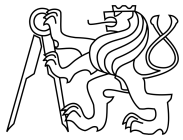
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Generování hudby pomocí neuronových sítí
Student:	Bc. Jan Staněk
Vedoucí:	Ing. Zdeněk Buk, Ph.D.
Studijní program:	Informatika
Studijní obor:	Znalostní inženýrství
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Prozkoumejte metody a algoritmy pro generování hudby včetně vhodných reprezentací (např. notový zápis, analogový signál). Zaměřte se na metody založené na neuronových sítích, zejména LSTM. Vybrané metody implementujte a otestujte na různých hudebních skladbách. V rámci experimentů otestujte schopnost sítě přesně reprodukovat původní skladbu a také schopnost generovat skladby zcela nové.

Seznam odborné literatury

Dodá vedoucí práce.

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 27. ledna 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra znalostního inženýrství

Diplomová práce

Generování hudby pomocí neuronových sítí

Bc. Jan Staněk

Vedoucí práce: Ing. Zdeněk Buk, Ph.D.

9. května 2017

Poděkování

Tímto bych rád poděkoval Ing. Zdeňku Bukovi, Ph.D. za cenné rady při vedení této práce. Dále bych rád poděkoval přítelkyni a rodině za nekonečnou podporu.

Chtěl bych poděkovat i za přístup k výpočetním zařízením a úložným prostorům, které jsou pod správou stran a projektů přispívajících do National Grid Infrastructure Metacentrum, které jsou poskytovány programem „Projects of Large Research, Development, and Innovations Infrastructures“ (CESNET LM2015042).

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 9. května 2017 .....

Odkaz na tuto práci

Staněk, Jan. *Generování hudby pomocí neuronových sítí.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Tato práce se zabývá generováním hudby pomocí rekurentních neuronových sítí. V řešení byly použity modely s GRU a LSTM jednotkami. Ty se snažily v různých konfiguracích naučit vytvářet skladby, a to na základě klasické hudby (určené pro piano) uložené v MIDI formátu. Z tohoto základu se posléze generovala nová díla.

Nejprve se testovala schopnost naučit se konkrétní skladby a z nich vygenerovat identické kopie. Posléze se experimentovalo s generováním skladeb nových. Sítě byly schopné vygenerovat kopie, které se až z 83 % podobaly originálu.

Výsledkem práce je rešerše na téma generování hudby a vypracování programu schopného vygenerovat nové skladby z existujících.

Klíčová slova Generování hudby, neuronová síť, RNN, LSTM, GRU, MIDI, Python, Keras

Abstract

This thesis focuses on generating music using recurrent neural networks. Models with GRU and LSTM nodes in various configurations were trained on

a data with classical piano music stored in a MIDI format. Based on these data the model has generated new compositions.

Firstly the ability to learn a specific composition and to generate identical copies was tested. Secondly experiments with generating new music pieces took place. Neural nets were able to generate copies that were identical to the original by 83 %.

The result of this work is an analysis of several projects dealing with music generaction and creating a program capable of generating new compositions based on existing music pieces.

Keywords Music generation, neural network, RNN, LSTM, GRU, MIDI, Python, Keras

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Umělá inteligence v poslední době proniká do stále většího počtu odvětví, kde ještě před pár lety člověk hrál prim. Každým dnem přibývají projekty, které mají přímý dopad na život lidí a každým dnem desítky miliónů lidí komunikují s jedním z mnoha chytrých asistentů, aby si pomocí pár slov ulehčili práci. Dnes například existují modely, které dokážou rozpoznat rakovinu kůže stejně dobře jako dermatologové[1] nebo modely, které dokážou za určitých situací sami řídit auto[2].

Umělá inteligence proniká i do oblasti umění. V České republice vznikl projekt od Jiřího Materny, který pomocí rekurentních neuronových sítí vytvořil výběr básní v tištěné podobě[3]. Jiný projekt dále využívá hlubokou neuronovou síť k tomu, aby se vyfocený obrázek transformoval v umělecké dílo ve stylu různých světoznámých umělců[4].

