Brno University of Technology
Faculty of Information Technology

SFC
LSTM network demonstration

30. novembra 2020 Svätopluk Hanzel

1 Úvod

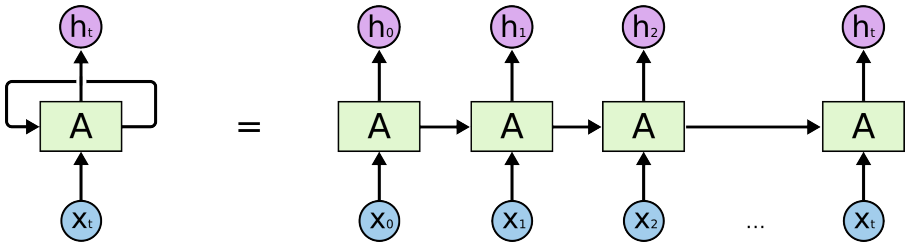
Cieľom tohto projektu je implementovať aplikáciu demonštrujúcu fungovanie rekurentnej neurónovej siete s tzv. Long Short Term memory architektúrou.

1.1 LSTM siete

Neurónové siete s LSTM architektúrou patria medzi tzv. rekurenté neurónové siete. Motiváciou pre ich použitie v kontraste s doprednými neurónovými sieťami je ich schopnosť spájať nové (aktuálne prijaté) informácie s inými dátami, ktoré boli prijaté skôr. Týmto pomáhajú bližšie emulovať napríklad ľudský mozog, nakoľko ani my - ľudia - nezačíname proces rozmýšlania odznova s každým novým slovom.

1.1.1 Architektúra

LSTM sieť výchadza architektúry rekurentných sietí, kde maju jednotlivé bunky spätné napojenie, čím umožňujú predávanie informácie do ďalšieho kroku (viď. obr. 1) Problém klasických

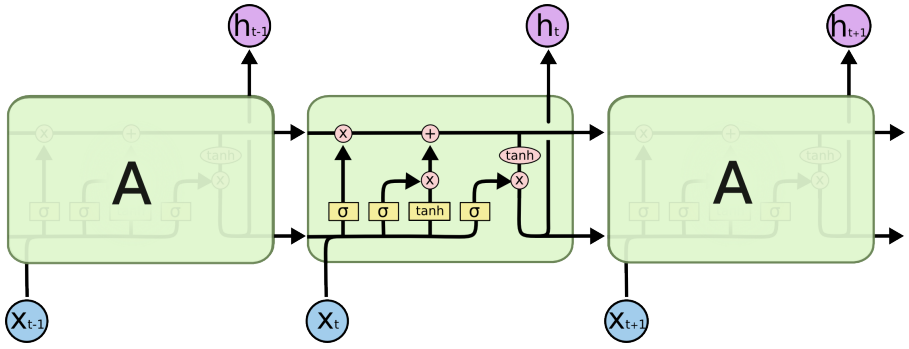


Obr. 1: Architektúra RNN siete s rozvinutou spätnou väzbou. [zdroj]

RNN sietí spočíva v ich jednoduchosti - obsahujú len 1 tanh vrstvu, ktorá neumožnuje detailnú kontrolu informácií, ktoré sieť može potrebovať v niekoľko časových krokov ďalej.

LSTM siete toto obmedzenie obchádzajú vytvorením ďalších tzv. brán (angl. gates), ktoré detailne kontrolujú tok dát.

Základom každej LSTM bunky je *Cell state* (C_t) - hodnota, ktorá ide cez všetky časové kroky a jednotlivé bunky do nej môžu "pridávaťalebo z nej môžu ödoberaťinformácie práve pomocou týchto brán.



Obr. 2: Základná architektúra jednej bunky LSTM siete. [zdroj]

Forget gate Forget gate f_t je sigmoidová vrstva, ktorá ma za úlohu rozhodnúť podľa dát z hidden vektora h_{t-1} z predchádzajúcej bunky a zo vstupu x_t o odstránení niektorých informácií z C_t. Výpočet potom vyzerá následovne:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_i)$$

Kde W_f a b_i sú naučené parametre siete.

Input gate Ďalej sa musí rozhodnúť o tom, ktoré informácie sa majú vložiť do C_t. Preto sa pomocou sigmoidovej vrstvy vypočíta vektor aktivácií i_t

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

Z ktorého sa ďalej spočíta vektor kandidátnych hodnôt $\tilde{C}_t$.

