SFC - Demonstrace činnosti sítě LSTM

Michal Glos (xglosm01)

5. prosince 2021

1 Úvod

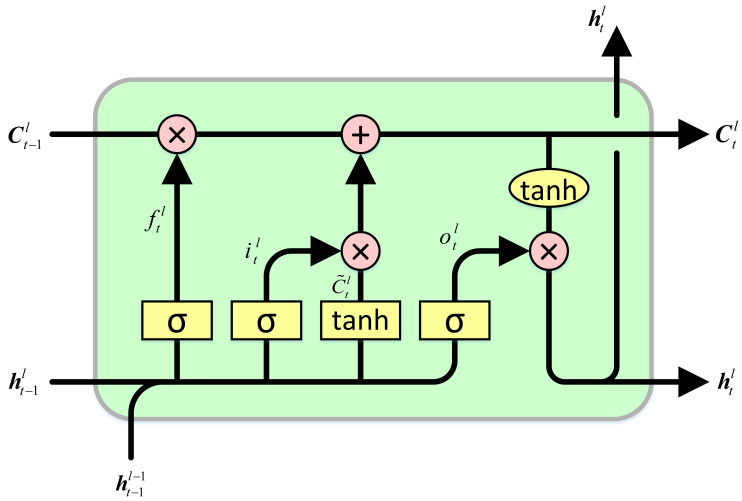
Jako zadání této semestrální práce jsem obdržel 'Demonstrace činnosti LSTM sítě'. Tuto neuronovou síť jsem implementoval v jazyce Python 3, použité knihovny jsou vypsány v sekci 3. Řešení projektu obsahuje celkem 3 soubory, a to modul implementující síť LSTM, modul umožňující práci s datovou sadou a skript, který je pro neuronovou síť rozhraním na příkazovém řádku. Podrobně implementaci popisuji v sekci 3 a práci s rozhraním v sekci 4.

Činnost sítě jsem se rozhodl demonstrovat na veřejné datové sadě od Amazonu, která je dostupná zde. Konkrétně na predikci hodnocení uděleného zákazníkem (1-5 hvězdiček) z textu jeho recenze produktu. Po natrénování síť predikovala přesné ohodnocení zákazníkem s 58,22% úspěšností, viz sekce 5.

2 LSTM

LSTM neboli *Long short-term memory* je architektura rekurentní neuronové sítě (dále jen RNN). Na rozdíl od obyčejných dopředných neuronových sítí, rekurentní sítě mají zpětné vazby, tj. na vstupu nemají pouze vstupní data, ale také svůj výstup z předchozího kroku, čímž tyto sítě získávají schopnosti jako je porozumění kontextu a analýza sekvencí dat nehomogenních velikostí.

Konkrétně architektura LSTM se skládá z LSTM buněk, které ještě často tvoří architekturu nadřazenou, a jejich výstupy bývají analyzovány plně propojenými vrstvami. Jednotka LSTM se skládá z buňky, vstupní brány, výstupní brány a "brány zapomnění", popsány v sekci 3.2. Výhodou LSTM oproti klasickým RNN je vlastnost pamatování si důležitých údajů po mnohem více kroků. Zatímco v klasických RNN informace každou další iterací úměrně klesá, architektura LSTM tuhle problematiku řeší právě svými bránami, kdy se síť de facto rozhodne, zda-li je aktuální informace na vstupu důležitá, či nikoli, a dle toho upraví svůj vnitřní stav. Síť LSTM je vhodná ke klasifikaci, zpracování a predikci sekvenčních dat.



Obrázek 1: Grafické zobrazení LSTM buňky. Výstupní brána je označena symbolem o_t^l, vstupní i_t^l a zapomnění f_t^l. C_t^l a h_t^l jsou pak vnitřní stavy buňky, kdy h_t^l je i její výstup. C_{t-1}^l a h_{t-1}^l jsou výstupy buňky z předchozí iterace a h_{t-1}^{l-1} je vstup, který je výstupem předcházející vrstvy sítě, nebo vstupní vektor sítě. *Převzato z* https://colah.github.io/posts/2015-08-Understanding-LSTMs/

3 Implementace

Součástí této semestrální práce jsou 3 soubory popsány v následujících subsekcích a několik dalších souborů blíže popsaných v subsekci 3.4.