Hlavním cílem této práce je prozkoumat současná řešení v oblasti strojového generování hudby a následně pomocí umělé inteligence tvořit vlastní hudební díla. K tomu budou využity různé konfigurace modelů využívající rekurentní neuronové sítě. Výkony sítí budou následně kvalitativně srovnány pomocí několika metrik – například schopnost sítě vygenerovat skladbu totožnou s originálem.

Motivace za výběrem neuronových sítí jako způsobu generování hudby je zejména ta, že v poslední době zažíváme boom v oblasti umělých neuronových sítí a tato práce je jeden ze způsobů, jak ověřit jejich sílu a jak se s nimi více seznámit.

Práce začíná úvodem do problematiky umělých neuronových sítí. Ta by měla přiblížit použité metody v experimentech a nastínit rozdíly mezi současnými řešeními, které využívají ke generování umělé neuronové sítě.

V další kapitole bude popsána analýza několika vybraných současných řešení, které generují hudbu prostřednictvím notového zápisu nebo přímo pomocí analogového signálu.

Třetí kapitola popíše zdroje a formát dat, které byly použity v experimentech. Následující kapitola potom přiblíží vývojové prostředí a samotnou

implementaci.

V kapitole Experimenty budou provedeny vybrané pokusy na vybraných typech modelů a ty se posléze vyhodnotí. V závěrečné kapitole dojde na testování výsledků experimentů na uživatelích a celkové vyhodnocení úspěšnosti vygenerovaných skladeb.

Kapitola 1

Úvod do problematiky neuronových sítí

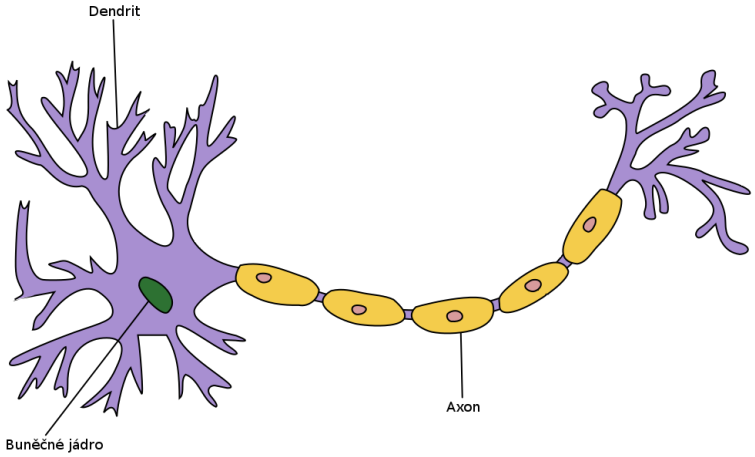
Jako první věc před seznámením se s výčtem současných řešení a před samotnou realizací je nutné uvést základní úvod do oblasti umělých neuronových sítí a uvést pár konkrétních příkladů. To je potřebné k alespoň částečnému pochopení rozdílů mezi uvedenými současnými řešeními, které používají neuronové sítě a následně u experimentů v rámci této práce.

Jako inspirace pro umělé neuronové sítě posloužil samotný lidský mozek. Oproti běžnému stolnímu počítači v domácnostech lidský mozek funguje na kompletně jiném principu. Běžný stolní počítač vykonává činnost na bázi naprogramovaných instrukcí a ty postupně plní. Sám od sebe neudělá nic, co by mu nebylo předem nastaveno. Na druhé straně lidský mozek se po dobu života vyvíjí a upravuje své chování podle zkušeností, které ve svém životě nabyl. Tak jako se lidský mozek přes všechny pády a chyby postupně naučí, jak se má správně jezdit na kole, podobným způsobem se učí neuronová síť.

