Gymnázium Jana Keplera
Obor vzdělání: Gymnázium 79-41-K/81

Blekota — generování zvuku pomocí rekurentních neuronových sítí

Václav Volhejn

Vedoucí práce:
Tomáš Obdržálek

Praha, 2017

Obsah

1 Anotace

Blekota používá rekurentní neuronové sítě aby předpověděl, jak bude pokračovat zadaný trénovací zvuk na základě jeho dosavadních hodnot. Generujeme nový zvuk tak, že necháme Blekotu předpovědět pokračovaní zvuku a tuto předpověď ke zvuku přidáme a proces opakujeme. Zkoumáme účinnost tohoto modelu při předvídání a vytváření lidského hlasu a vliv hyperparametrů na kvalitu předpovědí.

Blekota uses recurrent neural networks to predict how a given training sound continues based on the sound's previous data. We generate new sounds by letting Blekota predict the continuation of a sound, append the prediction to the sound and repeat the process. We analyse the model's efficiency when predicting and creating human speech and the influence of hyperparameters on the predictions' quality.

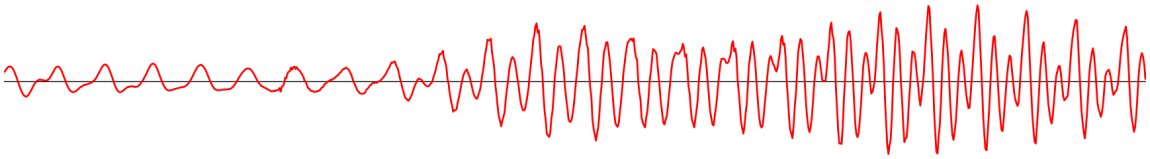
2 Prohlášení

Prohlašuji, že jsem jediným autorem této maturitní práce a všechny citace, použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium Jana Keplera, Praha 6, Parléřova 2 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

3 Úvod

Projekt byl inspirován článkem *The Unreasonable Effectiveness of Recurrent Neural Networks* [1]. Karpathy v něm využívá rekurentní neuronové sítě (RNN) ke generování textu, který se podobá nějakému vzorovému; model se například naučil psát Shakespearovské hry či články na Wikipedii. Blekota se snaží tyto výsledky reprodukovat ve sféře zvuku, tj. generovat zvuky, které mají znít podobně, jako ten, na kterém se model učí.

Zvuk je kvůli hustotě dat větší výzva, než text — aby model generující text vytvořil smysluplnou větu, musí (dostatečně) správně předpovědět desítky znaků. Oproti tomu jedna sekunda zvuku odpovídá tisícům vzorků — CD například používá vzorkovací frekvenci 44,1 kHz. Zvukový model proto musí pracovat na výrazně větší časové škále, aby dával srovnatelné výsledky.



Obrázek 1: Graf 1000 vzorků zvuku o vzorkovací frekvenci 8 kHz. Zvuk odpovídá hlásce *j*.

3.1 Rekurentní neuronové sítě

Cílem *supervised learning* („učení s učitelem“) je naučit výpočetní model (např. neuronovou síť) přiřazovat k daným vstupům správné výstupy na základě označených dat, tj. dat, kde je k dispozici vždy dvojice vstup—správný výstup.

Například bychom mohli chtít model naučit rozpoznávat, zda je na zadaném obrázku kočka. K trénovaní bychom modelu dali milion obrázků, o kterých bychom věděli, zda je na nich kočka, nebo ne. Model vždy zadaný obrázek nějak zanalyzuje a jako výstup dá pravděpodobnost, že je na obrázku kočka. Protože víme, jaká je správná odpověď, dokážeme upravit parametry modelu (čísla, které určují jeho chování) tak, aby na tento konkrétní vstup dával trochu lepší odpověď — pokud byla na obrázku kočka, poučený model by jako výstup nyní dal vyšší pravděpodobnost, a naopak.

