Domáca úloha DN2 - WGAN pre text

Vladimír Macko

1 Úloha

Úlohou bolo zreprodukovať výsledky článku https://arxiv.org/pdf/1704.00028.pdf.

Článok pojednáva o novej metóde trénovania architektúry GAN, ktorá sa snaží optimalizovať earth mover distance, respektíve Wassersteinovu vzdialenosť.

Optimalizácia tejto vzdialenosti si ale vyžaduje, aby diskriminátor spĺňal Lipschitz constraint a teda aby sa dal vyjadriť ako 1-Lipšicovská funkcia. 1-Lipšicovská funkcia znamená, že má všade gradienty s normou maximálne 1.

V pôvodnom článku sa tento problém riešil pomocou ödstrihávania váh"(weight clipping), kde sa v každom kole váhy prenormovali tak, aby mali normu 1.

Prínosom tohoto článku je, že navrhol optimalizovať funkciu ktorá závisela nie len od Wassersteinovej vzdialenosti ale aj od veľkosti gradientu. Toto sa pre implementáciu ukázalo ako relatívne problematické.

2 Riešenie

Autori v článku uvádzajú aj repozitár s ich kódom https://github.com/igul222/improved_wgan_training ktorý som si forkol do https://github.com/vlejd/improved_wgan_training. Tento kód je napísaný v pythone a používa knižnicu tensorflow. Tento kód podľa mňa písal človek, ktorý mal s pythonom (slušným, neakagemickým) veľmi málo skúseností. Veľmi ťažkopádne sa používal a obsahoval množstvo zlých programátorských praktík.

Za všetky spomeniem keď auto použil:

```
_iter = [0]
def tick():
        _iter[0] += 1
```

Namiesto:

```
_iter = 0
def tick():
        global _iter
        _iter += 1
```

Pochopiteľnosť kódu tiež sťažoval tensorflow (a hlavne moja neznalosť tejto knižnice). V kóde sa totiž používa veľké množstvo konvolúcií, ktoré si vyžadujú reshapovanie a permutovanie dimenzii tenzora s ktorým sa počíta.

Cieľ bol teda jasný, prepísať tento kód do knižnice keras (s ktorou mám tiež zatiaľ len veľmi málo skúseností).

3 Architektúra

Opíšme si ale najprv poriadne architektúru ktorú autori použili (z článku som ju mal problém vyčítať). Autor ako vstupný text používal prvých 32 (fixná dĺžka + padding) znakov z viet z datasetu Billion words od Googlu. Autori nepoužili žiadne predspracovanie. Dokonca v texte nechali všetky nealfanumericke znaky, ktorých sa v pôvodnom datasete nachádza celkom dosť. Predikcie generátora sú potom veľmi dlho počas trénovania "znečistené"týmito znakmi, často na zlých miestach.

Teraz si popíšeme presnú architektúru generátora a diskriminátora. Väčšina ďalej spomenutých tenzorov má rozmer $[batch\_size \times dim]$, kde dim je dimenzionalita modelu (vo všetkých tenzoroch rovnaká). Generátor aj diskriminátor pozostávajú z takzvaných reziduálnych blokov (resblock, pozor na zmenu s na z). To resblocku príde vstup, na tento input sa dva krát aplikuje relu aktivačná funkcia a 1D konvolúcia veľkosti 5. Tento výstup sa následne prenásobí 0.3 a priráta k pôvodnému vstupu. Takéto bloky sa potom ukladajú za seba.

Generátor Najprv vyrobí 128 rozmerný náhodný vektor. Naň aplikuje "hustú"vrstvu a dostane vektor s rozmermi $[sample\_size \times 32 * dim]$, ktorý následne pretvaruje do $[batch\_size \times 32 \times dim]$. Následne sa 5 krát aplikuje resblock. Prekvapilo ma že sa nepoužíva žiaden maxpooling. Posledným krokom je konvolúcia (z dim do 128 rozmerov, dĺžka konvolúcie je 1) a softmax, ktoré vyrobia one hot kódovanie generovanej vety.

Diskriminátor vyzerá takmer ako obrátený generátor. Najprv konvolúcia z 128 do dim, následne päť krát aplikovaný resblock, a posledná je "hustá"vrstva ktorá robí finálne predikcie.