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

Samotná aktualizácia C_t sa následnej počíta pomocou oboch týchto hodnôt a hodnoty cell state z predchádzajúceho kroku:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Output gate Posledným krokom vo výpočte LSTM bunky je vytvoriť nový výstupný vektor h_t, ktorý bude následne pripojený na ďalšiu bunku v poradí. Na toto slúži brána Output gate, ktorá vypočíta z aktuálneho C_t nový vektor h_t, ktorý je jeho filtrovanou verziou. Takto sa môže zachovať nejaká informácia a predá sa ďalšej bunke.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) h_t = o_t * tanh(C_t)$$

1.1.2 Backpropagation through time

Pre korektné natrénovanie každej neurónovej siete je dôležité vedieť aká chyba vznikla pri konkrétnom kroku učenia a napraviť jednotlivé parametre tak, aby dali v ďalšom kroku lepšie výsledky.

V prípade LSTM sietí je klasický backpropagation algoritmus aktualizovaný a rozšírený pre časovú závislosť krokov. Jeho výpočet spočíva v nájdení gradientov jednotlivých parametrov. Tieto sú následne sčítané skrz všetky časové kroky a na ich základe sa vhodne aktualizujú parametre modelu. [1]

Pre úplnosť uvádzam postup výpočtu gradientov počas BPTT.

$$
\begin{aligned}
dv_t &= \hat{y}_t - y_t \\
dh_t &= dh'_t + W_y^T \cdot dv_t \\
do_t &= dh_t * \tanh(C_t) \\
dC_t &= dC'_t + dh_t * o_t * (1 - \tanh^2(C_t)) \\
d\bar{C}_t &= dC_t * i_t \\
di_t &= dC_t * \bar{C}_t \\
df_t &= dC_t * C_{t-1}
\end{aligned}
$$

$$
\begin{aligned}
df'_t &= f_t * (1 - f_t) * df_t \\
di'_t &= i_t * (1 - i_t) * di_t \\
d\bar{C}'_{t-1} &= (1 - \bar{C}_t^2) * d\bar{C}_t \\
do'_t &= o_t * (1 - o_t) * do_t \\
dz_t &= W_f^T \cdot df'_t \\
&\quad + W_i^T \cdot di_t \\
&\quad + W_C^T \cdot d\bar{C}_t \\
&\quad + W_o^T \cdot do_t
\end{aligned}
$$

$$
\begin{aligned}
[dh'_{t-1}, dx_t] &= dz_t \\
dC'_t &= f_t * dC_t
\end{aligned}
$$

2 Implementácia

V rámci tohto projektu som v jazyku C++ implementoval LSTM siet schopnú predikovať jednotlivé znaky. Projekt je rozdelený do viacerých častí - načitanie a príprava dát, práca s maticami a napokon samotná siet.

2.0.1 Načítanie a príprava dát

Pre zjednodušenie práce je táto zodpovednosť presunutá do hlavnej (main) funkcie. Načítanie dát spočíva hlavne vo úprave znakov na malé písmena a vytvorení slovnika, v ktorom su jednotlivé znaky mapované na čisla a takto ďalej odovzdané na čítanie.

2.0.2 Práca s maticami

Vzhľadom na potrebu siete pracovat s vektormi a maticami čísel som sa rozhodol implementovať vlastnú triedu pre matice - `Matrix`, ktorá implementuje rôzne operácie nad maticami; hlavne maticové násobenie, sčitanie matíc, násobenie matíc so skalármi a rôzne matematické funkcie ako napr. tangens, umocnenie čísel v matici a pod.

Samotná trieda je templatovaná, čo umožnuje ukladanie rôznych dátových typov, nie len int čí double.

Hlavnú limitáciu tejto triedy vidím v jej obmedzení na 2 dimenzie a neschopnosti vytvárania rôznych pohľadov na jej dáta. Toto obmedzenie by som chcel do budúcnosti odstrániť.

2.0.3 LSTM sieť

Samotná LSTM sieť je implementovaná v triede `LSTM`. Jej konštruktor berie ako parametre hlavne hyperparametre siete a inicializuje váhy pomocou tzv. Xavierovej inicializácie. [2]

Ďalej má táto trieda 1 hlavnú verejnú metódu - `train`, ktorá spúšťa vo viacerých epochách túto sieť nad celým datasetom rozdeleným na viacero častí podľa dĺžky vstupu siete.

Po každom takomto doprednom prechode vstupu po sieti sa vypočíta chyba prechodu a spustí sa spätný (backward) prechod sieťou. Pri každom kroku spätného prechodu sa vypočítajú gradienty jednotlivých parametrov siete v tomto kroku, ktoré sa postupne sčítavajú.

