Eötvös Loránd Tudományegyetem

Természettudományi Kar

Illés Kincső Boriska

Szubmoduláris maximalizálás a gépi tanulásban

BSc Alkalmazott Matematikus Szakdolgozat

Témavezető:

Bérczi-Kovács Erika

Operációkutatási Tanszék

Budapest, 2019

Köszönetnyilvánítás

Szeretnék köszönetet mondani témavezetőmnek, Bérczi-Kovács Erikának a téma kiválasztásában és a cikkek összeszedésében nyújtott segítségéért.
Továbbá Sagmeister Ádámnak, amiért idejét rám szánva sokszor azon dolgozott, hogy nekem segítsen egy-egy adott szövegrészletben, struktúrában.
Hálával tartozom családomnak,barátaimnak a sok bíztatásért, a támaszért, amit nyújtottak és nyújtanak, Kedvesemnek külön, hisz miatta csillant fel az én szemem is a gépi tanulással kapcsolatos kérdésekben.

Tartalomjegyzék

1. fejezet

Bevezetés

Szakdolgozatomban a fő téma szubmoduláris halmazfüggvények maximalizálása mohó algoritmuson keresztül a gépi tanulásban. Részhalmaz kiválasztó algoritmusokról írok, melyek korlát nélkül és korlátozott módon is megvalósulhatnak.
A 2. fejezetben magát a neurális hálót, annak felépítését és működését mutatom be, mely háló alapja a gépi tanulásnak.
A 3. fejezet elején ismertetem a szubmoduláris függvény definícióját és jellemzőit. Kimondom a problémát, melyre megoldásokat keresünk a maximalizálás mohó megközelítésével. A korlát nélküli szubmoduláris maximalizálás (KNSM) esetében egy determinisztikus algoritmust és egy véletlenített algoritmust alkalmazunk erre a feladatra.
A legfontosabb rész a 4. fejezetben található, ahol a korábbi fejezetekben leírtakat összerakva kiderül, miként használható mindez az adatokból történő tanuláshoz, kísérleteket is bemutatva. Részletezem a Differenciálható Mohó Hálózat (DMH) felhasználását a FEVER feladatra, amely adathalmazt állítások igazolásához, annak bizonyító mondatainak visszakereséséhez használjuk.

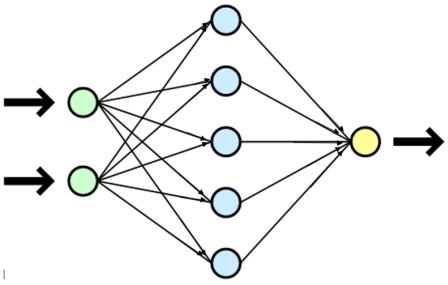
2. fejezet

A neurális háló

2.1. A neurális háló felépítése

2.1.1. Definíció. *A neuronok a háló alapegységei, melyek tulajdonképpen* $\mathbb{R}^n \to \mathbb{R}$ *függvények. Mindegyike rendelkezik egy aktivációs értékkel.Az* $a_i \in \mathbb{R}^+$ *aktivációs érték egy 0 és 1 közötti szám, mely az adott neuron aktivitását fejezi ki a bemeneti adatoktól függően. A neuron inputja a korábbi rétegek neuronjainak outputjaiból áll össze.*

2.1.2. Definíció. *A neurális háló struktúrálisan egy* $G(V, E, w_i)$ *irányított, élsúlyozott gráf (*$w_i \in \mathbb{R}, V = \{neuronok\}$*), melyben rétegekbe rendezett mesterséges neuronok kommunikálnak egymással nemlineáris aktivációs függvényeken keresztül. Három strukturálisan elkülöníthető részből áll: bemeneti réteg, rejtett rétegek, kimeneti réteg. Minden réteg valamennyi neuronja össze van kötve a korábbi réteg összes neuronjával. Egy réteg neuronjainak aktivációs értékei meghatározzák a következő réteg aktivációs értékeit. A teljes neurális háló egy függvény, melynek inputja a bemeneti réteg neuronjainak inputja, és outputja a kimeneti réteg neuronjainak outputja.*



A bemeneti réteg az input neuronokból áll, melyek nem végeznek információfeldolgozást, csupán a hálózat bemeneteinek a következő réteg bemeneteihez való eljuttatása a feladatuk.

A rejtett rétegek száma különböző esetekben eltérhet. Az első rejtett réteg minden egyes neuronja össze van kötve a bemeneti réteg összes neuronjával. Ezen neuronok aktivációs értékét egy súlyozott összeg adja, az előző réteg neuronjainak aktivációs értékeiből és az összekötő élek súlyaiból együttesen, ezután egy $F : \mathbb{R} \to \mathbb{R}$ aktivációs függvényen keresztül fut az eredmény: $F(w_1a_1 + ... + w_na_n)$. Ilyen aktivációs függvény például a Sigmoid függvény ($F(x) = \frac{1}{1+e^{-x}}$), melynek output értékei -1 és 1 közt mozognak, így korlátozva, stabilan tartva a számításokat a neurális hálóban. A leggyakrabban használt a ReLU függvény ($F(x) = \max(0, X)$). A b eltolásérték a súlyozott összeghez hozzáadott érték, mellyel szabályozza az adott neuron aktivitását, inaktivitását: $\sigma(w_1a_1 + ... + w_na_n + b)$. Ebből a számításból adódnak a következő réteg neuronjai. Az ezt követő réteg neuronjai hasonlóan jönnek létre, inputjuk az előző réteg neuronjainak outputja.

A bemeneti réteg neuronjainak aktivációs értékeit vektorba, az egyes rétegek neuronjai közti élek súlyait mátrixba, az eltolásértékeket vektorba rendezve kapjuk a megfelelő aktivációs értékeket.

A kimeneti réteg azon neuronja lesz a neurális háló válasza az adott problémára, melynek aktivációs értéke a legnagyobb.

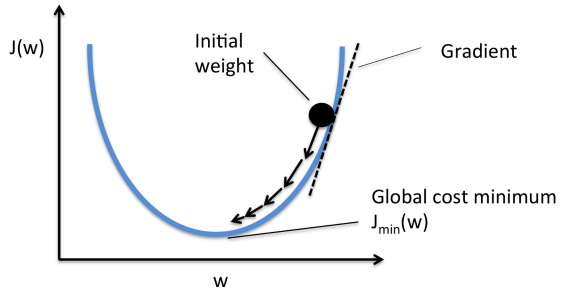
2.2. A neurális háló működése

A neurális háló egy adott problémára általában nem a teljesen igaz választ adja. Ennek kiküszöbölésére az ötlet a tanulás és tesztelés folyamata, ezáltal a megfelelő súlyokat, eltolásértékeket megtalálni, a hibákat a lehető legkisebbre szorítani.

2.2.1. Gradiens módszer

A hibák lecsökkentéséhez egy bizonyos C költségfüggvényt kell minimalizálni. Egy adott inputra gyakran használt költségfüggvény a helyes megoldás értékeinek és a kapott megoldás értékeinek különbség négyzetösszege. Ez a költség kicsi, ha a hálózat megfelelően működik. A költségfüggvény inputja a súlyok és az eltolásértékek, outputja egy szám, maga a költség, ami kifejezi a bemeneti adatok „jóság”-át. Ahhoz, hogy a költségfüggvényt hatékonyan tudjuk minimalizálni, érdemes először megvizsgálni, hogy mely irányba

kellene elmozdulni, hogy a legjobban csökkenjen a költség. A negatív gradiens irányába lépkedve általában gyorsan találhatjuk meg a minimumot.



A negatív gradiens vektor minden egyes komponense két dolgot hordoz magában: azt, hogy melyik irányba kellene változtatni az input vektoron, és milyen mértékben, tehát minden egyes súly és eltolásérték fontosságát írja le.

