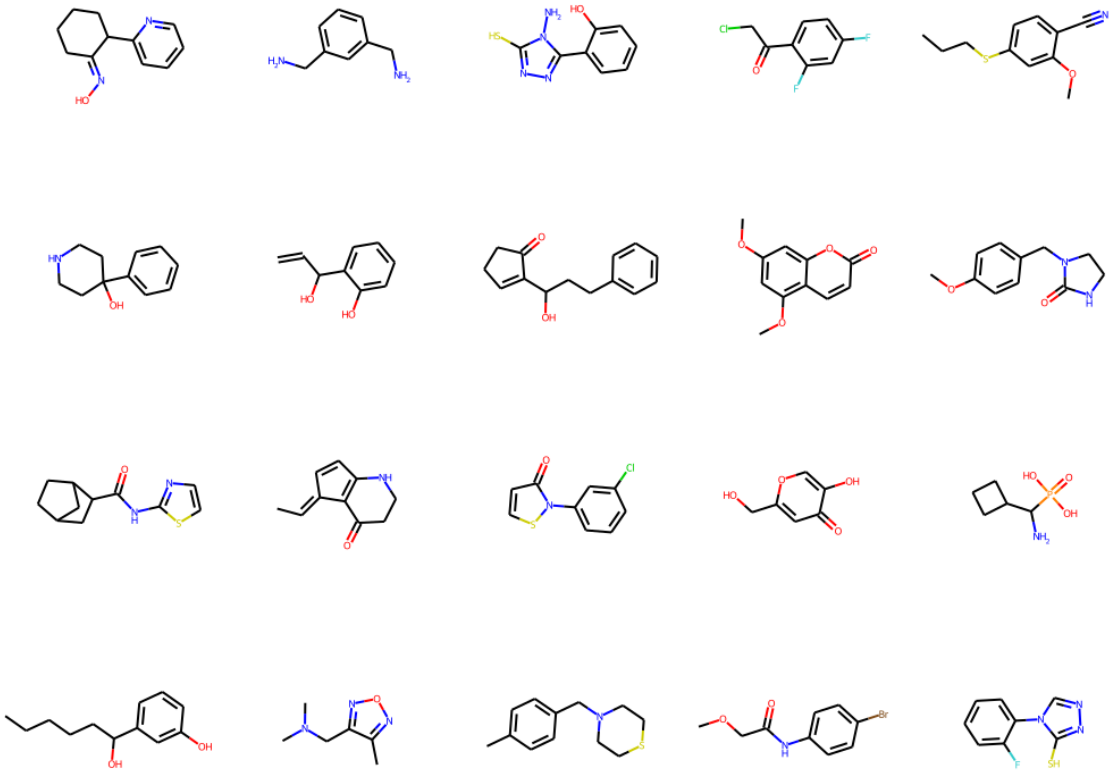
4.2. Megoldás

Mivel tudjuk, hogy nagy molekulák esetén áll fent ez a probléma, a targeteket tudjuk úgy módosítani, hogy legyen benne egy tömeget vagy atomszámot büntető tag, vagy a target mellé bevezetünk valamiféle kényszert (pl. az MDP-ben nem engedélyezzük azokat a lépéseket, amik túl nagy molekulákhoz vezetnének). Ezáltal elkerülhetjük az ilyen szélsőséges eseteket.

Egy másik lehetőség, hogy bevesszük egy ismert hatóanyaghoz való hasonlóságot a célfüggvénybe, ahogy láthattuk, ha kellően nagy súlyt adunk a hasonlóság tagnak, akkor nem fog olyan nagy molekulákat találni, amik kellően hasonlóak lennének.

Léteznek olyan módszerek, amik pontosan tudják szimulálni az egyes molekulák közötti interakciót. Egy ilyen módszer a Molecular Dynamics Szimuláció [17]. Ennek segítségével pontosan modellezhető, hogy hogyan fog egy adott molekula adott környezetben viselkedni. Célfüggvénynek megválaszthatjuk a szimuláció során kapott viselkedés jóságát, ami egy reprezentatív érték lesz, ellentétben az itt ismertetett, közelítéseket használó targetekkel. Ez a módszer viszont rendkívül számításigényes, nem használható az algoritmusainkkal együtt, hiszen azok több ezer molekulán tanulnak. Ennyi molekulára a szimuláció futtatása nem kivitelezhető.

4.3. Pár példa molekula a kiegészített targettel



21. ábra – Példák a generált molekulákra

5. Konklúzió

A mérések eredménye alapján azt tudom mondani, hogy nem feltétlenül szükséges a jelölthalmaz előállítását Gépi Tanulással megközelíteni, az igen egyszerű Top-M algoritmus is versenyképes azok teljesítményével, vagy akár jobban is teljesít.