3.1 amznDS.py

Tento soubor implementuje modul amznDS sloužící k práci s datovou sadou od Amazonu, obsahující data k recenzím k produktům. Vzhledem ke kardinalitě datové sady jsem použil pouze jeho část obsahující recenze k počítačovým hrám, avšak úpravou globální proměnné `DATA_URL` odkazující na **gzip** soubor s daty lze skript upravit tak, aby pracoval s jinými částmi dané datové sady. K implementaci tohoto modulu byly použity knihovny `gzip, json, tqdm, numpy, time, os, re, sys, pickle a requests`. Tento modul implementuje 2 operace, a to získání dat vhodně formátovaných pro učení neuronové sítě, popsáno v subsekci 3.1.1 a vytvoření slovníku pro vkládání slov, které je popsáno v subsekci 3.1.2, tzv. *word embedding*.

3.1.1 Získávání datové sady

Datovou sadu lze získat voláním funkce `get_data` ve formátu seznamu slovníků obsahujících dvě položky. První je `overall` - ohodnocení produktu zákazníkem, jehož hodnota nabývá celého čísla v intervalu od 1 do 5 a `reviewText` obsahující seznam řetězců (slov) dané recenze. Funkci lze volat se dvěma volitelnými argumenty, a to `one_class_size`, kterým specifikujeme počet dat pro každou hodnotu ohodnocení a `force`, který, je-li nastaven na `True`, ignoruje dříve uložené datové sady ve formátu pickle a datovou sadu vygeneruje znovu. Výchozí nastavení tohoto argumentu je na `True` kvůli paměťovým omezením na serveru Merlin.

Tato funkce nejprve sestaví cestu k pickle souboru, identifikovaného podle argumentu `one_class_size` a pokud daný soubor existuje a argument `force` je nastaven na `False`, soubor se načte a funkce vrátí připravenou datovou sadu. Pokud však soubor neexistuje nebo byl argument `force` nastaven na `True`, funkce datovou sadu sestaví následujícím způsobem. Nejprve je volána funkce `_load_data`, která nejprve zjistí, je-li *json* s datovou sadou v souboru `DATA_PATH` (data jsou z důvodu paměťových omezení serveru Merlin rozdělena do dvou souborů). Pokud se zde nachází, jsou načteny a navráceny. Pokud se však datová sada v tomto souboru nenachází, je volána funkce `_download_data`, která nejprve zjistí, je-li datová sada ve formátu gzip přítomna v souboru `DOWNLOAD_PATH`. Pokud není, stáhne ji z url uloženého v `DATA_URL`. Dále je gzip soubor extrahován do souborů `DATA_PATH`, odkud již mohou být načtena. Dále jsou hrubá data zpracována funkncí `_filter_json_entries`, která vrátí ze seznamu slovníků seznam podslovníků obsahující pouze klíče specifikované volitelným argumentem `keys`. Dále jsou recenze zpracovány funkcí `_filter_short_data`, která z dat odstraní recenze kratší, než 64 znaků (tuto hodnotu lze nastavit volitelným argumentem `min_len`). Dále je z dat vybráno stejné množství recenzí pro každé hodnocení pomocí funkce `_eliminate_apriori_inequity`. Požadované množství lze pozměnit nastavením volitelného argumentu `one_class_size`. Dále je v datech hodnota ohodnocení převedena na celé číslo a text recenze rozdělen na slova. Data jsou pomocí `np.random.shuffle` náhodně promíchána a případně uložena jako pickle soubor.