2.2.2. Visszaterjesztés

Ez a legfontosabb algoritmus a neurális hálók tanulásában, a gradiens kiszámolásához. Amíg a neurális háló még nem tanult eleget, véletlenszerű output értékekből indul ki. A kimeneti réteg neuronjainak aktivációs értékei megadják, hogy melyeket kellene csökkenteni és melyeket növelni, hogy az inputnak megfelelő eredményt kapjuk. Az értékek megváltoztatása az eredmény befolyásolása szempontjából nem egyenrangú. Tekintsük a megfelelő megoldás aktivációs értékét, ami mint korábbról tudjuk egy súlyozott összeg a korábbi rétegek aktivációs értékeiből. Ezt az értéket kellene megkapni, a felhasznált adatok megfelelő módosításával.

Az aktivációs értékek növelését háromféleképpen is megtehetjük: növelhetjük az eltolásértéket, a súlyokat vagy megváltoztathatjuk a korábbi rétegek aktivációs értékeit.

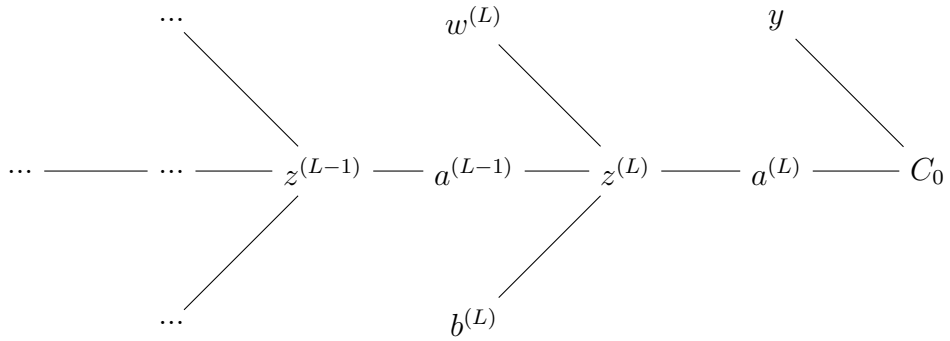
1. Súlyok változtatásával: a korábbi réteg „legfehérebb", legaktívabb neuronjainak van a legnagyobb hatása a súlyozott összegre. A legnagyobb súlynövelések és a legerősebb kapcsolatok a legaktívabb neuronok és azon neuron közt vannak, amit aktívabbá kívánunk tenni.

2. A korábbi réteg aktivációs értékeinek megváltoztatásával: a pozitív súlyú kapcsolatokat fehérebbé (aktívabbá), a negatív súlyú kapcsolatokat szürkébbé (inaktívabbá) kell tenni, hogy aktívabbá tegye a kívánt neuront.

A többi output neuront kevésbé aktívvá kellene tenni.

2.2.3. Visszaterjesztés kalkulusa

Tekintsünk egy egyszerűbb esetet a kalkulus megértéséhez. Álljon minden réteg egy darab neuronból. Ezek közül most nézzük az utolsó két réteg neuronját. Az utolsó réteg neuronját jelölje $a^{(L)}$, az ezt megelőző réteg neuronját $a^{(L-1)}$ és így tovább. A kívánt output aktivációs értéke 1 vagy 0, ezt a neuront jelölje y. A mostani példára nézve a költség $C_0 = (a^{(L)} - y)^2$, de ez csak egy tréningező példára értve. Emlékezzünk, hogy hogyan kapjuk meg a neuronok aktivációs értékeit: $a^{(L)} = \sigma(w^{(L)}a^{(L-1)} + b^{(L)})$ (Sigmoid vagy ReLU aktivációs függvénnyel). Jelölje a σ argumentumában lévő összeget ezesetben $z^{(L)}$. Tehát, 3 dolog kell z kiszámolásához (w, a, b), amelyből aztán megkapjuk a következő a értéket, mely az y-al együttvéve kiadja a költséget végül. Ezt iterálva az $a^{(L-1)}$ hasonlóan számolódik.



A cél ezt megkapni:

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}}.$$

A szorzat 3 tényezője együttesen adja meg, hogy C mennyire érzékeny a $w^{(L)}$-ben történő kis változásokra. Ha tetszőleges számú tréningező példára nézzük, akkor $\frac{\partial C}{\partial w^{(L)}} = \frac{1}{n}\sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w^{(L)}}$. Fontos, hogy ez a kiszámolt érték csak egy érték a gradiens vektorból. Ahhoz, hogy megkapjuk mennyire érzékeny az eltolásértékek változtatására, ehhez csak a láncszabályban a $w^{(L)}$ helyett $b^{(L)}$-t kell írni. Az aktivációs értékek esetében is csak egy tényezőt kell kicserélni, hogy az erre vonatkozó érzékenységét vizsgálhassuk. Ezt a láncszabály ötletet hátrafelé iteráljuk a korábbi rétegek felé. Innen jön a visszaterjesztés elnevezés.

Ha nagyobb neurális hálót tekintünk, ahol egy rétegben több neuron van, akkor egy alsó indexszel hasonló jelöléseket alkalmazhatunk a számításokban, mint a fentiek.

2.3. Kézzel írott számjegy felismerése

Hogyan ismeri fel a neurális háló a különféle kézzel írott számjegyeket?
Bemenetként, ha kap egy 28x28 pixeles képet egy kézzel írott számról, ez a 784 neuron alkotja az ehhez tartozó bemeneti réteget. Mindegyik rendelkezik a megfelelő aktivációs értékkel, melyek bevilágítják a bemeneti réteg összes neuronját ennek megfelelően. Ez a 0 és 1 közti szám fejezi ki a neuron állapotát, szürkeárnyalatot ad meg (1 a fehér, 0 a fekete).

28x28 pixeles kép egy 7-es számjegyről

A kimeneti réteget 10 neuron alkotja, amelyek a 0 - 9 számjegyeket jelölik. A kimeneti réteg neuronjainak aktivációs értékei mutatják, hogy a neurális háló válasza mennyiben tér el az eredetileg beadott képen lévő számjegytől. A rejtett rétegek száma sok esetben más, itt most vegyünk két rejtett réteget. Az első rejtett réteg minden egyes neuronja kapcsolatban áll a bemeneti réteg 784 neuronjával. Mindegyik kapcsolathoz tartozik egy súly, és minden neuronhoz a rejtett rétegből tartozik egy eltolásérték.

Miért van szükség ilyen rétegekbe rendezett rendszerre?
A neurális háló egy számjegyet részekre darabolva ismer fel, azaz hurkokra, vonalakra tagolja a számjegyek alakját. A hurkokat és vonalakat még kisebb részekre, alkomponensekre osztja. Az első rejtett réteget egy konkrét inputtól függően ezen alkomponenseknek megfelelően világítja be, a második rejtett réteget pedig a már ezekből alkotott nagyobb komponenseknek megfelelően.