Po našem modelu chceme, aby na základě dosavadního zvukového signálu předpověděl jeho další kousek (správně vzorek; *sample*). Zde se tradiční postupy nehodí — normální neuronové sítě nemají žádný stav, nepracují s konceptem minulosti. To, jestli byla na minulém obrázku kočka, nemá vliv na to, jestli bude na příštím. V tomto případě ale další vzorek na minulých zjevně závisí.

Tento problém řeší model zvaný rekurentní neuronová síť (*Recurrent Neural Network*, RNN). Na první pohled funguje model podobně jako běžné modely — dostane nějaký vstup a dá nějaký výstup. Když ale dostane vstup, změní nějak svůj vnitřní stav, takže každý výstup je ovlivněn všemi vstupy, které přišly před ním.

4 Teorie

Formálně máme vektor x, ve kterém x_t je hodnota t-tého vzorku napodobovaného zvuku. Po našem modelu chceme, aby předpověděl x_{t+1} z předchozích t vzorků, tj. aby určil $p(x_{t+1}|x_1 \dots x_t)$.

Přesněji, model má jako vstup x_t (další vzorek zvuku z trénovacího souboru) a jako výstup přiřazuje každé z možných hodnot a pravděpodobnost, se kterou si myslí, že bude následovat, tj. pravděpodobnost, že $x_{t+1} = a$.

Ve zvukových souborech se vzorky obvykle ukládají jako 16-bitová čísla, takže bychom museli v každém kroku spočítat $2^{16} = 65536$ pravděpodobností. Pro zjednodušení kvantujeme vzorky na 256 možných hodnot stejným způsobem, jako WaveNet od Googlu [3]. Jedná se o *μ-law compander* (ITU-T, 1988) [4], který hodnoty nelineárně transformuje tak, že přesněji zachová vzorky blíže nule, kde jsou nuance důležitější.

Abychom dále redukovali množství dat, učíme model na zvucích s vzorkovací frekvencí 8 kHz. Následkem toho se ale zvuk už výrazně zkresluje, protože maximální frekvence zaznamenatelná signálem s frekvencí f_s je $\frac{f_s}{2}$ (Nyquistova frekvence), takže se ztrácí složky s frekvencí nad 4 kHz.

Rekurentní neuronová síť má stav h, což je vektor reálných čísel od -1 do 1. Jeden krok sítě obnáší výpočet nového stavu h_t z předchozího stavu h_{t-1} a vstupu pro tento krok x_t. x_t je nejnovější vzorek zvuku, jehož další vzorek chceme předvídat (model se tedy snaží odhadnout, jaká bude hodnota x_{t+1}).

Nabízí se x reprezentovat jako jediné číslo, které by odpovídalo hodnotě vzorku (kterou jsme kvantovali na 256 hodnot, viz výše). Mocnější reprezentace ale docílíme *one-hot encoding*; vstup bude vektor o 256 prvcích, které odpovídají jednotlivým možným hodnotám vzorků. V každém x budou všechny hodnoty 0, kromě hodnoty toho prvku, který odpovídá hodnotě momentálního vzorku — ten nastavíme na 1.

4.1 Gated Recurrent Unit

Blekota používá variantu RNN zvanou *Gated Recurrent Unit* (GRU), kterou uvedli Cho et al. (2014) [2]. Výpočet h_t v ní vypadá takto:

$$
\begin{aligned}
r_t &= \sigma(W_r \cdot [x_t, h_{t-1}]) \\
h'_t &= tanh(W \cdot [x_t, r_t * h_{t-1}]) \\
z_t &= \sigma(W_z \cdot [x_t, h_{t-1}]) \\
h_t &= (1 - z_t) * h_{t-1} + h'_t * z_t
\end{aligned}
$$