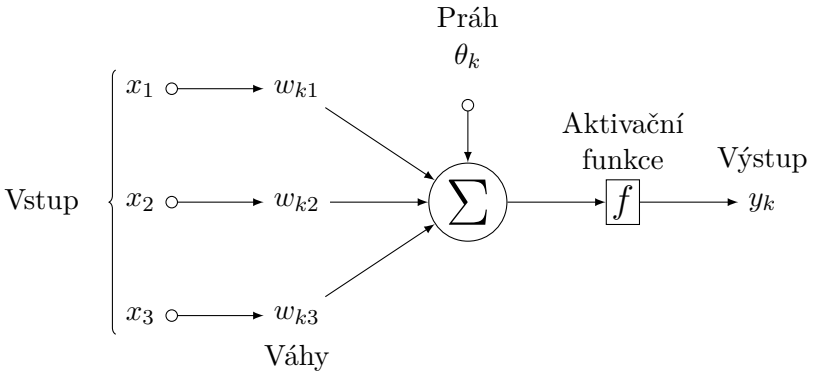
Mozek se skládá z desítek miliard buněk, které jsou spolu navzájem propojené a které spolu komunikují pomocí chemických a elektrických signálů. Tyto buňky jsou nazývány neurony a spolu tvoří neuronovou síť. Každý neuron reaguje na jiný podnět jinak a zjednodušeně podle reakce skupin neuronů na určitý podnět člověk například pozná, jestli pes na obrázku je labrador nebo retrívr.

1.1 Neuron

Biologický neuron (viz 1.1) se skládá ze tří hlavních částí: dendrity, buněčné jádro a axon. Signál putuje z těla neuronu přes axon k axonálnímu zakončení, kde dojde k přenosu signálu na jiný neuron pomocí synapsí. Ty jsou napojené na dendrity, které slouží jako přijímače signálů. Každá nervová buňka může mít v rozmezí od 20 do 1000 synapsí a samotný axon může nabývat délky



Obrázek 1.1: Biologický neuron [6]

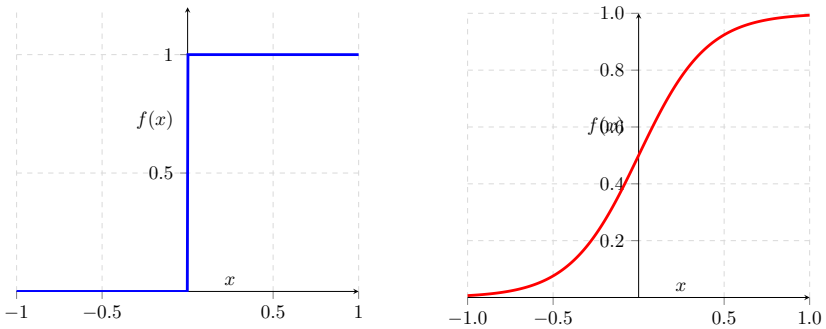


Obrázek 1.2: Umělý neuron k

až jednoho metru. Celkově lidský mozek obsahuje neuronovou síť, která má počet synapsí v řádu stovek trilionů.

Spojení mezi neurony není statické, ale naopak se s časem mění. Čím více signálů se emituje mezi dvěma neurony, tím silnější spojení mezi nimi vzniká. To vede ve finále k tomu, že s každou novou zkušeností a každou novou vzpomínkou se pozmění struktura neuronové sítě. [5, 6]

Umělý neuron (viz obrázek 1.2) se skládá ze vstupních hran x_i, které představují synapse. Každá tato hrana je charakterizovaná vlastní váhou w_i, kterou příchozí signál zesiluje a tím určuje, jakou mírou se daný vstup podílí na výstupu. Váha může nabývat i negativních hodnot, čímž příchozí signál tlumí. Dalším prvkem umělého neuronu (dále jen neuron) je agregační funkce, která sumarizuje všechny vstupní signály vynásobené váhou každé synapse. Následující část představuje přenosovou funkci f, která určuje výstupní hodnotu. Přenosová funkce se může vyskytovat v různých podobách jako například v po-



(a) Graf skokové funkce pro h=0

(b) Graf sigmoidy pro $\lambda = 5$

Obrázek 1.3: Grafy skokových funkcí

době sigmoidy nebo prahové funkce. Výstupní hodnota se potom předá dál.