3.1.2 Vkládání slov

Protože jsem se rozhodl činnost sítě LSTM demonstrovat na analýze textových dat, je potřeba zajistit způsob, kterým budou jednotlivá slova transformována na vektory. K tomu slouží funkce `create_embedding_dict`, která vytvoří pickle soubor obsahující slovník, ve kterém je pro každé slovo z datové sady uložen vektor stejné velikosti, který slouží jako vstup neuronové sítě. Slovník je tvořen postupnou iterací přes každé slovo v datové sadě, které je nejprve zbaveno ne-alfanumerických znaků a velká písmena jsou převedena na malá, aby se netvořilo několik vektorů pro jinak zapsaná slova stejného významu. Následně je spočítán počet výskytů každého slova v recenzích rozdělených podle ohodnocení. Tak pro každé slovo vznikne vektor velikosti 5, kdy první prvek vektoru obsahuje počet výskytu daného slova v recenzích s hodnocením 1, na druhém v recenzích s hodnocením 2 atd... Složky tohoto vektoru jsou následně poděleny celkovým počtem slov v recenzích s daným ohodnocením. Následně je každý vektor normalizován, aby suma jeho složek bylo rovna 1. Slovník je poté uložen do souboru na cestě `EMBED` ve formátu pickle.

3.2 lstm.py

Tento modul implementuje především třídu LSTM, viz sekce 2, která reprezentuje neuronovou síť LSTM a poskytuje metody k její trénování, vyhodnocení a několik pomocných metod. V tomto modulu jsem použil knohovny `tqdm, os, numpy, sys, pickle, math` a `matplotlib`.

3.2.1 Třída LSTM

Tato třída je inicializována s třemi argumenty. Prvním z nich je `n_classes`, který určuje počet výstupních neuronů a koresponduje s počtem tříd ke klasifikaci. Dalším je `lstm_cells`, který určuje počet buněk LSTM a posledním argumentem je `learning_rate`, který určuje rychlost učení sítě. Součástí inicializace této třídy je také vytvoření modelu - slovníku obsahujícím váhy a sklony (bias) buněk LSTM a na ně napojené plně propojené vrstvy a také načtení slovníku pro vkládání slov.

Další základní funkcí pro práci s modelem LSTM je `_forward_step`, která je volána s argumenty `in_X` - slovo, pro které je dopředný krok vykonán a `old_h` a `old_c` - výstupy buněk LSTM z předchozího kroku. Pokud je dopředný krok prováděn na prvním slovu z recenze a předchozí krok neexistuje, jsou tyto hodnoty vektory nul. Funkce nejprve převede vstupní slovo na vektor (vkládání slov) a vypočítá veškeré parametry sítě podle rovnic níže. Tyto parametry navíc ukládá do mezipaměti, protože jsou později využity při algoritmu zpětného šíření chyby.

$$
\begin{aligned}
i_t &= \sigma([h_{t-1}, x_t]W^i + b_i) \\
f_t &= \sigma([h_{t-1}, x_t]W^f + b_f) \\
o_t &= \sigma([h_{t-1}, x_t]W^o + b_o) \\
\widetilde{C}_t &= tanh([h_{t-1}, x_t]W^c + b_c) \\
C_t &= f_t * C_{t-1} + i_t * \widetilde{C} \\
h_t &= tanh(C_t) * o_t
\end{aligned}
$$

Obrázek 2: Rovnice popisující model LSTM. Písmenem i_t je značena vstupní brána, o_t výstupní brána a f_t brána zapomnění. Spodní index t značí číslo iterace sítě, x_t je vstupní vektor sítě. Operace $[x, y]$ značí konkatenaci dvou vektorů. C_t a h_t značí stav buňky. Proměnné korespondují se schématem 1.

Funkncí, která zajišťuje natrénování sítě LSTM je `_backward_step`, která počítá gradienty vah a posuvů z hodnot uložených v mezipaměti z volání funkce `_forward_step` pro odpovídající vstupní slovo. Tato funkce je implementací algoritmu zpětného šíření chyby a jako chybovou funknci používá křížovou entropii. Její argumenty jsou `y_gt` obsahující pravdivé ohodnocení recenze, `cache` s uloženými hodnotami mezivýpočtů z funkce `_forward_step`, `dh_chain` a `dc_chain` obsahující hodnoty vypočítány v předchozím kroku funkce `_backward_step`, které se využívají ve výpočtech gradientů podle tzv. řetězového pravidla.