3. fejezet

Approximációs algoritmusok szubmoduláris halmazfüggvények maximalizálására

3.1. Szubmoduláris halmazfüggvények

3.1.1. Definíció. *Adott egy N véges halmaz és egy $z : \mathbb{R} \to 2^N$ halmazfüggvény. Ekkor z szubmoduláris, ha*

$$z(A) + z(B) \geq z(A \cap B) + z(A \cup B) \;\; \forall A, B \subseteq N.$$

A szubmodularitás kifejezhető a csökkenő hozam jelölésével is. Jelölje egy $e \in V$ elem hozzáadásának hasznát $\triangle_f^+(e|S) = f(S \cup \{e\}) - f(S)$, és az $e \in S \subseteq V$ elem kivételének hasznát $\triangle_f^-(e|S) = f(S \backslash \{e\}) - f(S)$. Ezzel definiálva a szubmodularitást kapjuk, hogy

$$\triangle_f^+(e|A) \;\geq\; \triangle_f^+(e|B) \;\; \forall A \subseteq B \subset N, e \notin B.$$

3.2. Maximalizálás mohó megközelítése

Legyen K egy pozitív egész, z pedig egy szubmoduláris függvény egy N halmazon. A probléma, melyre megoldásokat keresünk:

$$\max_{S\subseteq N}\{z(S) : |S| \leq K, z(S) \text{ szubmoduláris}\}. \tag{3.1}$$

Az egyik legkézenfekvőbb megközelítés a mohó algoritmus, melyben kiindulunk az üres halmazból és minden egyes lépésben hozzávesszük azt az elemet, mely a legnagyobbat növel a z értékén.

A mohó algoritmus halmazfüggvényekre:

Kezdőállapot: Legyen $S^0{=}\varnothing$, $N^0{=}N$ és $t{=}1$.

A t. *iteráció:* válasszuk $i(t) \in N^{t-1}$, melyre $\triangle_z^+(i(t)|S^{t-1}){=}\max_{i\in N^{t-1}}(S^{t-1})$.

Legyen $\rho_{t-1} = \triangle_z^+(i(t)|S^{t-1})$.

1.lépés: Ha $\rho_{t-1} \leq 0$, megáll az S^{K^*} halmazzal, $K^* = $ t-1 $< K$.

Ha $\rho_{t-1} > 0$, legyen $S^t = S^{t-1} \cup \{i(t)\}$ és $N^t = N^{t-1}$ - $\{i(t)\}$.

2.lépés: Ha $t = K$, megáll az S^{K^*} halmazzal, $K^* = K$. Egyébként $t \to t{+}1$.

3.2.1. Megjegyzés. Jelölés: $Z :=$ az optimális megoldás értéke 3.1-re, és $Z^G :=$ egyéb megoldás értéke 3.1-re a mohó módszerrel.

3.2.2. Megjegyzés. $Z^G = z(\varnothing) + \rho_0 + \cdots + \rho_{K^*-1}$, $K^* \leq K$.

Tegyük fel, hogy $K^* \geq 1$, ezzel kizárva a triviális $Z = Z^G = z(\varnothing)$ esetet.

3.2.3. Megjegyzés. Jelölés: Legyen $C(\theta)$ azon szubmoduláris halmazfüggvényeknek az osztálya, melyekre teljesül $\triangle_z^+(j|S) \geq$ -θ, $\forall\ \ S{\subset}\ N$, $j \in N$-S.

3.2.4. Állítás. *Tegyük fel, hogy $z \in C(\theta)$, $\theta \geq 0$, és a mohó algoritmus megáll K^* lépés után, ekkor a megfelelő $\{\rho_j\}_{j=0}^{K^*-1}$ kielégi a következőt:*

$$Z \leq z(\varnothing) + \sum_{i=0}^{t-1}\rho_i + K\rho_t + t\theta,\ t = 0, ..., K^* - 1$$

$$\textit{és} \tag{3.2}$$

$$Z \leq z(\varnothing) + \sum_{i=0}^{K^*-1}\rho_i + K^*\theta,\ \textit{ha } K^* < K.$$

Bizonyítás. Felhasználjuk hozzá ezt az egyenlőtlenséget:

$z(T) \leq z(S) + \sum_{j \in T-S} \triangle_z^+(j|S) + |S - T|\ \theta.$

Legyen T a 3.1 probléma optimális megoldása, S a mohó algoritmus t iterációja után generált S^t halmaz, és tudjuk:

$Z{=}z(T), \quad \triangle_z^+(j|S^t) \leq \rho_t, \quad \rho_t \geq 0, \quad |S^t - T| \leq t, \quad \theta \geq 0, \quad |T - S^t| \leq K$

és $z(S^t){=}z(\varnothing){+}\ \sum_{i=0}^{t-1} \rho_i,$

így kapjuk

$Z \leq z(\varnothing) + \sum_{i=0}^{t-1} \rho_i + K\rho_t + t\theta, t = 0, ..., K^* - 1,$

továbbá,

ha $K^*{<}K, \quad S{=}S^{K^*}$-t véve adódik $Z \leq z(\varnothing){+}\ \sum_{i=0}^{K^*-1} \rho_i{+}\ K^*\theta,\ \rho_{K^*} \leq 0.$ $\square$

$\theta{=}0$-t véve, a 3.2.4 Állítás második egyenlőtlenségéből adódik:

3.2.5. Állítás. *Ha a mohó algoritmust monoton növő z függvénnyel alkalmazzuk a* 3.1 *problémára és az algoritmus megáll $K^*{<}K$ lépés után, a mohó megoldás optimális.*

A 3.2 első egyenlőtlenségéből $t{=}0$-ra kapjuk, hogy

3.2.6. Állítás. *A mohó algoritmust a* 3.1 *problémára alkalmazva*

$$\frac{Z - Z^G}{Z - z(\varnothing)} \leq \frac{K-1}{K}.$$

Bizonyítás. A 3.2 egyenlőtlenségből $t{=}0$-ra kapjuk, hogy Z-$z(\varnothing) \leq K\ \rho_0 \leq K(Z^G$-$z(\varnothing))$, ami azzal ekvivalens, hogy $\frac{Z-Z^G}{Z-z(\varnothing)} \leq \frac{K-1}{K}$. $\square$

3.3. Korlát nélküli szubmoduláris maximalizálás (KNSM)

Tekintjük a korlát nélküli szubmoduláris maximalizálás (KNSM) problémát [2], melyben adott egy f nemnegatív szubmoduláris függvény ($f : 2^N \to \mathbb{R}^+$). Találni szeretnénk egy $S \subseteq N$ részhalmazt, mely maximalizálja $f(S)$-t.

3.3.1. Megjegyzés. Adott egy S halmaz és egy u elem, jelöljük az $S \cup \{u\}$ kifejezést S+u -val, az $S \setminus \{u\}$ -t S-u -val.

3.3.2. Tétel. (Buchbinder et al. [2]) *Létezik egy determinisztikus lineáris idejű (1\3)-approximáló algoritmus a korlát nélküli szubmoduláris maximalizálás problémára.*

3.3.3. Tétel. (Buchbinder et al. [2]) *Létezik egy véletlenített lineáris idejű (1\2)-approximáló algoritmus a korlát nélküli szubmoduláris maximalizálás problémára.*

A bemutatandó algoritmusok a mohó megközelítésen alapszanak.
Tekintsünk egy nemnegatív szubmoduláris f függvényt. Vizsgáljuk meg f komplementerét, jelölje $\overline{f}$, a következőképpen definiálva: $\overline{f}(S) := f(N \setminus S)$, $\forall\ S \subseteq N$.
Mivel f szubmoduláris, így $\overline{f}$ is szubmoduláris. Továbbá, adott egy optimális megoldás $OPT \subseteq N$ a KNSM-ra f inputtal, $N \setminus OPT$ egy optimális megoldás $\overline{f}$-hez, és mindkét megoldásnak pontosan ugyanaz az értéke.
Tekintsük a korábban leírt mohó algoritmust. f esetében az üres halmazból indulunk ki és iteratívan elemeket adunk hozzá mohó megválasztással. $\overline{f}$ esetében fordítva működik a mohó algoritmus, az N halmazból indul ki és minden egyes lépésben kivesz belőle elemet. De mindkét algoritmus „megbukik”.
Ezért f-t és $\overline{f}$-t együttesen tekintjük. Két megoldással indulunk el: $\varnothing$ és N. Az algoritmus tekint egy elemet és meghatározza, hogy az adott elemet hozzá kellene-e adni az első megoldáshoz, vagy eltávolítani a második megoldásból. Végül a két megoldás megegyezik majd, ez lesz az algoritmus outputja.