Posledním prvkem zobrazeným v grafu je práh θ, který se přidává do aktivační funkce spolu s výše zmíněnou sumarizací. Výstup y_k neuronu k s n vstupy se dá popsat těmito vzorci:

$$s_k = \sum_{i=1}^{n} w_{ki} x_i \tag{1.1}$$

a

$$y_k = f(s_k + \theta_k) \tag{1.2}$$

Umělý neuron představuje základní jednotku velkého množství umělých neuronových sítí (ANN). Neuronovou síť lze vyjádřit pomocí orientovaného grafu, kde uzly představují jednotlivé neurony a hrany spojení mezi nimi. Orientace hrany reprezentuje směr toku signálů.

1.1.1 Aktivační funkce

Přenosová neboli aktivační funkce, na grafu označená f, definuje výstup neuronu. Jako příklad aktivačních funkcí můžou být použity následující.

1.1.1.1 Skoková přenosová funkce

Skoková přenosová funkce zobrazena na obrázku 1.3a se dá popsat pro vstup x a prahovou hodnotu h pomocí následujícího vztahu.

$$f(x) = \begin{cases} 1, & \text{pro } x \geq h \\ 0, & \text{pro } x < h \end{cases}$$

1.1.1.2 Sigmoida

Další typ aktivační funkce je sigmoida, která zastupuje speciální případ logistické funkce a která se využívá například při klasifikaci mezi dvěma třídami. Graf této funkce lze vidět na obrázku 1.3b. Sigmoida se dá zapsat pomocí následujícího vzorce, kde parametr λ udává sklon funkce.

$$f(x) = \frac{1}{1 + e^{-\lambda x}}$$

1.1.1.3 Softmax

Posledním uvedeným typem aktivační funkce je funkce Softmax. Funkce Softmax je taktéž jako sigmoida typ logistické funkce. V tomto případě se však používá pro multinominální logistickou regresi. V své podstatě smrskne K-dimenzionální vektor x reálných čísel do K-dimenzionálního vektoru y reálných čísel v intervalu <0, 1>, které dohromady dávají součet 1. Vektor y potom obsahuje hodnoty, které reprezentují pravděpodobnosti, že vstupní prvek patří do určité třídy.

Softmax lze popsat následující rovnicí:

$$y_k = \frac{e^{x_k}}{\sum_{j=1}^{K} e^{x_j}}$$

1.2 Učení

Chování neuronové sítě je kromě samostatné struktury určeno podle množiny vah a prahů, kterou každý neuron má. Upravování těchto hodnot způsobem, aby výstup odpovídal tomu, čemu chceme, představuje samotné učení sítě.

Většina algoritmů, které se používají k učení umělých neuronových sítí, používají nějakou formu klesání podle gradientu. Kromě toho se k učení dá ještě využít například evolučních metod.

Gradientní učení umělých neuronových sítí na datasetu probíhá iterativně a celý proces učení se dá rozdělit do epoch. Epocha představuje dobu, za kterou se síť setkala s každým vzorkem z datasetu alespoň jednou. Epocha se dá dále rozdělit do iterací, které vždy po svém doběhnutí aktualizují množinu vah a prahů. Během jedné epochy může dojít k mnoha iteracím.

Počet iterací je ovlivněn počtem vzorků v dávce, které v rámci jedné iterace projdou sítí. Dávka může obsahovat celý dataset nebo pevně daný počet vzorků. Čím větší dávka, tím přesněji bude určen gradient, ale bude to za cenu vyšších nároků na paměť.

