BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2018
http://excel.fit.vutbr.cz

Akcelerácia neurónovej siete pre jazykové modelovanie

Dominik Labaš

Abstrakt

Tento dokument sa zaoberá využitím neurónových sietí v jazykovom modelovaní. Naším cieľom bude takýto model vytvoriť a následne implementovať techniky pre urýchlenie výpočtu pravdepodobností. Implementovaná neurónová sieť sa skladá z jednej skrytej vrstvy so zvolenou základnou veľkosťou 150 neurónov. Pri urýchľovaní neurónovej siete sme pri zachovaní presnosti modelu dosiahli 3-násobné zrýchlenie pomocou zmeny aktivačnej funkcie. Predpočítaním matíc pre výpočet skrytej vrstvy sme výpočet skrytej vrstvy urýchlili 2.5krát. Ukladanie videnej histórie model spomalilo o viac ako 40%.

Kľúčové slová: Jazykové modelovanie — Dopredná neurónová sieť — Akcelerácia neurónovej siete

Priložené materiály: Downloadable Code

xlabas00@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Úvod

Neurónová sieť je výpočtový model navrhnutý tak, aby sa trénovaním naučil vykonávať určitú funkciu. V našom prípade neurónovú sieť využívame pri jazykovom modelovaní, čo je proces rozdelenia pravdepodobnosti nad reťazcami slov, kde samotný jazykový model je funkcia, ktorá tieto pravdepodobnosti dokáže vypočítať ako je znázornené v obrázku (1). Tento spôsob využitia neurónovej siete pri jazykovom modelovaní pre počítanie pravdepodobnosti je používaný od roku 2003 [1].

Pri jazykovom modelovaní, kde je potreba sa učiť komplexné funkcie vzniká takzvaná *Curse of Dimensionality*. Tento problém je dôsledok vysokého počtu rôznych slov nachádzajúcich sa v trénovacích dátach. Tieto slová je potrebné od seba rozlíšiť a trénovací algoritmus vyžaduje príklad pre každú dôležitú kombináciu. Neurónové siete sa v jazykovom modelovaní využívajú pre ich schopnosť učiť sa reprezentácie slov, ktoré pomáhajú proti tomuto efektu tým, že pre každé slovo je vytvorený vektor, ktorý toto slovo popisuje. Tieto vektory sa potom nachádzajú v spoločnom priestore, kde je možné popísať podobnosť a vzťahy medzi týmito slovami.

Skutočná práca jazykového modelu začína až po jeho natrénovaní, kde pri jeho využívaní po ňom vyžadujeme čo najväčšiu rýchlosť. Problému s rýchlosťou doprednej neurónovej siete sa venovali v článku [2], kde navrhli techniky pre urýchlenie výpočtu pre jazykový model. V tomto článku sa pokúsime zreplikovať tri techinky a aplikovať na náš jazykový model.

2. Neurónová sieť ako pravdepodobnostný jazykový model

Jazykové modelovanie pracuje s reťazcami slov v tvare $(w_1, w_2, ..., w_{N-1}, w_N)$ zkrátene (w_1^N), pričom jeho cieľ je vypočítať pravdepodobnosť $P(w_1^N)$. Túto pravdepodobnosť je možné faktorizovať na súčin podmienených pravdepodobností ako:

$$P(w_1^T) = P(w_1) \cdot P(w_2|w_1) \cdot ... \cdot P(w_T|w_1, w_2, ..., w_{T-1})$$

$$P(w_1^T) = \prod_{t=1}^{T} P(w_t|w_1^{t-1}) \quad (1)$$

Pravdepodobnosť $P(w_2|w_1)$ vyjadruje pravdepodobnosť výskytu slova w_2 ak predošlé slovo je práve w_1.