kde $[a, b]$ je skládání vektorů za sebe (z vektorů o rozměrech $[N, 1]$ a $[M, 1]$ vytvoříme jeden o $[N + M, 1]$ vektor), $\cdot$ znamená maticové násobení (v našem případě jen násobení matice-vektor); ostatní aritmetické operace jsou aplikovány po složkách. σ je logistický sigmoid který svůj vstup "zmáčkne"do intervalu $(0, 1)$. Hyperbolický tangens hraje podobnou roli, ale jeho obor hodnot je $(-1, 1)$. W, W_r a W_z jsou matice, které tvoří parametry modelu a při trénování jejich hodnoty optimalizujeme. Proberme význam jednotlivých kroků výpočtu:

r_t je *reset gate*, který vypočítáme z momentálního vstupu a předchozího stavu. Rozhoduje, co z předchozího stavu model zapomene při výpočtu nového - když jsou jeho hodnoty blízko 0, při výpočtu h'_t nezáleží už tolik na h_{t-1} a model musí počítat zejména s x_t; může tak zapomenout hodnoty předchozího stavu, které jsou nyní nepotřebné.

h'_t je *kandidát* (na nový stav), který počítáme z x_t a částečně zapomenutého h_{t-1}. Po vynásobením maticí W výstup necháme projít přes $tanh$, aby zůstaly hodnoty h_t v intervalu $(-1, 1)$.

Samotný h_t pak spočítáme váženým průměrem mezi h'_t a h_{t-1}. Váhy určuje z_t — pokud je nějaká hodnota z_t blízko 0, bude h_t v tomto podobná, jako předtím (tj. bude si dál pamatovat původní hodnotu h_{t_1}), naopak pokud bude blízko 1, hodnota h_t v tomto prvku bude takřka nahrazena novou. GRU se tak může rozhodnout, kdy co zapomenout.

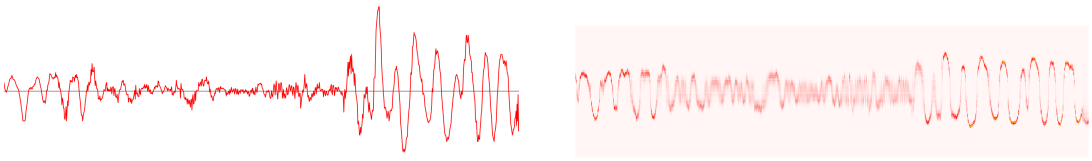
Poznamenávám, že zde ve výpočtu pro přehlednost vynechávám *bias* — ve skutečnosti je u každého maticového násobení k násobenému vektoru přidána konstantní 1, takže např. výpočet r_t vypadá takto: $r_t = \sigma(W_r \cdot [x_t, h_{t-1}, 1])$. Tato konstanta umožňuje modelu při násobení ke každé hodnotě přičíst nějakou konstantu (závislou v tomto případě na nějakém prvku W_r).

4.2 Výstup

Samotné h_t nám ale nestačí, je jen nějaký vnitřní stav modelu. Chceme přeci předvídat hodnotu x_{t+1}. Proto ještě musíme zadefinovat y_t a p_t:

$$
\begin{aligned}
y_t &= W_y \cdot h_t \\
p_t &= \frac{e^{y_t}}{\sum e^{y_t}}
\end{aligned}
$$

y_t je vektor o 256 prvcích, které znamenají váhy jednotlivých možných hodnot x_{t+1}: čím vyšší hodnota prvku, tím vyšší je podle modelu pravděpodobnost, že x_{t+1} bude právě tento prvek. V p_t pak z vah tvoříme skutečné pravděpodobnosti pomocí *softmax funkce*, která vyšší váze přiřadí vyšší pravděpodobnost a zařídí, že prvky p_t se sečtou na 1.