Tieto dve časti sa mi aj podarilo prepísať do kerasu. Problém bol, ako prepísať "cost function". Problém bol hlavne s gradientovou penaltou. Nevedel som totiž, ako a či sa môžem v kerase odvolať na gradiente pri rátaní

chyby a ako správne upraviť kerasový pipeline aby to podporoval. Som si istý, že to ide, ale v momente ako som to zistil som sa rozhodol radšej upraviť pôvodný kód a skúsiť ho na iných dátach.

Autor pri rátaní gradientu postupoval nasledovne. Zobral reálne dáta rd a vygenerované dáta fd a zrátal bod pre rátanie gradientu ako $\text{gd} = \text{rd}(1-\alpha) + \text{fd}\alpha$, kde α je nahodný vektor rozmerov $[batch\_size \times 1 \times 1]$, a teda pre každý vstup sa náhodne rozhodol ako veľmi je reálny a ako veľmi je z generátora. Následne zistil gradient diskriminátora v bode gd a pridal ho do chybovej funkcie.

Ako optimalizátor použil Adama s rýchlosťou učenia $1e-4$, $\beta_1 = 0.5$ a $\beta_2 = 0.9$.

Následné trénovanie fungovalo tak, že v každej epoche sme sa generátorom vygenerovalo niekoľko príkladov, niekoľko prikladov sa vybralo z reálnych dát, na týchto dátach sa po dobu niekoľkých epôch (20) trénoval diskriminátor. Následne sa urobil update generátora.

4 Chybové funkcie

Autor priebežne rátal (vykresľoval a ukladal) chybu diskriminátora.

$$L = \mathop{\mathbb{E}}_{\tilde{x}\sim\mathbb{P}_g}[D(\tilde{x})] - \mathop{\mathbb{E}}_{\tilde{x}\sim\mathbb{P}_r}[D(\tilde{x})] + \lambda \mathop{\mathbb{E}}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}[(\parallel \nabla_{\hat{x}} D(\hat{x}) \parallel_2 -1)^2]$$

Okrem toho pribežne pozoroval kvalitu generátora. Na pôvodných dátach natrénoval 1-gram, 2-gram, 3-gram a 4-gram jazykový model.

Následne natrénoval takéto n=gramové modely aj na generovaných dátach a zaznamenával Jensen–Shannonovu vzdialenosť od tých trénovaných na reálnych.

5 Experimenty

Kvôli týmto experimentom som musel obetovať svoju zásobu mrazenej zeleniny a plechovkového piva, ktoré som počas experimentov musel mať položené na notebooku, aby sa mi kvôli náročným výpočtom a veľkému teplu neprehrial.

5.1 Billion words

Ako prvé som sa pokúsil zreprodukovať výsledky z článku (priečinok cuted). Z časových dôvodov a výpočtových dôvodov som ale musel použiť len časť Billion words datasetu, konkrétne prvých 1000000. Taktiež som nemohol použiť veľkosť dimenzie 500, ako bolo použité v článku, ale len maximálne 300. Trénoval som po dobu 1300 epôch. V článku nie je úplne jasné, koľko epôch použili autori.

Tu sú ukážky textu z článku

```
Busino game camperate spent odea
In during the Uitational questio
Dour Fraps higs it was these del
Divos from The ’ noth ronkies of
```

Vidíme, že tieto texty vyzerajú na prvý pohľad ako normálny jazyk. Sú použité rozumné písmená, rozumne sa striedajú samohlásky a spoluhlásky, veľké písmená sú na začiatkoch slov. Dáta ale stále obsahujú nezmyselné slová.

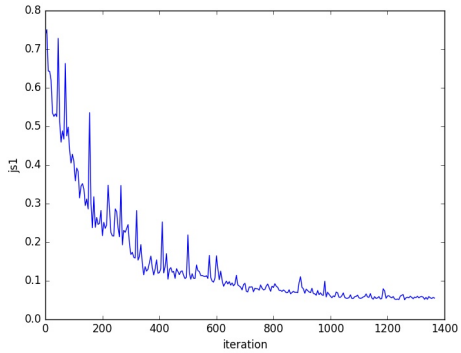
Pre porovnanie, takto vyzerajú reálne dáta.

```
Brent North Sea crude for Novemb
That seems to have been their mo
Gordon will join Luol Deng on th
Nikam maintains the attacks were
```