Úlohou jazykového modelovania je teda naučiť sa funkciu rozdelenia pravdepodobností f v tvare:

$$f(w_N, w_{N-1}, ..., w_1) = P(w_N | w_1^{N-1}) \quad (2)$$

Táto úloha bola riešená napríklad za pomoci *n-gram* modelov [3]. Tie využívajú Markovské procesy n-tého rádu, teda slovo vzdialené n a viac miest od nášho aktuálneho slova nemá na pravdepodobnosť žiaden vplyv:

$$P(w_t | w_1^{t-1}) = P(w_t | w_{t-n+1}^{t-1}) \quad (3)$$

V našom prípade využijeme neurónovú sieť, ktorá bude mať za úlohu naučiť sa funkciu pre počítanie pravdepodobnosti:

$$f(w_t, w_{t-1}, ..., w_{t-n+1}) = P(w_t | w_{t-n+1}^{t-1}) \quad (4)$$

2.1 Využitie neurónovej siete

Neurónová sieť sa skladá z niekoľkých vrstiev: vstupnej, výstupnej a jednej alebo viac takzvaných skrytých vrstiev.

Na vstupe modelu sa nachádzajú indexy na slová zo slovníku V, kde slovník V je konečná množina slov vyskytujúcich sa vo vstupných dátach. Tieto indexy model mapuje na vektory vlastností $\mathbf{C}(w_i)$, kde $\forall w_i \in V$. Vektory vlastností majú veľkosť m a sú uložené v matic $\mathbf{C}$ o veľkosti $|V| \times m$. Tieto reprezentácie slov sa model učí v priebehu trénovania súčasne s pravdepodobnostnou funkciou (4).

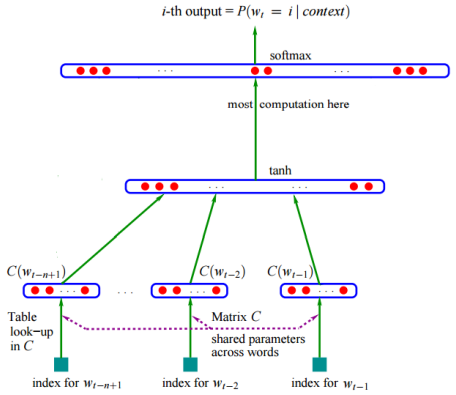
Obsah vstupnej vrstvy neurónovej siete je zreťazená sekvencia vektorov $\mathbf{c} = (\mathbf{C}(w_{t-n+1}), ..., \mathbf{C}(w_{t-1}))$. Túto sekvenciu ďalej model mapuje na rozdelenie pravdepodobnosti pre nasledujúce slovo w_t nad slovníkom V. Výsledkom je vektor pravdepodobností o veľkosti $|V|$ vyjadrujúci pravdepodobnosti (4) pre všetky slová slovníka V.

$$f(w_t, w_{t-1}, ..., w_{t-n+1}) = g(w_t, \mathbf{C}(w_{t-n+1}), ..., \mathbf{C}(w_{t-1})) \quad (5)$$

Funkcia g predstavuje neurónovú sieť, ktorá počíta pravdepodobnosť slova w_t.

2.2 Počítanie pravdepodobností pomocou neurónovej siete

V našom prípade budeme využívať doprednú neurónovú sieť s jednou skrytou vrstvou, ktorá počíta nenormalizované logaritmické pravdepodobnosti. Po namapovaní indexov slov na ich reprezentácie a vložení na vstupnú vrstvu siete sa na jednotlivé hodnoty aplikujú váhy, ktoré predstavujú prechod z jednej vrsvy do druhej.