1.2.1 Ztrátové funkce

Výsledek ztrátové funkce, jinak také chybové funkce, reprezentuje celkovou ztrátu, která označuje rozdíl mezi výstupem sítě a požadovaným výstupem. Cílem učení je tuto funkci minimalizovat. Konkrétním typem ztrátové funkce je například Mean squared error nebo cross-entropy.

1.2.1.1 Mean squared error (MSE)

Mean squared error neboli střední kvadratická odchylka se vypočítá následujícím vzorcem:

$$E = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{1.3}$$

, kde y značí vektor s požadovanými výstupními hodnotami a vektor $\hat{y}$ značí predikovaný výstup pro n vstupů.

1.2.1.2 Cross-entropy

Pokud existuje pravděpodobnostní model nad navzájem se nepřekrývajícími třídami a je potřeba změřit rozdíl mezi předpovězenou pravděpodobností pro daný vstup a známou pravděpodobností pro daný vstup, ztrátová funkce cross-entropy je pro tento případ vhodným kandidátem. Cross-entropy ztrátová funkce pro rozdělení vstupů do dvou tříd se dá zapsat tímto vzorcem:

$$E = -\sum_{i=1}^{n}(\hat{y}_i ln(y_i) + (1 - \hat{y}_i) ln(1 - y_i)) \tag{1.4}$$

, kde značení je stejné jako u funkce MSE.

1.2.2 Optimalizační metody

Úkolem optimalizačních metod je minimalizovat ztrátovou funkci. Mezi zástupci optimalizačních metod bude uveden Stochastic gradient descent (SGD) a metoda RMSprop. Kromě nich existují další typy, kde většina nějakým způsobem zrychluje SGD (Momentum, Adagrad, Adadelta,...)

1.2.2.1 Stochastic gradient descent

Klesání podle gradientu se iterativně provádí tak, že se nejprve spočítají parciální derivace chybové funkce vzhledem k vahám, které určí, jak moc se jednotlivé váhy podílejí na tvorbě výstupu a jak moc se mají změnit, aby se výstup co nejrychleji přiblížil ke správnému výsledku. Tímto způsobem se po krocích postupně dojde až do lokálního minima.

Míru učení, neboli velikost kroku směrem po gradientu, se určuje globálním parametrem α. Pokud je α nízké, konvergence bude pomalá, avšak na druhou stranu, pokud bude α příliš vysoké, dochází k divergenci.

Krok obecného klesání dle gradientu probíhá vždy po součtu chyb u všech trénovacích dat. To může být při větším počtu trénovacích dat velice náročné jak na paměť, tak i na čas. SGD postupuje tím způsobem, že nejprve poupraví váhy u jednoho vstupu (nebo u menší dávky vstupů), aby se více přibližoval správnému výstupu a pak pokračuje na dalším. Takto postupně upraví váhy u každého vstupu. Toto může n-krát zopakovat.

1.2.2.2 RMSprop

RMSprop je jeden z mnoha druhů vylepšení SGD. Rozdíl oproti SGD je ten, že v tomto případě se míra učení α podle parametrů adaptuje. Adaptuje se tím způsobem, že v každém kroku se míra učení vydělí pohybujícím se průměrem kvadrátů gradientů. Výsledkem je rychlejší konvergence oproti SGD. [7]

1.2.3 Typy učení

Způsoby učení se dají kategorizovat do tří hlavních skupin, a to na učení s učitelem, bez učitele a posilované učení.

1.2.3.1 Učení s učitelem

O učení s učitelem se mluví ještě jako o „Supervized learning“. Při této formě učení učitel zná prostředí. Má sadu trénovacích dat, kde u každého vzorku je zaznamenán jak vstup, tak i požadovaný výstup. Tato znalost se následně předává neuronové síti, aby se posléze v optimálním případě sama bez učitele dokázala vypořádat s novými, neoznačenými daty.