Összességében, ha molekula optimalizálásra kéne ajánlanom egy módszert az itt megvizsgáltak közül, a Top-M mohó-t vagy a MolDQN-t választanám. Mindkettő nagyon jó jelölthalmazokat tud előállítani órákon belül.

Valós helyzetben ezen modellek még messze nem tudják kiváltani az emberi szakértőket, habár ezt főleg a targetek gyenge minőségét okozza véleményem szerint. A jövőben, ha elő tudnánk állni olyan targetekkel, amik sokkal pontosabban reprezentálják a molekulák jóságát, akkor lenne érdemes az itt bemutatott modelleket akár valós gyógyszerkutatásoknál is önállóan alkalmazni.

Köszönetnyilvánítás

A Méréstechnikai és Információs Rendszerek Tanszéknek köszönöm, hogy a méréseimhez futtatásához használhattam a tanszék egyik számítógépét.

Irodalomjegyzék

[1] Bellman, R. A markovian decision process. *J. Math. Mech.* 679–684 (1957).

[2] Yu, L., Zhang, W., Wang, J., & Yu, Y. (2017, February). Seqgan: Sequence generative adversarial nets with policy gradient. In Thirty-first AAAI conference on artificial intelligence.

[3] Zhou, Z., Kearnes, S., Li, L., Zare, R. N., & Riley, P. (2019). Optimization of molecules via deep reinforcement learning. Scientific reports, 9(1), 1-10.

[4] Osband, I., Blundell, C., Pritzel, A., & Van Roy, B. (2016). Deep exploration via bootstrapped DQN. In Advances in neural information processing systems (pp. 4026-4034).

[5] Fu, R., Zhang, Z., & Li, L. (2016, November). Using LSTM and GRU neural network methods for traffic flow prediction. In 2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC) (pp. 324-328). IEEE.

[6] Schulman, J. (2016). Optimizing expectations: From deep reinforcement learning to stochastic computation graphs (Doctoral dissertation, UC Berkeley).

[7] Gao, J., Shen, Y., Liu, J., Ito, M., & Shiratori, N. (2017). Adaptive traffic signal control: Deep reinforcement learning algorithm with experience replay and target network. arXiv preprint arXiv:1705.02755.

[8] Blaschke, T., Olivecrona, M., Engkvist, O., Bajorath, J., & Chen, H. (2018). Application of generative autoencoder in de novo molecular design. *Molecular informatics*, *37*(1-2), 1700123.

[9] Tokic, M., & Palm, G. (2011, October). Value-difference based exploration: adaptive control between epsilon-greedy and softmax. In *Annual Conference on Artificial Intelligence* (pp. 335-346). Springer, Berlin, Heidelberg.

[10] Rdkit. Rdkit: Open-source cheminformatics software, http://www.rdkit.org/, https://github.com/rdkit/rdkit (2016).

[11] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., ... & Lerer, A. (2017). Automatic differentiation in pytorch.

[12] Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace.

[13] „The 0.85 myth" - Maggiora, G., Vogt, M., Stumpfe, D., & Bajorath, J. (2014). Molecular similarity in medicinal chemistry: miniperspective. Journal of medicinal chemistry, 57(8), 3186-3204.

[14] RL taxonómia kép - https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

[15] GRU cella kép - https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

[16] Oliphant, T. E. (2006). *A guide to NumPy* (Vol. 1). Trelgol Publishing USA.

[17] Andersen, H. C. (1980). Molecular dynamics simulations at constant pressure and/or temperature. *The Journal of chemical physics*, *72*(4), 2384-2393.

Függelék

Multinomiális eloszlás

A multinomiális eloszlás a binomiális eloszlás egy kiterjesztése. Ugyanúgy, mint a binomiális eloszlásnál itt is egymástól független kísérletek eredményét modellezzük, a különbség az, hogy itt nem bináris a kísérletek kimenete, hanem *k* darab kimenetel lehetséges, és minden kimenetelnek van egy valószínűsége (ezen valószínűségek összege 1, hiszen egyik kimenetel mindenképpen bekövetkezik).

Softmax függvény

A softmax függvénnyel egy *N* elemű $\underline{\boldsymbol{x}}$ valós vektorból készíthetünk egy valószínűségi eloszlást. Az így kapott eloszlás *i*. elemének valószínűsége arányos lesz az $exp(\underline{\boldsymbol{x}}_i)$-vel. A függvényt a következő módon definiáljuk:

$$softmax(\boldsymbol{x})_i = \frac{e^{\boldsymbol{x}_i}}{\sum_{j=0}^{N} e^{\boldsymbol{x}_j}}$$

Sigmoid függvény

Gyakran használjuk neurális hálózatokban aktivációs függvényként, így definiáljuk:

$$sigmoid(x) = \frac{e^x}{e^x + 1}$$