Obrázok 1. Jazykový model, vstupmi sú indexy na slová zo slovníka V. Vektory vlastností $\mathbf{C}(w_i)$ sú zreťazené na vstupnej vrstve neurónovej siete predstavovanej funkciou g z (5). Výstupom je vektor rozdelenia pravdepodobností nad V. Prevzaté z [1].

$$\mathbf{h} = \tanh(\mathbf{d} + \mathbf{Hc}) \quad (6)$$

Vektor $\mathbf{h}$ predstavuje vektor s výstupom skrytej vrstvy. Vektor $\mathbf{d}$ obsahuje biase pre skrytú vrstvu, matica $\mathbf{H}$ (o veľkosti $|\mathbf{h}| \times |\mathbf{c}|$) obsahuje váhy pre prechod z vstupnej vrstvy do skrytej a vektor $\mathbf{c}$ obsahuje vstupnú vrstvu, kde sú zreťazené sekvencie vektorov $(\mathbf{C}(w_{t-n+1}), ..., \mathbf{C}(w_{t-1}))$. Na každý element skrytej vrstvy je potom aplikovaná aktivačná funkcia, ako aktivačná funkcia sa používa napríklad logistická funkcia alebo hyperbolický tangens.

Následne sú na skrytú vrstvu aplikované váhy pre prechod do výstupnej vrstvy:

$$\mathbf{y} = \mathbf{b} + \mathbf{Uh} \quad (7)$$

Vektor $\mathbf{y}$ (o veľkosti $|V|$) predstavuje nenormalizované logaritmické pravdepodobnosti, vektor biasov $\mathbf{b}$ pre výstupnú vrstvu a maticou váh $\mathbf{U}$ (o veľkosti $|V| \times |\mathbf{h}|$) pre prechod zo skrytej do výstupnej vrstvy a $\mathbf{h}$ predstavuje vektor so skryteou vrstvou.

Množina $(\mathbf{b}, \mathbf{d}, \mathbf{U}, \mathbf{H}, \mathbf{C})$ predstavuje všetky voľné parametre, ktoré sa neurónová sieť učí počas trénovania.

Na výstupnú vrstvu sa aplikuje softmax, ktorý zaistí že súčet všetkých pravdepodobností je 1:

$$P(w_t | w_{t-1}^{t-n+1}) = \text{Softmax}(y) = \frac{e^{y_{wt}}}{\sum_{i=1}^{|V|} e^{y_{wi}}} \quad (8)$$

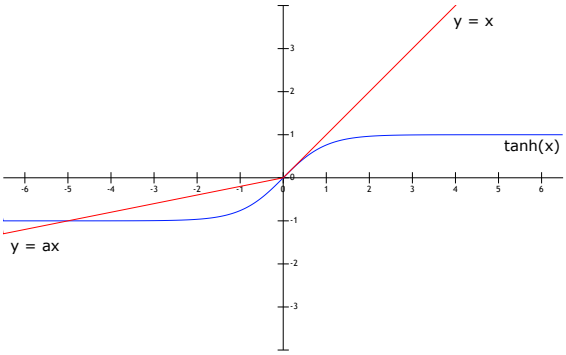
3. Optimalizácia modelu

Pri optimalizácii sa venujeme štyrom technikám popísaným v [2]. Zmene aktivačnej funkcie za rýchlejšiu, zapamätanie si hodnôt z predošlých priechodov cez model, predpočítaním matíc pre rýchlejší výpočet skrytej vrstvy a odstránenie normalizácie z modelu.

3.1 Zmena aktivačnej funkcie

V tejto časti budeme skúmať zrýchlenie pri nahradení stávajúcej aktivačnej funkcie tanh() na rýchlejšiu funkciu PReLU() (Parametric Rectified Linear Unit) [4], ktorá počíta výstup skrytej vrstvy ako:

$$\mathrm{PReLU}(x) = max(0,x) + a \cdot min(0,x) \qquad (9)$$



Obrázok 2. Graf aktivačných funkcií PReLU(x) a tanh(x).

Táto zmena nahradí výpočet funkcie tanh() za rýchlejší výber maxima, s použitím funkcie PReLU() pribúda ďalší voľný parameter a, ktorý sa model učí v priebehu trénovania.