Obrázek 2: Ukázka 1000 vzorků (0,125 s) vygenerovanému zvuku a teplotní mapa jemu odpovídající. Hodnoty vzorků, které model považuje za pravděpodobnější, jsou tmavší. Rozmazaná místa pak odpovídají nejistotě, kde model považuje více různých hodnot za podobně pravděpodobné. [1]

Jak ale pomocí tohoto modelu generovat nový zvuk, ne pouze předvídat vzorky existujícího? Při trénování je postup takový, že napřed dáme modelu jako vstup x_1, ten nám jako výstup dá pravděpodobnosti různých hodnot x_2. Pak mu dáme jako vstup x_2 a on předpoví x_3, atd.

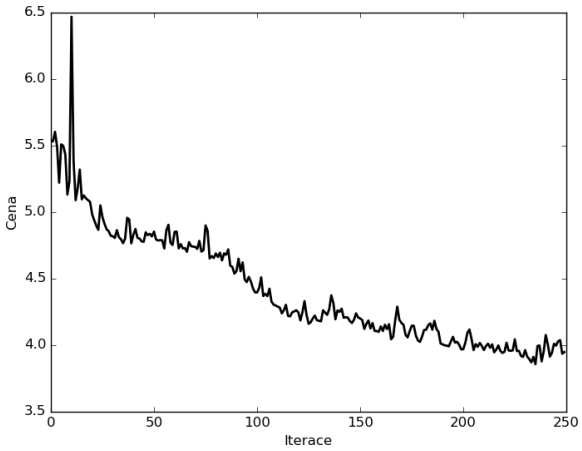
Zde ale žádné x neznáme. To vyřešíme tak, že hodnotu prvního vstupu, tj. x_1 zvolíme arbitrárně. Model pak vytvoří nějakou pravděpodobnostní distribuci hodnot x_2. Podle této distribuce vybereme jednu z hodnot a prohlásíme ji za skutečné x_2. Tuto hodnotu pak modelu dáme jako další vstup. Toto můžeme opakovat, jak dlouho chceme. Výsledné x je pak vygenerovaný zvuk.

4.3 Učení

Abychom model mohli učit, potřebujeme nějak ohodnotit, jak dobré předpovědi dělá. Používáme *cross-entropy loss* — pro naše účely stačí, že cena (chyba) nějaké předpovědi je $-\log(p_i)$, kde p je pravděpodobnostní distribuce, kterou model vytvořil, a i je index skutečné hodnoty dalšího vzorku. Čím nižší pravděpodobnost model přiřadí správné hodnotě, tím větší cena.

Podstatnější než sama hodnota ceny je ale její gradient. Model učíme pomocí *gradient descent*. Všechny kroky výpočtu jsou diferenciovatelné, takže můžeme spočítat gradient (parciální derivace) všech proměnných — jinak řečeno, dozvíme se, jakým směrem musíme posunout parametry modelu (W_y, W, W_r, W_z), aby se cena snížila.

Trénování pak vypadá tak, že necháme model předvídat určitou část zvuku (kolem 100 vzorků), pak spočítáme gradient a změníme parametry podle gradientu. Opakováním tohoto procesu model postupně zlepšuje své předpovědi. Ke konvergenci potřebuje zpravidla tisíce iterací (v posledních pokusech jsem pracoval s 10000 iteracemi).



Obrázek 3: Graf ceny podle počtu iterací

Přesně jakým způsobem měníme parametry závisí na použité optimalizační metodě. Starší verze Blekoty používaly AdaGrad [5], nyní používá RMSProp [6], který se ukázal být výrazně účinnější. Při vytváření programu jsem také dlouho bojoval s laděním rychlosti učení (*learning rate*) α, která říká, jak moc chceme v každé změně parametry upravit. Příliš velké α může optimální hodnoty „přestřelit“, když parametry změní moc.

[1] Důvod, proč tvar Obr. **??** nevypadá stejně jako Obr. **??** je ten, že v teplotní mapě má každá hodnota shodnou výšku; jedná se o obrázek s výškou 256 pixelů. Oproti tomu v praxi jsou hodnoty kvantované nelineárně; viz sekce 4 Teorie. Ve výsledku to znamená, že teplotní mapa je oproti skutečnosti kolem prostředka vertikálně protažená.