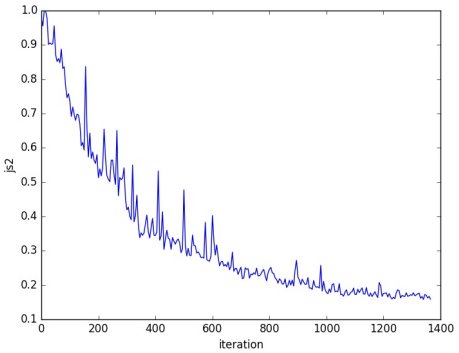
Name natrénovaný model mal nasledovné výsledky:

```
" Is outhis Darners ame e8 ) mat
Uo, wuaut twen ist whit worl th
Thr werk  It a Powast wo tdepe
```

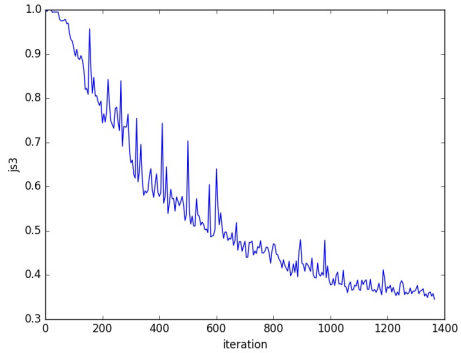
Upozorňujem, že naše dáta neboli cherry pickované. Vidíme, že ich kvalita je o čosi horšia, no dáta majú isté kvality. Správne používanie veľkých písmen, rozumný pomer písmen, správne medzery.



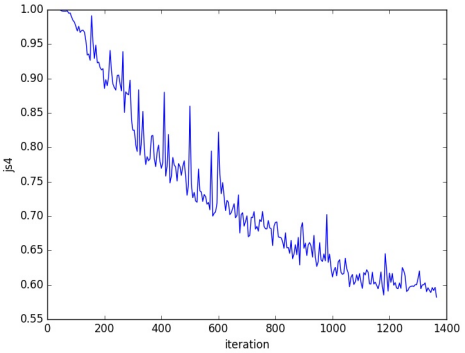
(a) 1-gram



(b) 2-gram

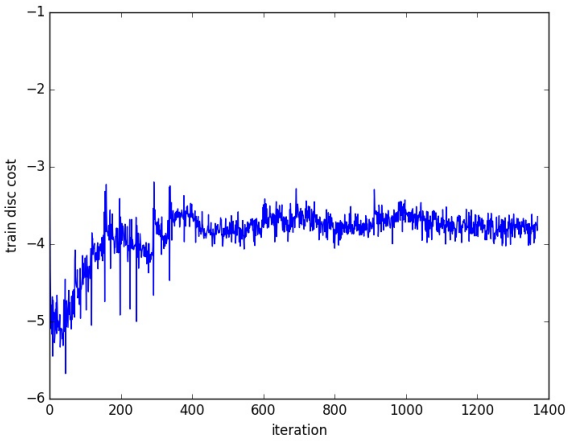


(c) 3-gram



(d) 4-gram

Obr. 1: JS divergencia oproti n-gramom (billion words)



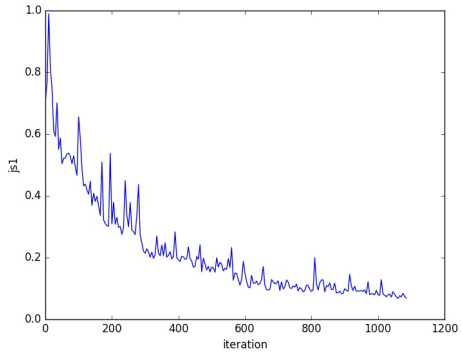
Obr. 2: Chyba diskriminátora (billion words)

Zaujímavé je, že v našom prípade sa síce ešte generátor zlepšoval (v porovnaní s n-gramovým modelom), ale chyba diskriminátora už veľmi (takmer vôbec) nerástla.

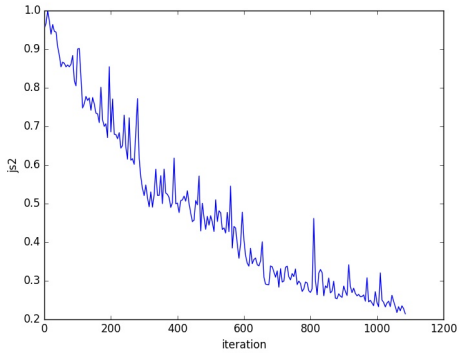
Ako druhý experiment sme skúsili použiť ten istý dataset, ale s dimenziou modelu len 200 (priečinok cuted_small).