3.2 Ukladanie histórie

Pretože pri priechode cez vstupné dáta je šanca, že nám príde rovnaká kombinácia vstupných indexov, budeme si ukladať časť videnej histórie. V tomto prípade ak nám prídu indexy, ktoré máme uložené, cena vypočítania skrytej vrstvy je vynechaná a skrytá vrstva je načítaná z pamäte. Ak čas T je čas potrebný pre výpočet skrytej vrstvy, ukladanie histórie ušetrí p * T, kde p značí úspešnosť toho, že sa vstup zhoduje s jedným zo záznamov histórie.

3.3 Predpočítanie matíc pre výpočet skrytej vrstvy

Pretože po trénovaní sa výsledky mapovania indexov na reprezentácie slov a váhy pre prechod do skrytej vrstvy ďalej nemenia, je možné si skrytú vrstvu "predpočítať". Toto docielime tým že si vytvoríme matice $\mathbf{M}_j$ o veľkosti $|V| \times |\mathbf{h}|$ pre všetky slová zo slovníku V a všetky možné pozície vo vektore c. Pri počítaní skrytej vrstvy potom stačí sčítať vektory z matíc pre zadané indexy a v modeli pokračovať od aplikačnej funkcie.

$$\mathbf{h} = \tanh(\sum_{j=0}^{n-1} \mathbf{M}_j[w_j]) \qquad (10)$$

Výraz $\boldsymbol{M}_j[w_j]$ značí indexáciu na pozíciu prvku w_j. Biase pre skrytú vrstvu pripočítavame len v jednej matici. Týmto spôsobom ušetríme čas potrebný pre výpočet funkcie (6).

4. Implementácia

Jazykový model je implementovaný v jazyku Python za použitia frameworku PyTorch [5].

Základný model je navrhnutý ako dopredná neurónová sieť, ktorá implementuje rovnicu (5).

Modely prechádzajú dvomi programami. Prvý na trénovanie modelu, kde sa úspešnosť modelu pozoruje pomocou vypočítavania straty. Druhý pre sledovanie času pri priechode cez model využitím profileru z knižníc PyTorch.

4.1 Zmena aktivačnej funkcie

Pri tomto kroku sme funkciu `tanh()` v programe zamenili za funkciu `PReLU()` [4], ktorej implementácia je zahrnutá v knižniciach PyTorchu.

4.2 Ukladanie histórie

Pri ukladaní histórie sa pri evaluačnom využití modelu vytvárajú dve polia, kde v jednom sú uložené vstupné indexy, ktoré boli nedávno modelom videné a v druhom sú uložené hodnoty skrytej vrstvy po využití aktivačnej funkcie.

4.3 Predpočítanie matíc pre výpočet skrytej vrstvy

V tomto bode sa do modelu vložia vypočítané matice M_j, do ktorých ukladáme vynásobené matice reprezentácií slov s váhami pre danú polohu vo vstupnom vektore. Prvá matica sa líši od ostatných pridaním biasov.

Pri ďalších evaluačných volaní modelu potom model le indexuje matice M_j, ktorých hodnoty spočíta do skrytej vrstvy [6]. A model pokračuje od využitia aktivačnej funkcie.

5. Experimenty

Pri trénovaní modelov sa ich úspešnosť meria v dvoch hodnotách loss a perplexita. Tieto hodnoty sa získavajú priebežne z trénovacích dát a po každej epoche z validačných dátach. Po dokončenom trénovaní sa loss a perplexita vyráta pre testovacie dáta. Naše modely porovnávame na základe hodnôt loss a perplexity z testovacích dát, model je presnejší čím je loss alebo perplexita menšia.

Rýchlosti modelov sledujeme podľa času, ktorý potrebujú na jeden priechod cez testovacie dáta. Na začiatok sme vytvorili tri štandardné modely s rôznou veľkosťou skrytej vrstvy. Program využíva validačné dáta a po ich spracovaní vypíše najmenší, najväčší čas

a priemer zo všetkých časov. Hodnoty loss všetkých modelov mali podobne veľkú hodnotu a ich rozdiel nebol signifikantný, preto môžeme povedať, že presnosť všetkých modelov je takmer zhodná.