Další velice důležitou funkncí je `_train_step`, která spočítá výstup sítě na jedné recenzi a spočítá gradienty vah a posuvů. Funkce příjmá argumenty `X` obsahující seznam slov jedné recenze a `y` obsahující pravdivou třídu recenze. Funkce nejprve postupně volá funknci `_forward_step` pro všechna slova recenze a uloží si všechny její výstupy do seznamu mezipamětí a z posledního výstupu spočítá ztátovou funknci - křížovou entropii. Následně se iteruje přes seznam mezipamětí od konce a vypočítají se akumulované gradienty pomocí funkce `_backward_step`. Funkce pro každou recenzi vrací akumulované gradienty a hodnotu ztrátové funkce.

Trénování sítě je pak implementováno funkcí `train`, která příjmá 6 argumentů, a to `XX` obsahující seznam recenzí a `Y` seznam ohodnocení k recenzím. Dalšími jsou `testX` a `testY`, určeny k evaluaci sítě mezi epochami trénování, `model_name` reprezentující šablonu pro tvorbu pickle souborů naučeného modelu a `epochs` určující kolikrát bude iterováno přes vstupní trénovací data.

Nejprve jsou trénovací data i pravdivé klasifikace náhodně promíchána a nad každým prvkem z trénovacích dat je zavolána funkce `_train_step`, která spočítá gradienty pro úpravu vah a posuvů. Dále je volána funkce `_apply_gradients`, která upraví váhy a posuvy modelu odečtením vypočítaných gradientů vynásobených rychlostí učení. Po ukončení každé epochy je síť vyhodnocena na datech předaných v argumentech `testX` a `testY` a výsledky jsou vypsány do terminálu. Poté je model uložen jako pickle soubor a vývoj ztrátové funkce je pomocí funkce `_plot_losses` zanesen do grafu jak za uplnynulou epochu, tak za všechny iterace aktuálního učení.

Vyhodnocení sítě se pak provádí pomocí funkce `evaluate`, která příjmá dva argumenty, a to `XX` a `Y` obsahující texty recenzí a k nim přiřazené pravdivé ohodnocení. Funkce nad každým prvkem této datové sady volá funkci `_evaluate_entry`, která vrací vektor pravděpodobností příslušnosti textu recenze k daným třídám. Funkce `evaluate` pak vrací seznam dvojic, jejichž prvním prvkem je vektor získán funkcí `_evaluate_entry` a druhým je pravdivá hodnota ohodnocení. Lidsky srozumitelný výstup pak lze získat předáním navráceného seznamu funkci `print_success_rate`, která na terminál vypíše úspěšnost daného vyhodnocení.

3.2.2 Naivní vyhodnocení

Naivní vyhodnocení datové sady je uskutečněno pomocí funkce `neive_evaluate`. Tato funkce příjmá dva argumenty, a to `XX` obsahující texty recenzí a `Y` obsahující jejich pravdivou klasifikaci. Pripcipiálně funkce iteruje přes všechny recenze a pro každou z nich převede všechna její slova na vektory pomocí slovníku na vkládání slov, které pro každou recenzi sečtou. Funkce vrací opět seznam dvojic, kdy prvním prvkem je vektor délky 5, sumu všech vektorů pro slova z recenze a druhým prvkem je pravdivá klasifikace dané recenze. Úspěšnost vyhodnocení lze opět vypsat pomocí funkce `print_success_rate`.

Tato metoda dosahuje úspěšnosti 61,91 % na celé datové sadě.