Na prvý pohľad vyzerajú dáta ešte o trochu lepšie (keď sa podarí). Zaujímavé je, že v chybách medzi týmto modelom a tým s dimenziou 300 nie je vôbec veľký rozdiel, aj keď bol tento model trénovaný o trochu menej epôch (1000).

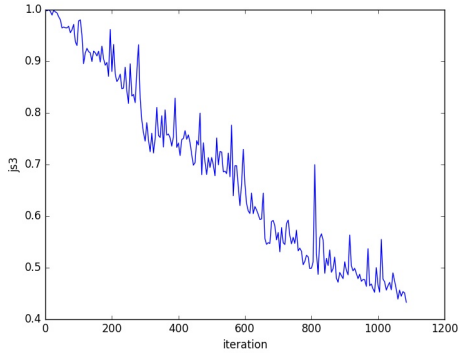
```
Dtrinnges tonto as gig cand in n
Comesnimiog tnk oo reos tho Dtna
A Shin nohe fom tennne han enid
The rospin Site A fter erthe Ant
```



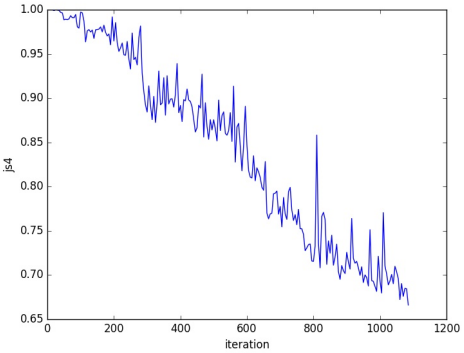
(a) 1-gram



(b) 2-gram

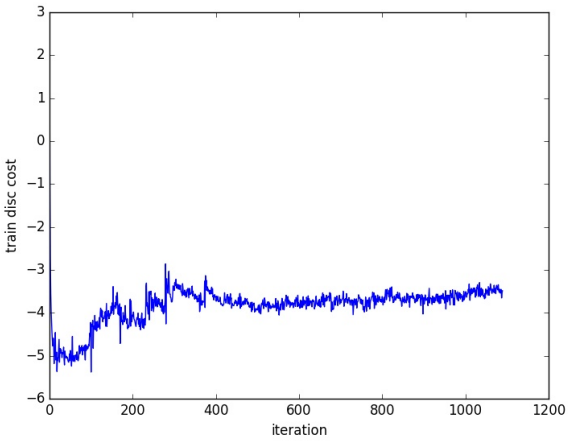


(c) 3-gram



(d) 4-gram

Obr. 3: JS divergencia oproti n-gramom (billion words 200)



Obr. 4: Chyba diskriminátora (billion words 200)

5.2 Shakespeare

Ako ďalší sme použili oveľa menší dataset a to Shakespearove hry. Dáta sme získali pomocou knižnice nltk, kde je to (takmer) štandardný korpus. Tieto dáta sme pozmenili tak, aby každý riadok začínal veľmi podobne. Ospravedlňujem sa za chybu v slove speech, všimol som si to až teraz.

Príklad dát:

```
Speach 1: That I have much ado t
Speach 2: Your mind is tossing o
Speach 2: There, where your argo
Speach 2: Like signiors and rich
```

Náš model po 85 epochách:

```
Speach tt te ah eh h htah ec aah
Speach hh  tah  eh ahaeh eeaahh
Speach ee  eh et ach eth  ht  h
Speacthc etc  e  tt hee ah  ah
```

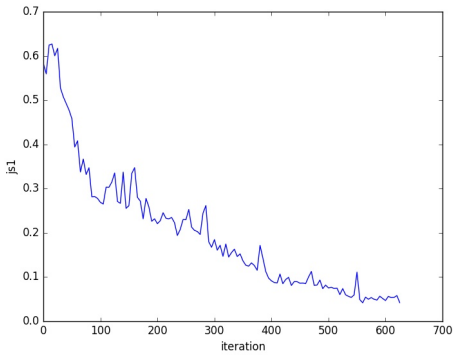
Náš model po 200 epochách:

```
Speach 3q3 ahte 3hpe Speach tatt
Speach 3c3 ppa3  s6t  ath ee at
Speach asacp e 3hce3th e  sath a
Speach l3 thpba3 s sa 6s3a te ah
```