3.3.1. Determinisztikus lineáris idejű (1\3)-approximáló algoritmus

Jelölje az N halmaz elemeit u_1, ... ,u_n. Az algoritmusban n iteráció van és két halmazt tart fent végig, legyenek ezek az i. iterációban: X_i és Y_i. Kezdetben legyen $X_0 = \varnothing$ és $Y_0 = N$. Az i. iterációban vagy hozzáadja u_i -t X_{i-1} -hez, vagy kiveszi u_i -t az Y_{i-1} -ből. Ennek eldöntése mohó módón zajlik a két opció marginális hasznától függően. Az n. iteráció után azt kapjuk, hogy $X_n = Y_n$, mely az algoritmus outputja.

Algorithm 1 Determinisztikus KNSM (f,N)

$X_0 \leftarrow \varnothing$, $Y_0 \leftarrow N$.
for i=1 to n **do**
    $a_i \leftarrow f(X_{i-1} + u_i)$ - $f(X_{i-1})$.
    $b_i \leftarrow f(Y_{i-1}$ - $u_i)$ - $f(Y_{i-1})$.
    if $a_i \geq b_i$ **then**
        $X_i \leftarrow X_{i-1} + u_i$.
        $Y_i \leftarrow Y_{i-1}$.
    else
        $X_i \leftarrow X_{i-1}$.
        $Y_i \leftarrow Y_{i-1}$ - u_i.
    end if
end for
return X_n (vagy ekvivalensen Y_n)

3.3.4. Lemma. $\forall\, 1 \leq i \leq n : a_i + b_i \geq 0$.

Bizonyítás. Tudjuk, hogy $(X_{i-1} + u_i) \cup (Y_{i-1} - u_i) = Y_{i-1}$ és $(X_{i-1} + u_i) \cap (Y_{i-1} - u_i) = X_{i-1}$. Ezekhez hozzávéve a szubmodularitást kapjuk, hogy

$$\begin{aligned} a_i + b_i &= [f(X_{i-1} + u_i) - f(X_{i-1})] + [f(Y_{i-1} - u_i) - f(Y_{i-1})] = \\ &= [f(X_{i-1} + u_i) + f(Y_{i-1} - u_i)] - [f(X_{i-1}) + f(Y_{i-1})] \geq 0. \end{aligned}$$

□

A továbbiakban a 3.3.2 Tétel kerül bizonyításra.
A bizonyítás fő ötlete, hogy korlátozzuk a teljes értékveszteséget ezen sorozat mentén: $f(OPT_0), ..., f(OPT_n)$, ahol $OPT_i := (OPT \cup X_i) \cap Y_i$, azaz OPT_i megegyezik X_i-vel az

első i elemen, és OPT-al az utolsó $n-i$ elemen. Így $OPT_0 = OPT$, és az algoritmus outputja $OPT_n = X_n = Y_n$.
Szükség lesz a következő lemmára, mely felülről korlátozza a veszteséget a sorozat két egymást követő eleme között.

3.3.5. Lemma. $\forall\ 1 \leq i \leq n : f(OPT_{i-1}) - f(OPT_i) \leq [f(X_i) - f(X_{i-1})] + [f(Y_i) - f(Y_{i-1})]$.

Bizonyítás. Tegyük fel $a_i \geq b_i$, azaz $X_i \leftarrow X_{i-1} + \{u_i\}, Y_i \leftarrow Y_{i-1}$. Ekkor $OPT_i = OPT_{i-1} + u_i$ és $Y_i = Y_{i-1}$. Így kapjuk a megfelelő egyenlőtlenséget:
$f(OPT_{i-1}) - f(OPT_{i-1} + u_i) \leq f(X_i) - f(X_{i-1}) = a_i$. Két eset lehetséges.
Ha $u_i \in OPT$, az utolsó egyenlőtlenség bal oldala 0, így az kell csak, hogy nemnegatív az a_i. És ez igaz, mert $a_i + b_i \geq 0$ a korábbi lemmából adódóan, és feltettük, hogy $a_i \geq b_i$.
Ha $u_i \notin OPT$, akkor $u_i \notin OPT_{i-1}$-nek sem, így $f(OPT_{i-1}) - f(OPT_{i-1} + u_i) \leq f(Y_{i-1} - u_i) - f(Y_{i-1}) = b_i \leq a_i$. □

Bizonyítás. A 3.3.2 Tétel pedig következik ebből a lemmából, hiszen $\forall\ 1 \leq i \leq n$ -re összegezve a lemmában lévő egyenlőtlenséget azt kapjuk, hogy

$$\sum_{i=1}^{n}[f(OPT_{i-1}) - f(OPT_i)] \leq \sum_{i=1}^{n}[f(X_i) - f(X_{i-1})] + \sum_{i=1}^{n}[f(Y_i) - f(Y_{i-1})].$$

Ez egy teleszkópikus összeg. Így adódik: $f(OPT_0) - f(OPT_n) \leq [f(X_n) - f(X_0)] + [f(Y_n) - f(Y_0)] \leq f(X_n) + f(Y_n)$. Az OPT_0 és OPT_n definíciójával kapjuk, hogy $f(X_n) = f(Y_n) \geq f(OPT)/3$.
□

3.3.2. Véletlenített lineáris idejű $(1\backslash 2)$-approximáló algoritmus

Az 1-es algoritmus összehasonlítja az a_i, b_i marginális hasznokat. Az összehasonlítás következtében az algoritmus hoz egy mohó determinisztikus döntést, hogy az adott u_i elemet beveszi vagy kizárja az outputból. A véletlen algoritmus esetében az egyes elemek bevál asztása véletlenszerű.
$\forall\ 1 \leq i \leq n$, X_i és Y_i véletlen változók, melyek az i. iteráció végén kapott két megoldás elemeinek halmaza.

Algorithm 2 Véletlenített KNSM (f,N)

1: $X_0 \leftarrow \varnothing$, $Y_0 \leftarrow N$.
2: **for** i=1 to n **do**
3: $a_i \leftarrow f(X_{i-1} + u_i)$ - $f(X_{i-1})$.
4: $b_i \leftarrow f(Y_{i-1}$ - $u_i)$ - $f(Y_{i-1})$.
5: $a_i{}' \leftarrow \max\{a_i, 0\}$,
6: $b_i{}' \leftarrow \max\{b_i, 0\}$.
7: **withprobability** $a_i{}'/(a_i{}' + b_i{}')^*$ do: $X_i \leftarrow X_{i-1} + u_i$, $Y_i \leftarrow Y_{i-1}$.
8: **else** (a komplementer valószínűséggel $b_i{}'/(a_i{}'+b_i{}')$) do: $X_i \leftarrow X_{i-1}$, $Y_i \leftarrow Y_{i-1}-u_i$.
9: **end for**
10: **return** X_n (vagy ekvivalensen Y_n)

*Ha $a_i{}' = b_i{}' = 0$, azt kapjuk, hogy $a_i{}'/(a_i{}' + b_i{}')$=1.

Ezesetben tekintsük a következő sorozatot: $\mathbb{E}[f(OPT_0)], ..., \mathbb{E}[f(OPT_n)]$. A kezdőérték $f(OPT)$, és az algoritmus outputjának várható értékével végződik a sorozat. A következő lemma felülről korlátozza a veszteséget a sorozat bármely két egymást követő tagjára nézve.