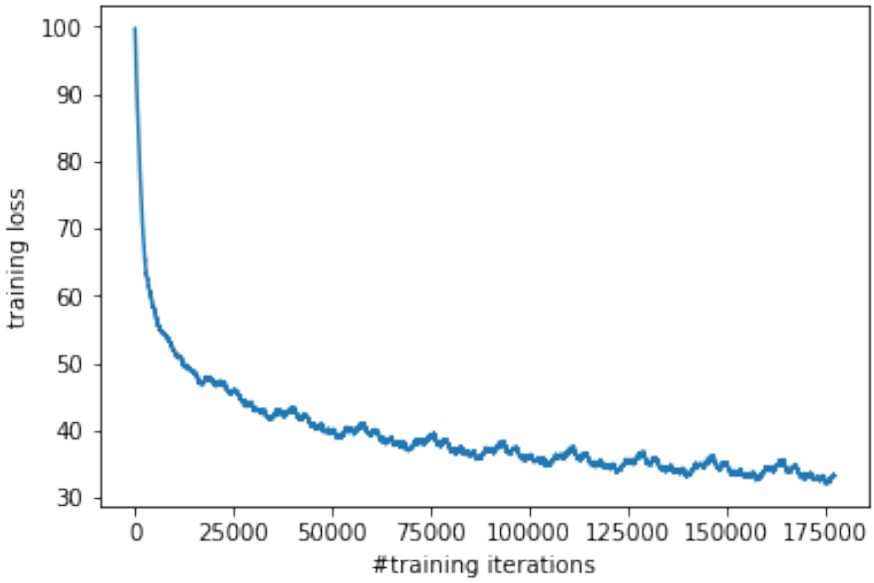
Na ich základe sa potom v metóde `update_params` podľa parametrov optimalizátora a learning rate vypočítajú ich nové hodnoty.

Tento proces sa opakuje pre celý dataset = jednu epochu. Po skončení epochy aplikácia vypíše na štandardný výstup loss pre epochu a ukážkový výstup,

3 Trénovanie

Trénovanie som skúšal na niekoľkých datasetoch - menách detí, dinosaurov a na kompletnej dramatickej tvorbe Williama Shakespeara či knihách Harryho Pottera. Najlepšie výsledky som dosahoval aplikácia pri jednoduchších datasetoch so znakmi pozostávajúcich len z malých písmen anglickej abecedy, teda názvy dinosaurov.

Pri týchto jednoslovných datasetoch sa najviac osvedčila dlžka sekvencie okolo hodnoty 20 znakov pri 20-100 epochách. Po natrénovaní dokáže sieť vygenerovať nové mená dinosaurov na základe vhodného prvého znaku. Ukážka výstupu:



Obr. 3: Priebeh trénovania siete nad datasetom dinos.txt. 100 epôch a lr = 0.001

`austrosaurus utarapateks rapator venasugngowx sinornitholog aladromeus toriosaurus zenimaceitisaurus varaikenoplota colopteryx velocirapteros omaisaurus steronthodesceus zhenyurocoese`

Ako môžme vidieť, sieť sa naučila rozumne striedať spoluhlásky so samohláskami a používať časté prípony ako *-saurus* či *-us* a slová ako *raptor*.

4 Používateľský manuál

4.1 Preklad

Aplikácia používa štandard C++11 a nemá žiadne externá závislosti, takže by mala byť preložiteľná na všetkých strojoch s nainštalovaným prekladačom gcc verzie 4.8 a vyššie.

Na samotný preklad má následne aplikácia priloženy `Makefile` súbor so všetkými potrebnými targetmi pre preklad. Na samostatné preloženie stačí spustiť príkaz `make`. Výsledná binárka potom bude uložená v adresári `dist`.

4.2 Používanie

Aplikácia má niekoľko prepínačov, ktoré umožňujú zmeniť hyperparametre siete alebo parametre učenia. Kompletný zoznam aplikácia vypíše po spustení s prepínačom `-h`. Najdoležitejší je prepínač `-f`, ktorý udáva cestu k datasetu.

Po spustení sa automaticky spustí učenie siete podľa parametrov, po každej epoche vypíše aktuálnu hodnotu loss a vypíše ukážku textu. Po skončení učenia program končí.

5 Záver

Implementovaná verzia je funkčnou LSTM sieťou s učením a BP. Možné vylepšenia vidím hlavne v oblasti práce s multidimenzionálnymi dátami. Mnou vytvorená trieda `Matrix` je príliš neefektívna, čo má za následok dlhé a menej spoľahlivé učenie.

Literatúra

[1] Werbos, P. J. *Backpropagation through time: what it does and how to do it Proceedings of the IEEE* ročník 78, č. 10 1990: s. 1550–1560 doi:10.1109/5.58337.

[2] Kumar, S. K. *On weight initialization in deep neural networks CoRR* ročník abs/1704.08863 2017 URL http://arxiv.org/abs/1704.08863 1704.08863.