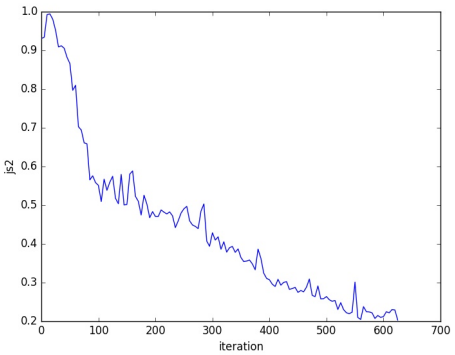
Náš model po 600 epochách:

```
Speach 3: Inothdthee o pas And f
Speach 1h: parlaielmfca fsherth
Speach 2: Thaneat si thf theu m
Speach 23: Sheal h ueald hand ih
```

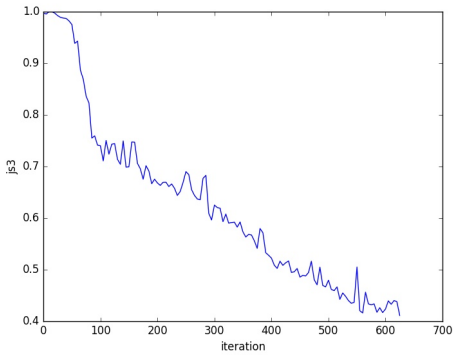
Je krásne vidno ako si model postupne začal uvedomovať, čo musí byť na začiatku riadku.



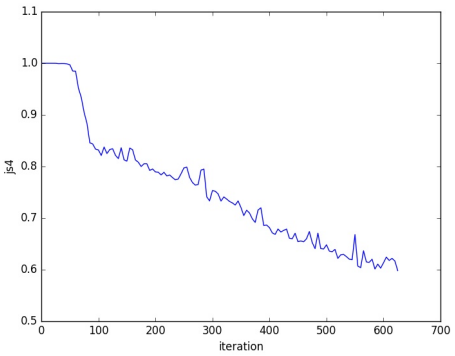
(a) 1-gram



(b) 2-gram

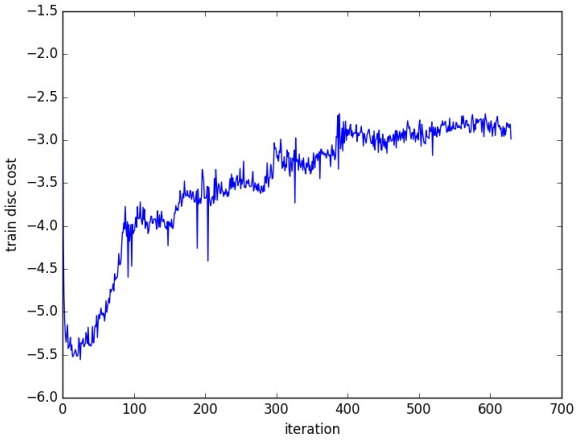


(c) 3-gram



(d) 4-gram

Obr. 5: JS divergencia oproti n-gramom (shakespeare)



Obr. 6: Chyba diskriminátora (shakespeare)

5.3 Quora

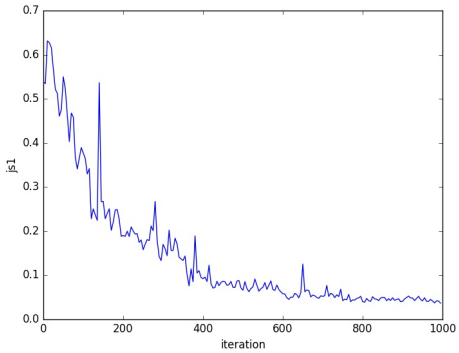
Ďalší dataset ktorý som skúsil boli otázky z quori. Tento dataset je z kagglovej súťaže "Quora question pairs". Tieto dát boli jemne predspracované, boli lowercasnotú a boli z nich odstránené niektoré nealfanumerické znaky.