Tabuľka 1. Porovnanie úspešností modelov. Základný model je označený ako tanh *X*, kde *X* značí veľkosť skrytej vrstvy a reprezentácií slov.

Model	Perplexita	Loss
tanh 50	173,23	5,15
tanh 150	155,35	5,05
tanh 250	159,39	5,07
PReLU	164,21	5,10
cache	158,01	5,06
projekcia	155,35	5,05

Tabuľka 2. Porovnanie rýchlostí základných modelov s rôznou veľkosťou.

Model	Min	Max	Avg
tanh 50	4,72 ms	4,88 ms	4,75 ms
tanh 150	5,55 ms	5,81 ms	5,60 ms
tanh 250	6,27 ms	6,54 ms	6,30 ms

Tabuľka 3. Ukážka výstupu z profileru znázorňujúci časuvú náročnosť jednotlivých krokov pre baseline model tanh 150.

Funkcia	Čas
embedding	145,2 μs
narrow	4,7 μs
narrow	2,4 μs
cat	27,5 μs
view	3,0 μs
t	3,0 μs
expand	3,3 μs
addmm	77,8 μs
tanh	147,7 μs
t	2,8 μs
expand	2,6 μs
addmm	1530,9 μs
logsoftmax	4002,0 μs

V tabuľke výstupu z profileru môžme vidieť časové náročnosti jednotlivých úkonov modelu, kde jednoznačne najdlhšie trvajúci je výpočet funkcie pre logmax, ďalej lineárna transformácia pre výstupnú vrstvu a na koniec výpočet aktivačnej funkcie, mapovanie indexov na reprezentácie slov (embeddings) a lineárna transformácia pre skrytú vrstvu.

5.1 Vstupné dáta modelu

Ako vstup pre trénovanie a testovanie modelu sú využívané dáta *Penn Tree Bank*, jedná sa o text s dĺžkou asi 1M slov, počet rôznych slov v texte je 10k. Dáta sú rozdelené na 3 časti: trénovacie, testovacie a validačné. Dĺžka trénovacieho textu je zhruba 10-krát väčšia ako testovacie alebo validačné dáta.

5.2 Zmena aktivačnej funkcie

Pri použití funkcie `PReLU()` experiment dokázal jednoznačné zrýchlenie pre výpočet aktivačnej funkcie. Výsledkom bolo takmer 3-násobné zrýchlenie výpočtu.

Tabuľka 4. Rýchlosti výpočtu aktivačnej funkcie.

Aktivačná f	Avg
tanh()	146,0 μs
PReLU()	55,7 μs

Tabuľka 5. Celková rýchlosť modelov.

Model	Min	Max	Avg
tanh()	5,55 ms	5,81 ms	5,60 ms
PReLU()	5,37 ms	5,41 ms	5,39 ms

Zrýchlenie celého modelu bolo 3.75%.

5.3 Ukladanie histórie

Pri implementácii tejto techniky sa vyskytol problém, kde pôvodná implementácia umožňuje prácu s 2-rozmerným vstupom, kde potrebný počet zhodných indexov niekoľkonásobne stúpa a tým klesá aj úspešnosť histórie. Preto sme implementáciu pozmenili tak, aby na vstupe modelu bolo vždy len 1-rozmerné pole. Pri využití ukladania videnej histórie pracujeme s úspešnosťou histórie, tá nám značí koľko krát sa v programe hľadaný vstup nachádzal v aktuálne uloženej histórie. Pre porovnávanie so štandardným modelom sme pozmenili vstup modelu len na 1-rozmerné pole. Pri experimentovaní sme vytvorili modely s rôznou dĺžkou histórie.