Učení probíhá tak, že se síť snaží minimalizovat rozdíly, mezi vstupem a výstupem. Učení většinou končí v momentě, kdy se velikost chyby dostane pod určitý práh, nebo když se chyba nezmenšila za posledních několik epoch o určitou hodnotu, či po nastaveném počtu epoch.

Jako příklad se dá použít problém, kdy je potřeba síť naučit rozpoznávat druhy ovoce podle obrázku. Nejprve se vezme trénovací množina se známými, označenými vstupy, a následně se upravuje síť, aby vracela chtěné výsledky.

Při učení je však nutné si dávat pozor na to, aby se síť na trénovacích datech nepřeučila. To by vedlo k tomu, že trénovací data dokáže síť rozpoznat perfektně, ale při aplikování sítě na dalších datech by vykazovala vysokou chybovost. Přeučení jde totiž ruku v ruce se schopností zobecňovat (generalizovat).

Přeučení lze předcházet zvýšením například počtu trénovacích dat nebo použitím nějaké z regularizačních metod, které se snaží o snížení přeučení. Jako příklad je nutné uvést metodu Dropout (DO), která se ještě objeví dále

v experimentech. Metoda Dropout [8] spočívá v náhodném vynechání poměrové části neuronů na každém trénovacím vzorku. To posléze zabraňuje tvoření skupin neuronů, které reagují stejně na stejné vzorky, ale naopak každý neuron se učí rozpoznávat příznaky nezávisle. Jinými slovy Dropout se stará o to, aby model byl odolný vůči ztrátě kousků informace.

1.2.3.2 Učení bez učitele

Jak už název napovídá při učení bez učitele neboli „Unsupervized learning“ se v učícím procesu nevyskytuje nic, co by vedlo model. Model při učení nemá k dispozici trénovací množinu, která by už byla označená. Namísto toho si model musí sám zjistit nové informace. Učení bez učitele se vyskytuje například při shlukování, kde je cílem nalézt podobnosti na vstupních datech a podle nich je potom třídit.

Jako příklad problému, na který se dá použít učení bez učitele je například rozdělení objektů na obrázku do skupin podle jejich textury.

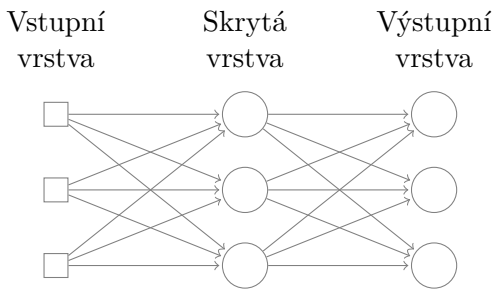
1.2.3.3 Posilované učení

Posilované učení je způsob učení pomocí odměn. Do interaktivního prostředí se vloží agent, který dokáže s prostředím pomocí akcí interagovat a získávat informace. Na základě svých aktuálních a již uskutečněných akcí dostává agent zpětnou vazbu v podobě odměn.

Agent předem neví, které akce má podniknout. Naopak postupným zjišťováním a zkoušením různých možností se snaží najít ty kroky, které vedou k celkově nejvyšší odměně. Každý další krok je soubojem dvou faktorů, kterými jsou explorace a exploatace. Na jednu stranu se agent snaží vybrat akci, která by mu měla podle získaných informací přinést nejvyšší odměnu a na druhou stranu se agent snaží nasbírat informace nové. Ty by posléze mohly vést k ještě lepšímu řešení, než jaké je dosavadní.

Jako příklad posilovaného učení může být uvedený model, který by měl být schopný hrát hru Mario. Mario je arkádová 2D hra, kde cílem je dopravit hlavní postavičku, Maria, přes překážky k unesené princezně a tu osvobodit. Agent by se například naučil, že vysokou odměnu dostane za co nejrychlejší dovršení cíle, ale postupným zkoušením dalších akcí by přišel na to, že sbíráním mincí dosáhne na odměny ještě vyšší.