4.4 Vrstvení

Mocná technika, která se objevuje napříč strojovým učením, je použití více *vrstev* (layers). Můžeme udělat to, že vezmeme výstup jednoho GRU a dáme ho jako vstup dalšího. Takto můžeme napojit libovolně mnoho GRU za sebe. Každé z nich vstup nějakým způsobem zpracuje a přiblíží se cílovému výstupu – předpovědím.

Místo základního schématu *vstup (zvuk)* → *GRU* → *výstup (předpověď)* tak můžeme dostat pomocí tří GRU toto: *vstup (zvuk)* → *GRU* → *GRU* → *GRU* → *výstup (předpověď)*. Vícevrstvé modely opravdu pracují mnohem lépe, než jednovrstvé – důvodem by mohlo být, že vrstvy blíže u výstupu pracují na dlouhodobější úrovni, protože jejich vstup je stav předchozí vrstvy, který se mění pomaleji, než původní zvukový vstup. Všechny aspoň obstojné výsledky z této práce používají vícevrstvé GRU. V kódu nazývám jednu vrstvu *GRULayer* a celou síť dohromady jednoduše *GRU*.

5 Výsledky

Název	Popis	h_n	l_n	t_n	b_n	it	Parametry
`warble`	Sinusoida s oscilující frekvencí	100	1	100	1	20000	132 956
`drip`	Pravidelně kapající voda	100	2	100	80	10000	193 256
	Varianta s vyšším h_n	256	2	100	80	10000	853 760
`hunger_artist`	Mužský hlas; mluvené slovo v angličtině	256	2	100	80	7000	853 760
	Varianta se třemi vrstvami ($l_n = 3$)	256	3	100	80	4500	1 247 744
`alice`	Ženský hlas; mluvené slovo v němčině	256	3	100	80	9000	1 247 744
`pi`	Mužský hlas; číslice π	512	3	80	60	5000	4 461 312
	Varianta se čtyřmi vrstvami	300	4	80	60	13000	2 201 056
`vali`	Ženský hlas; mluvené slovo v češtině	256	5	80	60	10000	2 035 712
	Varianta se vzorkovací frekvencí 16 kHz	256	5	80	60	10000	2 035 712
`bety`	Ženský hlas; zpěv (slabika „ma“)	256	5	80	80	8000	2 035 712

h_n — velikost stavového vektoru každé vrstvy; l_n — počet vrstev; t_n — délka sekvence (kolik kroků provést před aktualizací parametrů); b_n — kolik sekvencí počítat současně; it — počet iterací trénování; Parametry — celkový počet parametrů modelu

5.1 Warble

Zpočátku jsem hledal co možná nejjednodušší zvuky na testování modelu. Sinusoida s konstantní frekvencí byla ale příliš jednoduchá (modelu stačí předpovědět stejnou hodnotu, jako byla před periodou vlny). Použil jsem proto sinusoidu s rychle kolísající frekvencí. Použitý model byl velmi jednoduchý, ale zvuk napodobil poměrně věrně. Problém má se správným kolísáním; jeho paměť zřejmě není dostatečně dlouhodobá, aby si tyto souvislosti uvědomil.

5.2 Drip

Dalším krokem měl být zvuk ne uměle vytvořený, ale přirozený. Takový zvuk je mnohem méně pravidelný a proto těžší na napodobení. Chtěl jsem proto napřed použít nějaký pravidelný, jednoduchý zvuk. Dobře posloužilo pravidelné kapání vody z kohoutku. Porovnal jsem dvě varianty, z nichž jedna měla vyšší h_n. Ukázka zvuku první varianty ($h_n = 100$) se nachází v `drip_v1.wav`, druhá pak v `drip_v2.wav`. Obě už dokážou generovat nestejnorodý zvuk, který začíná jinak, než končí (narozdíl od předchozí sinusoidy, kde je zvuk „spojitý“). U druhé varianty je napodobení výrazně přesnější; má také asi čtyřikrát více parametrů.