3.3.6. Lemma. $\forall\ 1 \leq i \leq n$-re

$$\mathbb{E}[f(OPT_{i-1}) - f(OPT_i)] \leq \frac{1}{2}\,\mathbb{E}[f(X_i) - f(X_{i-1}) + f(Y_i) - f(Y_{i-1})].$$

Bizonyítás.

Ez kielégíti minden $X_{i-1} = S_{i-1}$ eseményre a lemmában szereplő egyenlőtlenséget, ahol $S_{i-1} \subseteq \{u_1, ..., u_{i-1}\}$. Fixáljunk le egy eseményt az S_{i-1} halmaznak megfelelően. A továbbiakban mindent ezen az eseményen feltételezünk. A feltételezés miatt a következő véletlen változók konstansok lesznek:

1. $Y_{i-1} = S_{i-1} \cup \{u_i, ..., u_n\}$.
2. $OPT_{i-1} := (OPT \cup X_{i-1}) \cap Y_{i-1} = S_{i-1} \cup (OPT \cap \{u_i, ..., u_n\}$.
3. a_i és b_i.

A korábbi lemma miatt $a_i + b_i \geq 0$, így nem lehet a_i és b_i is 0-nál kisebb.

Három eset lehetséges.

1. Eset: $(a_i \geq 0, b_i \leq 0)$

Ekkor $\frac{a_i'}{a_i'+b_i'} = 1$, és ekkor $Y_i = Y_{i-1} = S_{i-1} \cup \{u_i, ..., u_n\}$ és $X_i \leftarrow S_{i-1} + u_i$. Ennélfogva $f(Y_i) - f(Y_{i-1}) = 0$. A definíció alapján $OPT_i = OPT_{i-1} + u_i$ adódik.

Már csak azt kell belátni, hogy

$$f(OPT_{i-1}) - f(OPT_{i-1} + u_i) \leq \frac{1}{2}[f(X_i) - f(X_{i-1})] = \frac{a_i}{2}.$$

Ha $u_i \in OPT$, akkor az egyenlőtlenség bal oldala 0, amely triviálisan nem nagyobb, mint a nemnegatív $a_i/2$. Ha $u_i \notin OPT$, akkor

$$f(OPT_{i-1}) - f(OPT_{i-1} + u_i) \leq f(Y_{i-1} - u_i) - f(Y_{i-1}) = b_i \leq 0 \leq a_i/2.$$

Az első egyenlőtlenség a szubmodularitásból következik, mivel $OPT_{i-1} = (OPT \cup X_{i-1}) \cap Y_{i-1} \subseteq Y_{i-1} - u_i$.

2. Eset: $(a_i < 0, b_i \geq 0)$

Ez az eset az előzőhöz hasonlóan adódik.

3. Eset: $(a_i \geq 0, b_i > 0)$

Ebben az esetben $a_i^{'} = a_i,\ b_i^{'} = b_i$. Ebből adódik $\frac{a_i}{a_i+b_i}$ valószínűséggel: $X_i \leftarrow X_{i-1} + u_i$ és $Y_i \leftarrow Y_{i-1}$, míg $\frac{b_i}{b_i+a_i}$ valószínűséggel: $X_i \leftarrow X_{i-1}$ és $Y_i \leftarrow Y_{i-1} - u_i$. Így,

$$\begin{aligned}
&\mathbb{E}[f(X_i) - f(X_{i-1}) + f(Y_i) - f(Y_{i-1})] = \\
&= \frac{a_i}{a_i + b_i}[f(X_{i-1} + u_i) - f(X_{i-1}) + f(Y_{i-1}) - f(Y_{i-1})] + \\
&+ \frac{b_i}{b_i + a_i}[f(X_{i-1}) - f(X_{i-1}) + f(Y_{i-1} - u_i) - f(Y_{i-1})] = \\
&= \frac{a_i}{a_i + b_i}[f(X_{i-1} + u_i) - f(X_{i-1})] + \frac{b_i}{b_i + a_i}[f(Y_{i-1} - u_i) - f(Y_{i-1})] = \frac{{a_i}^2 + {b_i}^2}{a_i + b_i}.
\end{aligned}$$

Az OPT_i definíciójából adódóan, kapunk egy felső becslést a következőre:
$\mathbb{E}[f(OPT_{i-1}) - f(OPT_i)] = \frac{a_i}{a_i+b_i}[f(OPT_{i-1}) - f(OPT_{i-1} + u_i)] + \frac{b_i}{b_i+a_i}[f(OPT_{i-1}) - f(OPT_{i-1} - u_i)] \leq \frac{a_i b_i}{a_i+b_i}$.

Az utolsó egyenlőtlenség két esetből következik. Jegyezzük meg, hogy $u_i \in Y_{i-1}$ és $u_i \notin X_{i-1}$.

Ha $u_i \notin OPT_{i-1}$, akkor az utolsó egyenlőtlenség bal oldalának második kifejezése 0. Valamint, $OPT_{i-1} = (OPT \cup X_{i-1}) \cap Y_{i-1} \subseteq Y_{i-1} - u_i$. Tehát, a szubmodularitás miatt $f(OPT_{i-1}) - f(OPT_{i-1} + u_i) \leq f(Y_{i-1} - u_i) - f(Y_{i-1}) = b_i$.

Ha $u_i \in OPT_{i-1}$, akkor az egyenlőtlenség bal oldalának első kifejezése lesz 0 és $X_{i-1} \subseteq ((OPT \cup X_{i-1}) \cap Y_{i-1}) - u_i = OPT_{i-1} - u_i$. Ekkor $f(OPT_{i-1}) - f(OPT_{i-1} - u_i) \leq f(X_{i-1} + u_i) - f(X_{i-1}) = a_i$.

Mindent összegezve azt kapjuk, hogy

$$\frac{a_i b_i}{a_i + b_i} \leq \frac{1}{2}\left(\frac{a_i^2 + b_i^2}{a_i + b_i}\right).$$

□

Bizonyítás. Ebből a lemmából viszont következik a 3.3.3 Tétel, így $\forall\ 1 \leq i \leq n$-re összegezve a lemmában lévő egyenlőtlenséget kapjuk, hogy

$$\sum_{i=1}^{n} \mathbb{E}[f(OPT_{i-1}) - f(OPT_i)] \leq \frac{1}{2} \sum_{i=1}^{n} \mathbb{E}[f(X_i) - f(X_{i-1}) + f(Y_i) - f(Y_{i-1})].$$

Ez is egy teleszkópikus összeg, így adódik

$$\mathbb{E}[f(OPT_0) - f(OPT_n)] \leq \frac{1}{2}\mathbb{E}[f(X_n) - f(X_0) + f(Y_n) - f(Y_0)] \leq \frac{\mathbb{E}[f(X_n) + f(Y_n)]}{2}.$$

Az OPT_0 és az OPT_n definícióját felhasználva megkapjuk, hogy

$$\mathbb{E}[f(X_n)] = \mathbb{E}[f(Y_n)] \geq \frac{f(OPT)}{2}.$$

□

4. fejezet

Szubmoduláris halmazfüggvények tanulása adatokból

Jellemzően, a szubmoduláris függvény tanulása és a függvény optimalizálása külön kezelendő. Azaz a tanult függvény később maximalizálódik. Bemutatok egy módszert [3], mely együttesen hajtja végre a tanulás és maximalizálás folyamatát. Az algoritmusok a standard mohó algoritmusból [1] és a dupla mohó algoritmusból [2] származnak.