Reálne dáta

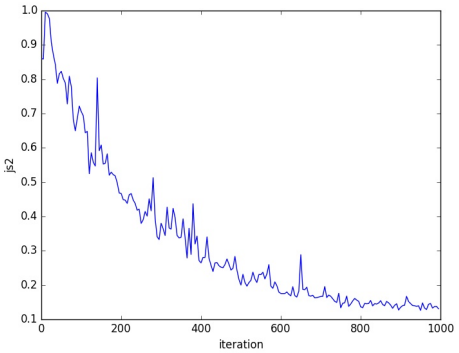
```
how does the surface pro himself
should i have a hair transplant
what but is the best way to send
which food not emulsifiers ?
```

Generované dáta po 995 epochách.

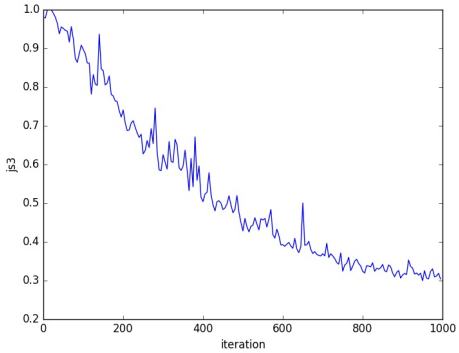
```
what menelicaca mbust is do penv
is ntare she fet if boel sha ‘or
what is is rrand pheples erd ft
is t ane the e wenticonirhap a o
```



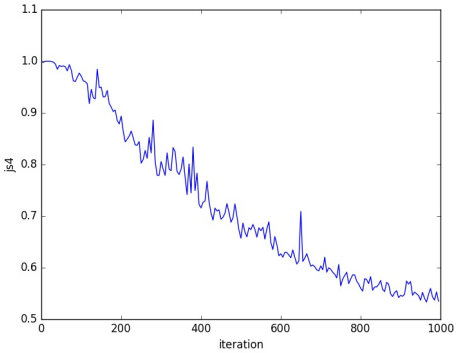
(a) 1-gram



(b) 2-gram

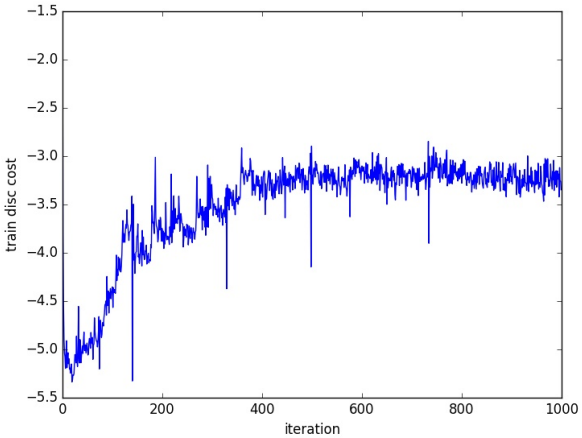


(c) 3-gram



(d) 4-gram

Obr. 7: JS divergencia oproti n-gramom (quora)



Obr. 8: Chyba diskriminátora (quora)

Boli sme sklamaný, že sa model nenaučil používať otázniky.

5.4 Kanye

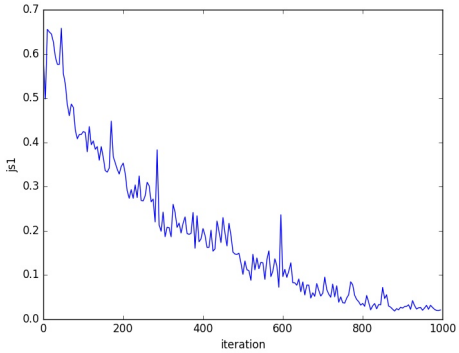
Keďže výsledky nášho modelu boli celkom dobré, ale pre potreby bežnej reči ešte nedostatočné, rozhodli sme sa ako posledný dataset použiť texty speváka Kanye West.

Tento dataset mal len 6554 riadkov a preto dúfame, že na ňom bude mať generátor naozaj dobré výsledky. Reálne dáta

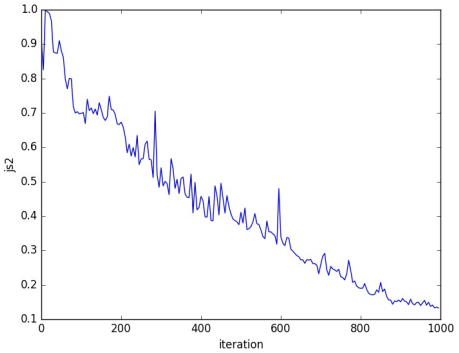
```
Throw 'em some Maybach keys
Fuck it, c'est la vie
I know that we the new slaves
Y'all niggas can't fuck with me
```

Generované dáta po 995 epochách.