3.2.3 Pomocné funkce

Modul lstm obsahuje také několik pomocných funkcí. Konkrétně funkci `sigmoid` sloužící k výpočtu funkce sigmoid pro vstupní vektor, `d_sigmoid` sloužící k výpočtu derivace funkce sigmoid ze vstupního vektoru, `d_tanh` sloužící k výpočtu derivace funkce tanh, `softmax` pro realizaci funkce softmax, `cross_entropy` pro výpočet křížové entropie a `print_success_rate`, která příjímá výstupy funkcí `naive_evaluate` a `evaluate`, které zpracují a na terminál vypíše procentuální úspěšnost vyhodnocení.

3.3 lstm-cli.py

Tento skript slouží jako rozhraní pro model z `lstm.py`. Umožňuje model natrénovat, vyhodnotit a nebo provést naivní vyhodnocení. Všechny tyto úkony jsou blíže popsány v sekci 3.2. Tento skript se spouští s několika argumenty:

- `--naive-evaluate`: Odhad ohodnocení recenzí podle algoritmu 3.2.2. Má prioritu před `--evaluate`.
- `--evaluate`: Odhad ohodnocení recenzí podle lstm sítě
- `--train-data-entries n`: Počet recenzí pro trénování lstm sítě ($n > 0 \wedge n \in \mathbb{N}$)
- `--eval-data-entries n`: Počet recenzí pro evaluaci lstm sítě ($n > 0 \wedge n \in \mathbb{N}$)
- `--data-ratio r`: Část recenzí použitých k trénování lstm sítě, zbylá část celé datové sady bude určena k evaluaci sítě. Má prioritu před `--eval-data-entries` a `--train-data-entries` ($r \in \langle 0, 1\rangle \wedge r \in \mathbb{R}$)
- `--learning-rate r`: Rychlost učení sítě lstm ($r > 0 \wedge r \in \mathbb{R}$)
- `--lstm-cells n`: Počet buněk lstm v síti ($n > 0 \wedge n \in \mathbb{N}$)
- `--epochs n`: Počet epoch pro trénování sítě ($n > 0 \wedge n \in \mathbb{N}$)
- `--load-model path`: Cesta k souboru s uloženou sítí lstm
- `--save-model path`: Šablona pro cestu k uložení souboru se sítí lstm

Příklady spuštění jsou v sekci 4. Tento skript pracuje s knihovnami `argparse, pickle, os a numpy`. Nejprve jsou pomocí knihovny `argparse` zpracovány argumenty předány skriptu při spuštění z příkazové řádky. Tyto argumenty dále specifikují velikost datové sady, se kterou bude pracovat a pokud nebyl programu předán argument `--naive-evaluate`, bude buďto model LSTM načten z pickle souboru, nebo dle předaných argumentů bude inicializován model nový. Dále se podle argumentů spustí požadovaný proces učení, vyhodnocení nebo naivního vyhodnocení.

3.4 Další soubory z odevzdaného projektu

V odevzdaném adresáři se vyskytují dvě další složky, a to `./models` a `./data`. V adresáři `./models` se nachází pickle soubor obsahující natrénovanou síť LSTM s nejvyšší přesností klasifikace, příklady jejich použití se nachází v sekci 4. V adresáři `./data` se v moment odevzdání nenachází žádný soubor z důvodu omezení velikosti odevzdávaných souborů, avšak po spuštění se zde vytvoří 4 soubory, a to datová sada ve formátu gzip, rozbalená datová sada rozdělena do dvou souborů json a pickle soubor pro vkládání slov.

4 Příklady spuštění

`python3 lstm-cli.py`
Spuštění trénování modelu s výchozími paramtry.

`python3 lstm-cli.py --data-ratio 0.95 --save-model models/my_model.p --learning-rate 0.0001 --lstm-cells 69 --epochs 10`
Spuštění trénování modelu se specifikací poměru trénovacích dat ku součtu dat trénovacích a vyhodnocovacích na 0.95 (načtena jsou všechna dostupná data), specifikace cesty k uložení modelu do šablony model/my_model_epoch_N.p, specifikace rychlosti učení na 0.0001, počtu lstm buněk na 69 a počet epoch trénování na 10.