V této fázi jsem zprovoznil ukládání vytrénovaných modelů do souboru, proto nejsou starší modely dostupné k vyzkoušení.

5.3 Hunger Artist

Přesunul jsem se k lidskému hlasu, který jsem pak zkoumal až do konce. Začal jsem s největší výzvou: čtený text. Zvuk je silně nepravidelný a pro model je těžké posunout se dál než k samému zvuku hlasu mluvčího, případně odlišení samohlásek. Jako trénovací soubor jsem použil povídku Franze Kafky *The Hunger Artist* přečtenou Hanifem Kureishi. [**?**]Porovnal jsem dva modely, které se lišily jen počtem vrstev (`v1` — dvě vrstvy; `v2` — tři vrstvy). Oba modely jsem trénoval stejnou dobu z hlediska skutečného času, proto prošla `v2` menším počtem iterací. Výsledky jsou ale ve `v2` výrazně lepší. `v1` zůstává na úrovni jednolitého šumu nebo mumlání. Oproti tomu `v2` už dělá pauzy v řeči a používá různé hlásky, přestože se většinou drží zvuku anglického *w*.

Zde jsem také zkoušel generovat s různou *teplotou*. Všechny váhy v pravděpodobnostní distribuci umocníme na $\frac{1}{\tau}$, kde τ je teplota. Pokud $\tau < 1$, budeme častěji vybírat možnosti, které model považuje za pravděpodobnější. Dostaneme často správnější, ale méně různorodé výsledky. Oproti tomu pokud $\tau > 1$, pravděpodobnosti se vyrovnávají.

U modelu `v2` byla nejlepší teplota kolem 0,9. Model dělá pauzy v řeči, náznaky v intonaci a hlas zní nejpřirozeněji. Nižší teplota generuje většinou jen ticho (protože pro model je většinou nejpravděpodobnější možnost pokračovat v tichu). Naopak při vyšší teplotě nedělá pauzy vůbec a hlas zní zkresleněji, protože se model méně drží správného tvaru zvukové křivky.

5.4 Alice

Stejný model jako `v2` u předchozího pokusu jsem aplikoval na *Alices Abenteuer im Wunderland* Lewise Carrolla; kapitoly 6 a 8 [7]. Cílem bylo srovnat zaprvé mužský a ženský hlas a zadruhé dva různé jazyky. Bohužel tento model dával výrazně horší výsledky než předchozí; generuje většinou ticho protkané občasnými náznaky řeči. Ani při vyšší teplotě nejsou výsledky o moc lepší. Uvádím ještě ukázky z verze modelu po 7300 iteracích, která se naopak vždy dlouho drží na jedné hlásce a pauzy nedělá vůbec.

5.5 Pi

Po neúspěchu *alice* jsem zvolil výrazně pravidelnější zvuk i silnější modely. Přečetl jsem za dvacet minut necelých 2000 cifer *pi* a na tomto zvuku trénoval dva modely. `v1` má velké h_n, ale `v2` má o vrstvu více. Oba modely jsem trénoval stejnou dobu a protože `v1` má asi dvakrát víc parametrů a trénuje se pomaleji, prošel méně iteracemi. Oba modely už ale stagnovaly a dále se nezlepšovaly.

`v1` většinou mění mezi hláskami *š* a *n*. Občas slyšíme náznak čísla (typicky *devět*), ale rychle se vrátí k držení jedné hlásky. Překvapivě lepší výsledky dával `v2`, který je co do počtu parametrů dvakrát menší. Dělá pauzy a co je hlavní, dokáže vyslovit jednotlivá čísla. To je opravdu složité, protože jsou velmi často dvojslabičná a poskládat za sebe všechny hlásky slova *jedna* je nelehký úkol. Stává se také, že model v půlce slova zapomene, co předcházelo, a vzniknou výroky typu *šenda*, což je hybrid *šest* a *jedna*. Přesto je působivé, že model (relativně) správně rozlišuje mezi deseti možnými slovy. Při vyšší teplotě sklouzává stejně jako `v1` k *š*, možná proto, že to je hláska nejpodobnější šumu (který vzniká, když generujeme zcela náhodně).