End to end tréningezés

Általában a tanulási rendszerek több szakaszban történő feldolgozást igényelnek, az end to end módszer viszont mindezt helyettesíti egy egyszerű neurális hálóval. Ez a tréningezés fajta gyakran vezet jelentős teljesítmény növekedéshez. Ez a tanulási mód általában elhagyja a közvetítő algoritmusokat és közvetlenül tanulja egy adott probléma megoldását a minta adathalmazból. Ilyen például a beszéd felismerés, melynél a legtöbb módszer nagyon sok szakaszra bontja a feldolgozását egy hangfelvételnek. Amit az end to end tanulás csinál ehelyett a sok lépésből álló lánc helyett az az, hogy veszi inputnak a hangfelvételt és utána közvetlenül kiadja az outputot.

4.0.1. Megjegyzés. Legyen $V = \{e_1, e_2, ...\}$ az elemek alaphalmaza egy rögzített sorrenddel. A V egy részhalmazát szeretnénk kiválasztani. Továbbá, legyen $f : 2^V \to \mathbb{R}$, mely hozzárendel egy valós értékű skalárt V minden részhalmazához.

4.0.1. Problémafelvetés

Tekintjük egy ismeretlen szubmoduláris $f(S|V)$ függvény adatokból való tanulásának problémáját. A tréningező adatok az $f(S|V)$-t maximalizálók $\mathcal{X}$ gyűjteményeként vannak megadva (különféle V_i alaphalmazokra); $\mathcal{X} = \{(V_1, X_1), ..., (V_M, X_M)\}$.
Feltételezzük, hogy az alaphalmaz nemcsak elemek absztrakt halmaza, hanem minden eleme rendelkezik további információkkal. Fontos, hogy a képzési adatok tartalmazhatnak többféle mintát ugyanahhoz az alaphalmazhoz.
Az $\mathcal{X}$-ben lévő maximalizálók mindegyike korlátok nélküli módon közel maximalizálja $f(S|V)$-t, tehát $\forall\ (V, X) \in \mathcal{X}$-re $f(X|V) \approx \max_{S \in V} f(S)$. (Vagy, ha van megadott méretkorlát, akkor $f(X|V) \approx \max_{S \in V, |S| \leq k} f(S|V)$.)
Adott egy $ApproxMax$ algoritmus, mely megközelítőleg maximalizál egy szubmoduláris függvényt korlát nélküli módon, vagy korlátozva. A cél az, hogy tanulni tudjon egy θ-val paraméterezett $h_\theta(S|V)$ halmazfüggvény oly módon, hogy ha ezt maximalizáljuk az $ApproxMax$-al, akkor a visszatérő megoldások megközelítőleg maximalizálják $f(S|V)$-t.

4.1. Mohó maximalizálás tanulása

Két $ApproxMax$ algoritmust készítünk, melyek imputja egy θ-val paraméterezett $h_\theta(.|V)$ függvény, outputja a $h_\theta(.|V)$-t megközelítőleg maximalizálók. Ezek az algoritmusok a standard mohó algoritmuson és a dupla mohó algoritmuson alapulnak. Az algoritmusok indukálnak egy eloszlást az $S \subseteq V$ halmazok felett: $P_{ApproxMax}(S; h_\theta(.|V)) = P(ApproxMax(h_\theta(.|V)) = S)$. A cél megtalálni ezt:

$$\theta^* = arg\max_\theta \sum_{(V,X) \in \chi} logP_{ApproxMax}(X; h_\theta(.|V)).$$

4.1.1. Definíció. *Egy szubmoduláris függvényt maximalizáló approximációs algoritmust akkor nevezünk differenciálhatónak, ha azt választva $ApproxMax$ eljárásunk $P_{ApproxMax}(S; h_\theta(V))$ kifejezése mint θ függvénye θ-ban differenciálható.*

4.1.1. Differenciálható korlát nélküli maximalizálás

Itt mutatom be a valószínűségi $PD^2GREEDY$ algoritmust nemnegatív szubmoduláris halmazfüggvények korlát nélküli maximalizálásához. Ez a dupla mohó algoritmusból származik [2]. Az algoritmus V elemeinek egy fix sorrendjén megy keresztül. Minden iterációban kiszámolja az a_i hasznot az X_{i-1} halmazhoz való i. elem hozzáadáskor és a b_i hasznot az Y_{i-1} halmazból az i. elem eltávolításakor. Összehasonlítja a két hasznot és ehhez mérten hoz egy véletlen döntést.

Algorithm 3 $PD^2GREEDY$: Véletlenített dupla-mohó

$h_\theta : V \to \mathbb{R}_{\geq 0}$
$X_0 \leftarrow \varnothing, Y_0 \leftarrow V.$
for $i = 1, ..., |V|$ **do**
    $a_i = h_\theta(X_{i-1} + e_i) - h_\theta(X_{i-1})$
    $b_i = h_\theta(Y_{i-1} - e_i) - h_\theta(Y_{i-1})$
    if $g(a_i, b_i) \geq U$ **then**
        (U egyenletes eloszlású a [0,1]-en)
        $X_i = X_{i-1} \cup \{e_i\}$
    else
        $Y_i = Y_{i-1} \backslash \{e_i\}$
    end if
end for
return Megközelítőleg maximalizáló $X_{|V|}$

A 3. fejezetben felírt determinisztikus és véletlenített algoritmusok ebből a $PD^2GREEDY$-ből megkaphatóak speciálisan megválasztott $g(a_i, b_i)$ függvénnyel.
Ha $g(a, b) = g_1(a, b) := 1_{a>b}$; ezzel kapjuk meg az (1/3)-approximáló determinisztikus dupla mohó algoritmust pozitív nem monoton szubmoduláris függvények maximalizálásához.
Ha $g(a, b) = g_2(a, b) := \frac{[a]_+}{[a]_+ + [b]_+}$ (ahol $[x]_+ = max\{x, 0\}$); ezzel kapjuk az (1/2)-approximáló véletlenített dupla mohó algoritmust. Az algoritmus indukál egy eloszlást a V részhalmazai felett. Az eloszlás meghatározásához tekintsük az X halmazt reprezentáló x bináris vektort. Ezek után az eloszlást eképpen írhatjuk fel:

$$P_{PD^2GREEDY}(X) = \Pi_{i=1}^{|V|} g(a_i, b_i)^{x_i}(1 - g(a_i, b_i))^{1-x_i}. \tag{4.1}$$

4.1.2. Megjegyzés. Időigény: A 4.1 -hez $4|V|$ függvénykiértékelésre van szükség, de az $f(X_i),\ f(Y_i)$ függvényértékek nyomonkövetésével ez lecsökkenhet $2|V|$-ra.

4.1.3. Megjegyzés. Előfordulhat, hogy negatív értékű függvények is bekerülnek kísérletekbe. Ez azért nem tényleges probléma, mert bármely $f^1(S)$ halmazfüggvény átalakítható egy $f^2(S)$ nemnegatív halmazfügvénnyé, eképpen:
$f^2(S) = f^1(S)$ - $\min_{S' \in V}\ f^1(S')$. Ez nincs hatással az a_i, b_i értékekre, viszont megváltoztatja a maximum értékeit.

4.1.4. Megjegyzés. Az elemek sorrendje az alaphalmazban befolyásolja a halmazok felett indukált eloszlást. Jelentős hatása lehet az elért maximumra. Tekintsünk egy egyszerű példát ennek értelmezéséhez:
Tekintsük a $PD^2GREEDY$-t $g = g_1$-el. Legyen $V = \{e_1, e_2\}$, $w_{e_1} = 2, w_{e_2} = 1$ és $f(S) = 2 + \max_{i \in S} w_i - |S|^2$. Ez egy nemnegatív szubmoduláris függvény eképpen: $f(\varnothing) = 2, f(\{e_1\}) = 3, f(\{e_2\}) = 2, f(V) = 0$. Ha a $PD^2GREEDY$ az (e_1, e_2) sorrendet tekinti, akkor az algoritmus az e_2 értékével tér vissza (2-vel), míg az (e_2, e_1) sorrendnél az e_1 értékét adja vissza (ez 3).