`python3 lstm-cli.py --train-data-entries 200000 --eval-data-entries 20000`
Další z možností specifikace trénovací datové sady přímo počtem trénovacích a vyhodnocovacích dat.

`python3 lstm-cli.py --evaluate --load-model models/LSTM_1e-06_64_epoch_18.p --eval-data-entries 100000`
Vyhodnocení modelu uloženého v cestě models/LSTM_1e-06_64_epoch_18.p na 100000 datech.

`python3 lstm-cli.py --evaluate --load-model models/LSTM_1e-06_64_epoch_18.p --data-ratio 0`
Vyhodnocení modelu na všech dostupných datech.

`python3 lstm-cli.py --naive-evaluate --data-ratio 0`
Použití algoritmu naivního vyhodnocení na všech datech.

5 Výsledky klasifikace

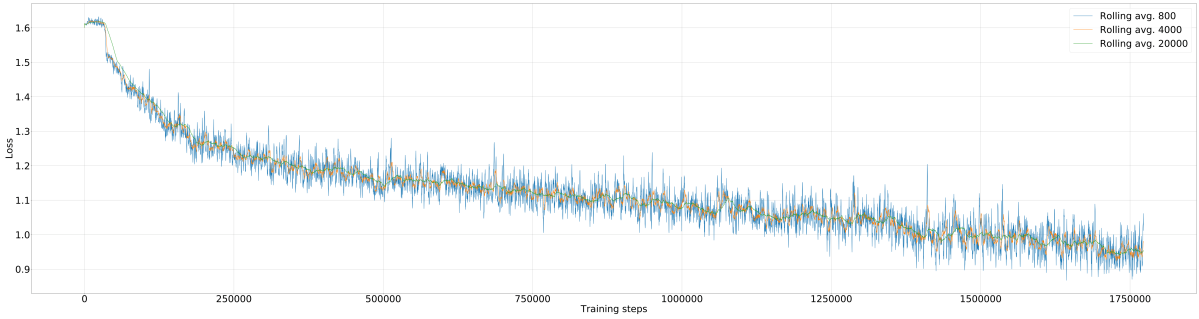
Na výkon sítě LSTM mají vliv 2 parametry, a to rychlost učení a počet LSTM buněk. Několika experimenty bylo zjištěno, že tyto parametry pro tento konkrétní případ klasifikace recenzí z Amazonu mají vliv především na stabilitu sítě během trénování. Nestabilitou v tomto případě myslíme, kdy ztrátová funkce během trénování začne rychle stoupat a síť začne klasifikovat náhodně, viz graf 5. Pokud je rychlost učení příliš vysoká, nebo počet buněk LSTM příliš nízký, trénování sítě LSTM je velice nestabilní. Ideální hodnota rychlosti učení je pod 0.00005, při jejíž použití se síť dokáže učit rozumně rychle a je i relativně stabilní. Snižováním rychlosti učení lze dosáhnout přesnějších výsledků, avšak doba trénování je pak příliš dlouhá. Experimenty s rychlostí učení jsou uvedeny v tabulce 2. Větší počet LSTM buněk pak také vede ke stabilnějšímu procesu učení, experimenty s tímto parametrem jsou uvedeny v tabulce 1. LSTM síť s 96 buňkami však k natrénování potřebovala 12 hodin pro 1 iteraci přes celou datovou sadu, proto, ačkoli by bylo možné dosáhnout větším počtem buněk vyšší přesnosti, byla tato konfigurace sítě největší z konfigurací zahrnutých v experimntech. Ideální počet LSTM buněk pro relativně přijatelnou délku učení a relativně přesné výsledky je 64.