5.6 Vali

Doufal jsem, že se silnějším modelem bude trénování na přirozené lidské řeči úspěšnější než v předchozích dvou případech. Nechal jsem Valerii Plevovou přečíst úryvek z *1984* George Orwella. V tomto případě jsem srovnával dvě různé vzorkovací frekvence: Model označený `8k` používá 8 kHz jako předchozí modely, `16k` používá 16 kHz.

`8k` dělá vhodně pauzy, ale drží se převážně protažených samohlásek a zvuk proto běžnou řeč příliš nepřipomíná. `16k` je na tom ještě hůře, buďto tiše „syčí“ a nebo vydává hluk, který zní spíš jako vítr. Myslím si, že hlavní problém je malé t_n — t_n sotva stačí na jednu periodu zvuku, takže model si nemůže ani dobře uvědomit periodicitu, proto tvoří spíše prostě hluk. Dále jsem s modelem neexperimentoval zejména kvůli technickým omezením — všechny modely jsem testoval na svém laptopu, kterému na větší modely chybí paměť a výkon.

5.7 Bety

Poslední experiment se od řeči posunul ke zpěvu. Alžběta Volhejnová pro tento účel vytvořila nahrávku dvanácti minut jejího zpěvu, kde nahrazuje všechen text slabikou „ma“. Použil jsem model identický s `vali`. Vždy po 1000 iteracích jsem nechal model vygenerovat krátkou ukázku. První rozlišování ticha, m a a přichází po 4000 iteracích. Průběžně se mění to, jestli model vytváří spíše jednu nebo druhou z hlásek. Také úspěšně používá různé tóny a obstojně je drží — dobrá ukázka je v `bety_7000.wav`.

Když už trénování po 7500 iteracích zjevně stagnovalo, zmenšil jsem rychlost učení α na polovinu a pak ještě více po 8000, 9000 a 9500 iteracích. Při každém zmenšení se skokově zlepší průměr cenové funkce, ale tato technika je postupně méně a méně účinná. Opravdu jsou ukázky po 8000 iteracích výrazně čistší, než předchozí. Hlas už je k nerozeznání od původní nahrávky a tóny kolísají méně. Můžeme pozorovat jednoduché rytmické struktury, ale je možné, že jsou jednoduše dílem náhody.

6 Technologie

Při vytváření projektu jsem pracoval v operačním systému Linux Mint 17.3. Používal jsem editor Sublime Text 3. Projekt sám je implementovaný v Pythonu 3. Využívá matematickou knihovnu NumPy, v níž jsou operace jako násobní matic vysoce optimalizované. Dále používá knihovny Pyplot (na grafy), Pysoundcard (na přehrávání zvuku) a Pysoundfile (na otevírání a ukládání zvukových souborů).

Instalace není potřeba, protože Python je interpretovaný. Stačí mít nainstalované Python 3 a potřebné knihovny: NumPy, Pyplot, Pysoundcard a Pysoundfile.

7 Použití

V základní složce projektu (složka obsahující `src`) zadejte `python3 -i src/blekota.py`. Skript má nápovědu spustitelnou pomocí `python3 src/blekota.py --help` (`-i` můžeme vynechat). Můžeme načíst existující model pokud jako argument zadáme `.pkl` soubor obsahující dříve uložený model (např `python3 -i src/blekota.py mujmodel.pkl`).