Dalším příkladem užití posilovaného učení lze nalézt na burze, kde úspěšný burzovní makléř chce nakupovat akcie před tím, než jejich cena začne stoupat, a naopak je chce prodat než jejich hodnota klesne. Úkolem agenta by potom bylo naučit se na reálné burze obchodovat s akciemi za pomocí virtuálních peněz, které by reprezentovaly odměnu. Agent by se postupně naučil, jak manipulovat s akciemi, aby peníze množil a pokud možno neztrácel.



Obrázek 1.4: Dopředná neuronová síť

1.3 Architektury neuronových sítí

Způsob, jakým jsou neurony v neuronových sítích uspořádány, je pevně spjat s učícím algoritmem. V této kapitole budou nastíněny dva obecné typy sítí a několik konkrétních typů.

1.3.1 Dopředná neuronová síť

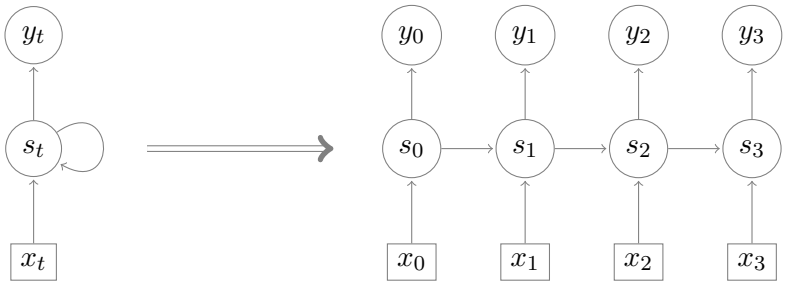
Dopředná neuronová síť je jedna z nejjednodušších sítí a jak už název napovídá, je typická tím, že signál proudí pouze směrem vpřed a netvoří se žádné smyčky. Síť se skládá z vrstev, kde v základní formě má pouze vrstvu vstupní a výstupní. Vstupní vrstva se stará pouze o promítnutí vstupů ze zdroje do další vrstvy, která je seskládaná z neuronů a která provádí nad daty výpočty.

Vícevrstvá dopředná neuronová síť navíc obsahuje vrstvy skryté, které jsou umístěny mezi vstupní a výstupní vrstvou. S každou další vrstvou umělá neuronová síť dokáže zachytit složitější významy. Jednotlivé vrstvy jsou spolu propojené stylem, že výstupy jedné vrstvy představují vstupy vrstvy následující a tímto způsobem se pokračuje až do výstupní vrstvy. Množina výstupních signálů neuronů na výstupní vrstvě reprezentuje celkovou odpověď sítě na data poskytnutá vstupní vrstvou.

Na obrázku 1.4 je vyobrazena plně propojená dopředná síť s jednou skrytou vrstvou.

1.3.1.1 Back-propagation

Síť se učí pomocí různých učících technik. Mezi nejpopulárnější patří algoritmus back-propagation (BP). Ten poté, co se vstupní hodnoty postupně dostanou přes všechny vrstvy neuronové sítě až k vrstvě výstupní, porovná výstup se správným výstupem pomocí předem zvolené ztrátové funkce. Ztrátová funkce se následně minimalizuje pomocí taktéž předem zvolené optimalizační metody. První zmínky o back-propagation lze nalézt zde [9].



Obrázek 1.5: Rekurentní neuronová síť. Na levé straně je síť v zabaleném stavu se smyčkou. Na pravé straně je síť rozvinutá do dopředné formy.

1.3.2 Rekurentní neuronová síť

Rekurentní neuronová síť má navíc oproti dopředné síti alespoň jednu zpětnou smyčku. To ji umožňuje rozhodovat se nejenom na základě aktuálního vstupu, ale i na základě kontextu získaného z minulých kroků. Do sítě vstupují sekvence, časové řady a ty i posléze ze sítě vystupují.