LSTM buněk	**1**	**16**	**32**	**64**	**96**
epocha 1.	*23.72 %*	*45.05 %*	*45.33 %*	*46.45 %*	*47.16 %*
epocha 2.	*26.01 %*	*31.96 %*	*46.41 %*	*46.34 %*	*46.02 %*
epocha 3.	*26.52 %*	*20.00 %*	*20.18 %*	*47.58 %*	*44.53 %*

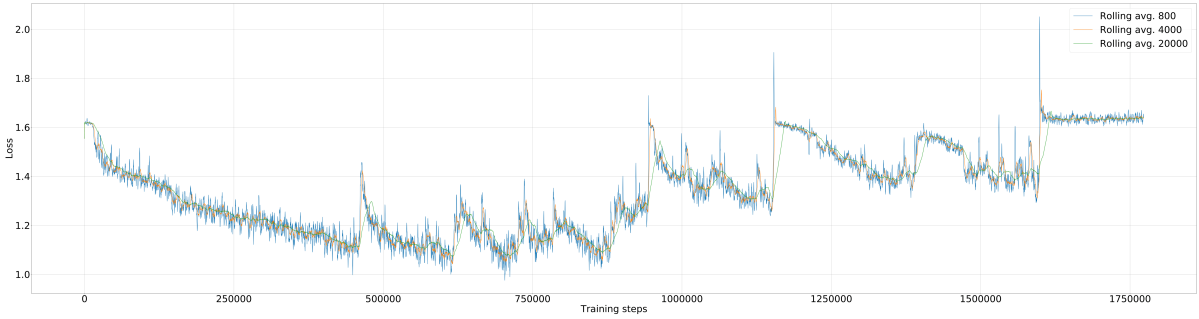
Tabulka 1: Výsledky vyhodnocení sítě LSTM s parametrem rychlosti učení nastaveným na 0.00005 a proměnlivým počtem LSTM buněk.

Rychlost učení	10^{-3}	5×10^{-4}	10^{-4}	5×10^{-5}	10^{-5}	5×10^{-6}	10^{-6}
epocha 1.	*20.00 %*	*35,62 %*	*43,89 %*	*46.45 %*	*47,87 %*	*47,47 %*	*22,03 %*
epocha 2.	*20.00 %*	*23,32 %*	*40,33 %*	*46.34 %*	*52,14 %*	*52,07 %*	*25,49 %*
epocha 3.	*20.00 %*	*23,51 %*	*20,00 %*	*47.58 %*	*32,41 %*	*52,83 %*	*42,23 %*

Tabulka 2: Výsledky vyhodnocení sítě LSTM s parametrem rychlosti učení nastaveným na 0.00005 a proměnlivým počtem LSTM buněk.



Obrázek 3: Na tomto grafu lze vidět průběh stabilního trénování sítě po dobu 3 epoch na modelu s 64 buňkami LSTM a rychlosti učení 0.00001.



Obrázek 4: Na tomto grafu lze vidět průběh nestabilního trénování sítě po dobu 3 epoch na modelu s 16 buňkami LSTM a rychlosti učení 0.00005.

Po analýze úspěšnosti klasifikace sítí natrénovaných různými parametry jsem pokračoval v trénování sítí s počtem LSTM buněk 64 a rychlostí učení 0.000001 a 0.000005. Nejlepšího výsledku jsem dosáhl po 18 epochách s rychlostí učení 0.000001, a to přesnosti 58,22 %.

6 Závěr

Ačkoli se síť dokáže relativně úspěšně naučit klasifikaci recenzí na celkové ohodnocení, i nejlepší dosažené výsledky (58,22 %) jsou však méně přesné, než při použití naivního algoritmu (61,91 %). Přesnost sítě by pravděpodobně bylo možné vylepšit použitím sofistikovanějšího optimalizátoru, jako je například Adam, který bývá pro učení sítí LSTM často používán. Další z možností, jak přesnost této sítě vylepšit, je použití sofistikovanější architektury sítě, kdy se LSTM buňky objevují jak v horizontálních, tak vertikálních vrstvách nebo i použití sofistikovanější architektury sítě klasifikující do tříd výstupy z LSTM sítě. Poslední z možností zpřesnění výpočtů by bylo jiná volba vkládání slov, která by v sobě nesla sémantickou informaci o slově, například použití neuronové sítě Word2Vec.