4.1.2. Differenciálható méretkorlátozott maximalizálás

Ez abban különbözik az ezt megelőző maximalizáló algoritmustól, hogy itt van egy megadott k méretkorlát a halmazokra. Azaz, $\forall\ (V, X) \in \mathcal{X} : f(X) \approx \max_{S \subseteq V,\ |S|=k} f(S)$. A $PGREEDY$ algoritmus úgy épít fel egy k elemű megoldást, hogy az üreshalmazból indul ki és minden lépésben egy elemet ad hozzá. Az i. iterációban hozzáadott elemet véletlenszerűen választja ki a $V \backslash X_{i-1}$ -ből még ki nem választottak közül, ahol az e elem kiválasztásának $p_{e|X_{i-1}}$ valószínűsége függ attól, hogy mennyi az e elem X_{i-1}-hez való hozzáadásának haszna. A kiválasztás valószínűségeket egy t paraméter szabályozza, és $t \to 0$-ra a $PGREEDY$ megegyezik a standard mohó algoritmussal. Az algoritmus indukál egy θ-ban differenciálható eloszlást a k elemű, rendezett részhalmazokon $(\sigma(e_1 ... e_k))$, eképpen:

$$P(\sigma) = \Pi_{i=1}^{k} \frac{\exp(\frac{1}{t} \triangle_{h_\theta}^{+}(\sigma_i | X_{i-1}))}{\sum_{e' \in V \backslash X_{i-1}} \exp(\frac{1}{t} \triangle_{h_\theta}^{+}(e' | X_{i-1}))}, \tag{4.2}$$

ahol $X_{i-1} = \{\sigma_1, ..., \sigma_{i-1}\}$. Összegezve az összes S-hez tartozó σ sorrendjét az elemeknek, megkapjuk az S halmazhoz tartozó valószínűséget: $P(S) = \sum_{\sigma \in \Sigma(S)} P(\sigma)$, ahol $\Sigma(S)$ jelöli az S halmaz elemeiből alkotott permutációk halmazát.

Algorithm 4 *PGREEDY*, véletlenített mohó

1: $h_\theta : V \to \mathbb{R}_{\geq}0,\ k$ méretkorlát
2: $X_0 \leftarrow \varnothing$
3: **for** $i = 1, ..., k$ **do**
4: $\quad C \leftarrow V \backslash X_{i-1}$
5: $\quad \forall\ e \in C : p_{e|X_{i-1}} \leftarrow \frac{exp(\frac{1}{t}\triangle^+_{h_\theta}(e|X_{i-1}))}{\sum_{e' \in C} exp(\frac{1}{t}\triangle^+_{h_\theta}(e'|X_{i-1}))}$
6: $\quad e^* \leftarrow e \in C,\ p_{e|X_{i-1}}$ valószínűséggel
7: $\quad X_i = X_{i-1} \cup \{e^*\}$
8: **end for**
9: **return** X_k

4.1.3. Kísérletek

Maximális vágás

Legyen $G(V, E, w)$ egy súlyozott irányítatlan gráf, ahol V a csúcsok halmaza, E az élek halmaza, és w nemnegatív élsúlyok halmaza ($w_{ij}, \forall\ i, j \in E$). A feladat az, hogy keresni kell egy S részhalmazát a csúcsoknak, melyre $f(S) = \sum_{i \in S} \sum_{j \in V \backslash S} w_{ij}$ vágás érték maximális.

Kép gyűjtemény összegzés

A problémafelvetés részben említett V_i-k itt most a képeknek felelnek meg, az X_i -k pedig az emberek által generált összegzések egy gyűjteményhez. Képek egy V gyűjteményéből szeretnénk kiválasztani egy S részhalmazt úgy, hogy a kiválasztott képek megfelelően összegezzék a V gyűjtemény tartalmát(S lefedje az összes V-beli fontos helyszínt, és az S-beli képek különbözőek).

4.2. Differenciálható mohó háló (DMH)

A Differenciálható Mohó Háló (DMH) [4] a pontosság és visszakeresés tekintetében túlteljesíti a diszkrét optimalizáló algoritmusokat és más módszereket.
A bizonyíték visszakeresés egy kulcsfontosságú része az állítás igazolásnak vagy pl. a kérdés megválaszolásnak. A $FEVER$ (mely egy nagyméretű adathalmaz a Fact Extraction and VERification feladathoz) adathalmazhoz tartozik egy bizonyíték visszakeresési rendszer (amelyhez viszonyítunk), amely a top k mondattal tér vissza, az állítás és a jelölt mondatok közti TF-IDF (Term frequency: egy szó gyakorisága egy dokumentumban, Inverse document frequency: megadja, hogy mennyire jelentős az adott kifejezés a teljes szövegben) jellemzők „koszinusz-hasonlósága" (cosine similarity: hasonlóság mértéke két nemnulla vektor közt, amely az általuk bezárt szög koszinuszát adja meg) alapján.
Egy olyan részhalmaz kiválasztó algoritmust mutatok be, amely gradiens alapú módszerekkel tréningezhető, ezzel közel optimális eredményt ér el a szubmoduláris optimalizáláson keresztül. A $FEVER$ feladaton szeretnénk megadni mondatok egy megfelelő halmazát állítások igazolásához. Erre a feladatra a hagyományos módszerek a mondatokat önmagukban értelmezve vizsgálják meg, így nem halmazként értelmezve optimalizálják a mondatok információtartalmát. Az ötlet, melyen a bemutatandó módszer alapul, egy mohó algoritmus beépítése egy számítási gráfba, ezzel lehetővé téve a gradiens alapú tréningezést és a kiértékelhetőséget. A háló, amelyről itt szó lesz, tanulja szubmoduláris függvény paramétereit és kódol mohó optimalizáláson keresztül, relaxált argmax függvényt használva a tréningezési időben. A tesztelés ideje alatt a modell az argmax-t használja.

Deep unfolding

A deep unfolding az ötlet, mely a továbbiakban a középpontba kerül. Ez jelenti például az autoencoderek (mesterséges neurális hálók egy típusa, melyet adatok kódolásához szoktak használni) esetében az input helyreállítását a kódolás - dekódolás alkalmazásával. Más szavakkal mondva, kisebb dimenzióba való összenyomás (folding) és utána nagyobb dimenzióba való kiterjesztés (unfolding), amíg az input eredeti dimenzióját eléri újra.

Az újítás a korábbi eredményekhez képest egy szubmoduláris halmazfüggvény mohó maximalizálásának differenciálhatóvá tétele. A mohó optimalizáló algoritmusok biztosítanak egy eszköztárat a részhalmaz kiválasztó problémákhoz, ilyenek például az összegzés, visszakeresés, jellemzők kiválasztása. A visszakeresési feladatokhoz, mint a bizonyíték kiszűrése, szubmoduláris függvény optimalizálás lehetővé tesz összefüggéseket a bizonyíték lehetséges darabjai között. Például, tekintsünk egy problémát, ahol adott egy állítás, a cél kiválasztani mondatokat, melyek segítenek igazolni vagy megcáfolni az állítást. Egy hasonló feltételezés a kiválasztási modellekkel kapcsolatban az, hogy a mondatok függetlenek egymástól, ezáltal a kiválasztás lineáris idejű a mondatok számának értelmében.

A szubmoduláris f függvény, amit optimalizálunk, egy SCMM (sums of concave composed with monotone modular function) függvény, mely a következőképpen van megadva:

$$f_\alpha(A) = \sum_{u \in U} \alpha^u \log(1 + \sum_{a \in A} h_a^u),$$

ahol $\alpha^u \in \mathbb{R}^+$, $\forall u \in U$ -ra nemnegatív tréningezhető paraméterek az u indexben és $h_a^u \in \mathbb{R}^+$ az output jellemzők a kódolási rétegből az A mondatokhoz és az U index jellemzőkhöz.