Zo získaných hodnôt sme zistili, že model je asi o 20% rýchleší ak narazí na hodnotu, ktorú má uloženú v histórii. No ak danú hodnotu v histórii uloženú nemá model je spomalený o viac ako 50%. Pri porovnaní

Tabuľka 6. Porovnanie rýchlostí modelov využívajúce ukladanie histórie (cache) s rôznou dĺžkou histórie a základného modelu.

Model	Min	Max	Avg
baseline	0,68 ms	1,04 ms	0,72 ms
cache 150	0,54 ms	1,61 ms	1,01 ms
cache 300	0,54 ms	1,73 ms	1,04 ms
cache 500	0,55 ms	1,96 ms	1,17 ms

priemerného času sme dokázali že naše riešenie model nezrýchluje ale naopak zpomaľuje o viac ako 40%.

Tabuľka 7. Úspešnosť histórie podľa jej dĺžky.

Model f	Úspešnosť
cache 150	11,3 %
cache 300	15,2 %
cache 500	18,2 %

Hoci úspešnosť histórie so zväčšujúciou sa dĺžkou stúpala, stúpal aj čas hľadania v histórii pri neúspešných pokusoch tak, že so zväčšujúcou sa dĺžkou histórie sa model viac a viac spomaľoval.

5.4 Predpočítanie matíc pre výpočet skrytej vrstvy

Pri využití predpočítaných matíc bola skrytá vrstva vypočítaná 2,5krát rýchlejšie ako so základným modelom.

Tabuľka 8. Rýchlosť vypočítania skrytej vrstvy modelu.

Model	Čas na získanie skrytej vrstvy
baseline	254,8 μs
projekcia	100,2 μs

Tabuľka 9. Celková rýchlosť modelu podľa typu získavania skrytej vrstvy.

Model	Min	Max	Avg
baseline	5,55 ms	5,81 ms	5,60 ms
projekcia	5,37 ms	5,39 ms	5,38 ms

Využitie predpočítaných matíc zrýchlilo model zhruba o 4%.

6. Záver

Táto práca sa venovala zrýchleniu výpočtu pravdepodobností pre jazykový model využívajúci doprednú neurónovú sieť.

V tomto dokumente sme videli, ako jednotlivé spôsoby zrýchľovania ovplyvnili náš model. Pri zmene aktivačnej funkcie sme dosiahli 3-násobné zrýchlenie výpočtu aktivačnej funkcie. Uchovávanie histórie nám kvôli cyklickému prehľadávaniu histórie model naopak 40% spomalilo. Predpočítanie matíc pre skrytú vrstvu nás vo výpočte skrytej vrstve zrýchlilo 2.5-krát.

V tomto dokumente sme úspešne implementovali niektoré zrýchlenia pre jazykový model využívajúci doprednú neurónovú sieť. Naďalej chcem v práci pokračovať a implementovať model bez normalizácie, ktorý sa z dôvodov implementačných ťažkostí nepodarilo do práce zahrnúť a preskúmať vplyv zrýchľovania pre modely s rôznymi veľkosťami skrytých vrstiev a dĺžkami vstupných reťazcov.

Poďakovanie

Chcel by som poďakovať pánovi Ing. Karlovi Benešovi za jeho rady, skvelú pomoc a nekonečnú trpezlivosť pri vypracovávaní tohto dokumentu.

Literatúra

[1] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[2] Yinghui Huang, Abhinav Sethy, and Bhuvana Ramabhadran. Fast neural network language model lookups at n-gram speeds. *Proc. Interspeech 2017*, pages 274–278, 2017.

[3] Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479, 1992.

[4] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[5] PyTorch. Tensors and dynamic neural networks in python with strong gpu acceleration., 2017.

[6] Jacob Devlin, Chris Quirk, and Arul Menezes. Pre-computable multi-layer neural network language models. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 256–260, 2015.