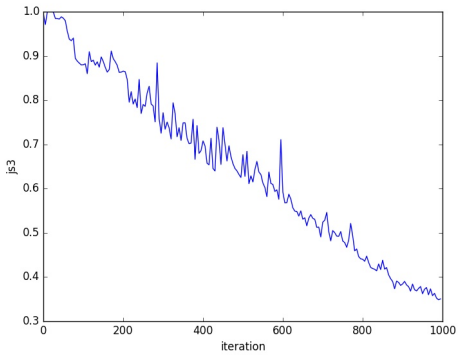
```
Ane thene shesp saike, uite ayn
Low, n’lfin, r‘‘‘‘‘‘‘‘sI n’e ‘‘‘
Thet aour‘d utoe byc‘e‘‘‘‘‘‘‘‘‘
TTmey betlino no hask athe the y
```



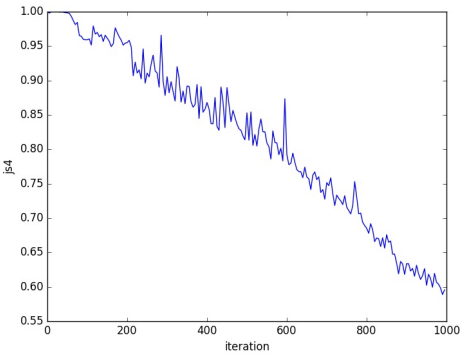
(a) 1-gram



(b) 2-gram

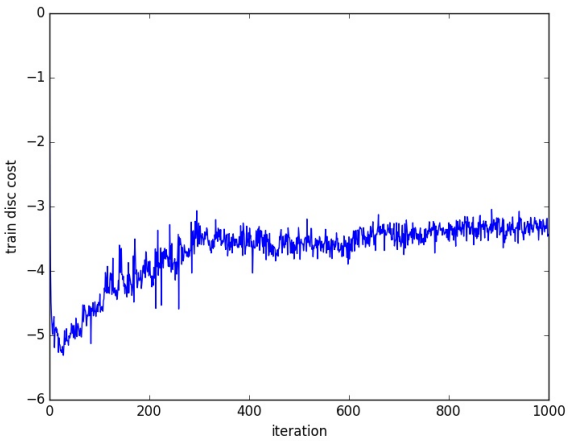


(c) 3-gram



(d) 4-gram

Obr. 9: JS divergencia oproti n-gramom (kanye)



Obr. 10: Chyba diskriminátora (kanye)

Na týchto dátach sa nám prvý krát stalo, že aj po 1000 epochách text obsahoval slová zložené len z artefaktov (apostrofov).

6 Vyhodnotenie

Aj keď sa nám z časových a výpočtových dôvodov nepodarilo presne replikovať kvalitu výsledkov v článku, podarilo sa nám ich replikovať aspoň do relatívne veľkej miery. Okrem toho sa nám podarilo upraviť kód z článku tak, aby bol prístupnejší pre ľudí, ktorý by s ním chceli robiť ďalšie experimenty.

Presvedčili sme sa, že WGAN architektúra sa dokáže naučiť naučiť niektoré aspekty jazyka a že so zlepšeným trénovaním relatívne rýchlo konverguje.

V článku sa pri obrázkových dátach spomína, že krivka chyby diskriminátora zodpovedá kvalite vygenerovaných dát. V našom prípade tak bolo len do prvých 400 epôch. Následne sa síce podľa n-gramov náš model stále zlepšoval, no chyba diskriminátora už toto zlepšovanie zachytávala len málo.

7 PC špecifikácia

Experimenty som bečal na osobnom notebooku lenovo Y 700, procesor Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz, graficka karta GM107M [GeForce GTX 960M] 4GB, 16GB operačná pamäť.

8 Linky

Dáta s popisom a zdrojmi: https://drive.google.com/drive/folders/0B7MLuc1jq3A8eFpVWUZ0eDEwdlE?usp=sharing

Môj vyčistený fork: https://github.com/vlejd/improved_wgan_training