4.2.1. Definíció. *Adott egy V véges halmaz és egy $m_0 : 2^V \to \mathbb{R}$ halmazfüggvény. Ekkor m_0 moduláris, ha*

$$m_0(A) + m_0(B) = m_0(A \cap B) + m_0(A \cup B) \;\; \forall A, B \subseteq V.$$

4.2.2. Állítás. *Legyen $m : 2^V \to \mathbb{R}^+$ egy moduláris függvény, és $g : \mathbb{R} \to \mathbb{R}$ egy konkáv függvény. Ekkor az $f : 2^V \to \mathbb{R}$ függvény eképp definiálva: $f = g(m(A))$ szubmoduláris.*

Bizonyítás. Adott $A \subseteq B \subset V$, $v \in V \backslash B$, legyen $m(A) = x$, $m(B) = y$, és $m(\{v\}) = z$. Ekkor $0 \leq x \leq y$, és $0 \leq c$. Mivel g konkáv, $g(x + z) - g(x) \geq g(y + z) - g(y)$, ebből adódóan $g(m(A) + m(\{v\})) - g(m(A)) \geq g(m(B) + m(\{v\})) - g(m(B))$, mely kielégíti a szubmoduláris függvény definíciójában szereplő egyenlőtlenséget. □

Adott egy monoton szubmoduláris függvény, az optimalizációs probléma megtalálni az optimális A^* részhalmazt, mely maximalizálja $f_\alpha(.)$-t:

$$A^* \in argmax_{A \in V, |A| \leq k} f_\alpha(A).$$

Algorithm 5 Mohó (V,k,f)

1: $\widehat{A} \leftarrow \varnothing$
2: **while** $|\widehat{A}| < k$ **do**
3: $\quad v* \leftarrow argmax_{v \in V \setminus \widehat{A}} \triangle_f^+(v|\widehat{A})$.
4: $\quad \widehat{A} \leftarrow \widehat{A} \cup \{v*\}$.
5: **end while**
6: **return** $\widehat{A}$

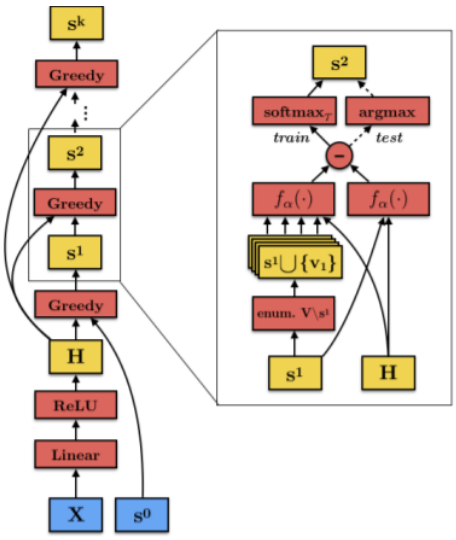
Használhatjuk az 5-ös mohó algoritmust az $\widehat{A}$ részhalmaz megtalálására úgy, hogy

$$f_\alpha(\widehat{A}) \geq (1 - 1/e) f_\alpha(A^*),$$

ahol $(1 - 1/e) \approx 0.63$.

4.2.1. A DMH működése

Ez egy unfolded (azaz kibontott, kiterjesztett) diszkrét optimalizáló algoritmus. A legfontosabb változtatás a háló differenciálhatóságához az $argmax$ függvény az 5-ös algoritmus harmadik sorában. A tréningezés ideje alatt az $argmax$ közvetlenül nem használható, mivel a gradiense nem jóldefiniált. Így megközelítjük az $argmax$-ot a $softmax$-al és használunk egy τ paramétert, mely azt mutatja, hogy ez az érték mennyire közelíti meg az $argmax$-ot.
A következő ábra számítási gráfként ábrázolja a DMH-t.



Ez az 5-ös algoritmusból épült fel, ahol a 2-4 sorok tartalmazzák a mohó rétegeket. Bal oldalon a háló teljes struktúrája mutatja az input jellemzőket, X, mely keresztül megy egy lineáris kódoló rétegen, és ezt követően egy ReLU aktivációs függvényen. Azért a ReLU függvényt választjuk itt, mert az SCMM függvény szubmodularitása nemnegativitást igényel. A kapott nemnegatív jellemzők, H, keresztül megy mohó rétegeken az állapotvektorral együtt, mely elkódolja a mondat kiválasztás előrehaladását. A jobboldali ábra szemléltet egy mohó réteget. A baloldali ágban az i. mohó iterációhoz a hálózat felsorolja a lehetséges következő állapotokat $\{s_i \cup \{v\} : \forall v \in V \backslash s^i\}$. Ezután a háló kiértékeli a potenciális következő állapotokat a jelenlegi s^i állapot értelmében és megtalálja azt a mondatot, amely a legnagyobb növelést biztosítja $f_\alpha(.)$ szempontjából, amely utána hozzáadódik a jelenlegi állapothoz, ez lesz az s^{i+1}. A tréningezés alatt a kiválasztás a $softmax$-al történik egy τ paraméterrel, ezzel differenciálhatóan megközelítve az $argmax$-ot. A tesztelési időben az $argmax$-ot használjuk.

4.2.2. Kísérletek

Alkalmazzuk a DMH-t a FEVER adathalmazra. A FEVER adat tartalmaz állításokat, állítások osztályozását (úgy, mint Támogatott, Megcáfolt vagy Nem rendelkezik elég információval) és bizonyítékot, amelyen az osztályozás alapszik. Az állítások és a bizonyíték a Wikipédiából származik. A FEVER rendszere (amelyhez viszonyítunk) 2 fő lépésből áll: bizonyíték visszakeresés és szövegi következmény felismerése. A rendszer k mondatot kiválaszt a visszakeresett dokumentumok halmazából, amelyeknek legmagasabb a TF-IDF „koszinusz hasonlósága” egy állítás esetében, a DRQA-t (ez egy rendszer a kérdés megválaszoláshoz, a Wikipédiából) használva. A FEVER pontozó függvény kiszámolja a pontosságot, viszahívást és az F1 pontot (egyesíti a pontosságot és visszahívást, értelmezhető, mint súlyozott átlaga a pontosságnak és visszahívásnak, amely a legjobb értékét 1-nél éri el, legrosszabbat 0-nál) a visszakeresett mondatokhoz a jelölt bizonyíték mondatok tekintetében az igazolandó állításokban. A bizonyíték mondatok ezután keresztülmennek egy RTE hálózaton az állítással együtt, hogy az osztályozás megtörténhessen.

Irodalomjegyzék

[1] GEORGE L. NEMHAUSER, LAURENCE A. WOLSEY, MARSHALL L. FISHER, *An analysis of approximations for maximizing submodular set functions-I*, 1978

[2] NIV BUCHBINDER, MORAN FELDMAN, JOSEPH NAOR, ROY SCHWARTZ, *A tight linear time* (1/2)*-approximation for unconstrained submodular maximization*, 2015

[3] SEBASTIAN TSCHIATSCHEK, AYTUNC SAHIN, ANDREAS KRAUSE, *Differentiable submodular maximization*, https://arxiv.org/pdf/1803.01785.pdf, 2018

[4] THOMAS POWERS, RASOOL FAKOOR, SIAMAK SHAKERI, ABHINAV SETHY, AMANJIT KAINTH, ABDEL - RAHMAN MOHAMED, RUHI SARIKAYA, *Differentiable greedy networks* , https://arxiv.org/pdf/1810.12464.pdf, 2018