Druhá možnost je vytvořit nový model. Jako argument pak slouží `.wav` soubor obsahující zvuk, na kterém budeme model trénovat. Název modelu (tj. prefix cesty, kam se bude model ukládat) můžeme specifikovat pomocí flagu `--model-name`. Např. nastavíme-li `--model-name "foo/bar"`, bude se model ukládat do souborů tvaru `foo/bar_N.pkl`, kde `N` je momentální počet iterací učení. Ostatní hyperparametry naleznete v nápovědě skriptu. Skript sám přiřazuje rozumné hodnoty hyperparametrům, které nespecifikujete.

Po načtení nebo vytvoření modelu s ním můžeme pracovat (flag `-i` při spouštění skriptu je potřeba právě proto, aby se skript po načtení hned nevypnul). Model (objekt třídy `GRU`) se nachází v proměnné `clf`. Je možné používat následující metody (uvádím základní použití, detaily je možné nalézt v kódu):

`clf.train(it)` — trénovat model `it` iterací. Programu nevadí, když trénování přerušíme v půlce pomocí `ctrl+C`, proto je možné nastavit `it` na vysoké číslo a přerušit v libovolnou chvíli. Každých 1000 iterací se model uloží sám.

`clf.sample(n)` — vygenerovat `n` vzorků zvuku (vrací numpy array se zvukem).

`clf.checkpoint(n)` — uložit model a vygenerovat a uložit `n` vzorků zvuku. Pro `n=0` se model jen uloží.

`play(sound)` — přehrát zvuk uložený jako numpy array v `sound`. Přehraný zvuk se uloží do `last_played.wav`. Toto je nejjednodušší způsob, jak uložit vytvořené zvuky.

`show(sound)` — zobrazit graf zvuku uloženého v `sound`.

`heatmap(start, length)` — zobrazit teplotní mapu naposledy vygenerovaného vzorku. Vizualizuje `length` vzorků počínaje od `start`.

K dispozici je ještě pár dalších funkcí a uvedené funkce mají i pokročilejší použití (např. generování s nápovědou a s jinou teplotou). Tyto pokročilejší funkce jsou popsané v kódu.

Předem vytrénované modely a jejich výstupy jsou ve složce `samples`.

Reference

[1] KARPATHY, Andrej. The Unreasonable Effectiveness of Recurrent Neural Networks. In: *Andrej Karpathy blog* [online]. 2015-5-21 [cit. 2017-2-25]. Dostupné z: `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`

[2] CHO, Kyunghyun, B. VAN MERRIENBOER, C. GULCEHRE, D. BAHDANAU, F. BOUGARES, H. SCHWENK, Y. BENGIO. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation [online]. 2014 [cit. 2017-3-8]. arXiv:1406.1078v3. Dostupné z: `https://arxiv.org/pdf/1406.1078v3.pdf`

[3] VAN DEN OORD, Aären, S. DIELEMAN, H. ZEN, K. SIMONYAN, O. VINYALS, A. GRAVES, N. KALCHBRENNER, A. SENIOR, a K. KAVUKCUOGLU. WaveNet: A generative model for raw audio [online]. 2016 [cit. 2017-3-8]. arXiv:1609.03499. Dostupné z: `https://arxiv.org/pdf/1609.03499.pdf`

[4] ITU-T. Recommendation G. 711. Pulse Code Modulation (PCM) of voice frequencies, 1988.

[5] DUCHI, John, E. HAZAN a Y. SINGER. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research* [online]. 2011, 12(2121-2159) [cit. 2017-3-8]. ISSN 1533-7928. Dostupné z `http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf`

[6] TIELEMAN, Tijmen a G. HINTON. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. In: *Neural Networks for Machine Learning* [online prezentace], 2012 [cit. 2017-3-8]. Dostupné z `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`

[7] `Dostupnzhttps://librivox.org/alices-abenteuer-im-wunderland-von-lewis-carroll`

[8] `Dostupnzhttp://download.guardian.co.uk/audio/kip/books/series/short-stories-podcast/1355504069169/8668/gdn.book.121221.tm.Franz-Kafka-story-Hanif-Kureishi.mp3`