S využitím informace o kontextu je posléze možné naučit síť řešit problémy, které závisí na čase a na minulých stavech. Mezi těmito problémy se vyskytuje například rozpoznávání mluveného slova nebo překlad textů.

Na obrázku 1.5 je na levé straně vyobrazena rekurentní neuronová síť se smyčkou a na pravé straně její rozvinutá forma pro lepší vizualizaci. V čase t je vstupní uzel označen x_t a výstupní y_t. Skrytý rekurentní uzel je označen s_t. Rozbalením rekurze podle časové osy vzniká dopředná neuronová síť, která je vyobrazena na pravé straně obrázku 1.5. Časová osa představuje jednotlivé kroky časové řady. Jak jde na obrázku vidět vstupem skrytého uzlu s_t není jenom x_t samotný, ale vstupuje do něj i informace z minulého kroku s_{t-1}.

1.3.2.1 Back-propagation through time

Jako hlavní metodu učení u rekurentních neuronových sítí se může uvést algoritmus back-propagation through time (BPTT). Algoritmus BPTT funguje na stejném principu jako algoritmus back-propagation popsaný v sekci 1.3.1.1. Aby se však jeho základní forma mohla použít, je nutné nejprve s rekurentní neuronovou sítí pracovat v její rozvinuté podobě, kdy se s ní může pracovat jako s dopřednou sítí. [10]

1.3.3 Vanishing/Exploding gradient

Problémem u učících metod založených na klesání gradientu je ten, že vývoj učení exponenciálně závisí na rozsahu nastavených vah. Z toho vyplývá, že při zpětném propagování chyby do sítě gradient má tendenci mizet nebo naopak explodovat. Podrobnější informace je možné nalézt zde [30].

Z tohoto důvodu má síť problém se zapamatováním starších informací. Například při zpracování textů síť na jednu stranu dokáže správně pochytit posledních několik slov, ale už nedokáže pokrýt to, co se psalo o odstavec dříve.

Na tento problém přišla odpověď v podobě paměťových buněk LSTM, které jsou popsány v další kapitole.

1.3.4 LSTM

Long short term memory (LSTM) jednotka byla v roce 1997 navržena výzkumníky S. Hochreiterem a J. Schmidhuberem jako odpověď na problém mizejícího gradientu[31]. V současnosti je síť LSTM jednou z nejpoužívanějších sítí.

LSTM je rekurentní neuronová síť vyobrazena na obrázku 1.6, která je vybavená speciálním uzávěrovým mechanismem, který ovládá přístup k vnitřnímu stavu buňky. Jelikož uzávěry můžou bránit zbytku sítě, aby upravoval obsah vnitřního stavu buňky, při propagování chyby zpět do sítě gradient jen tak lehce nemizí oproti obyčejným RNN.

Vnitřní stav LSTM v každém kroku určuje množství faktorů, které buď k aktuálnímu stavu přidávají informace nebo ubírají pomocí zmiňovaných uzávěrů. Uzávěry jsou implementovány pomocí sigmoidy, jejíž výsledek v rozmezí 0–1 určuje, jestli danou informaci pustí nebo nepustí dál. Nula označuje stav, kdy se nemá dál pustit nic a na druhé straně jednička značí, že se má pustit vše.

První uzávěr f_i^t v čase t a buňce i neboli „forget gate“ určuje, co za informace se vyhodí z vnitřního stavu buňky. Vypočítá se podle následující rovnice:

$$f_i^t = \sigma(b_i^f + \sum_j U_{i,j}^f x_j^t + \sum_j W_{i,j}^f h_j^{t-1}) \tag{1.5}$$

kde σ značí sigmoidu, x^t je aktuální vstupní vektor a h^t je aktuální vektor skryté vrstvy, který obsahuje výstup všech LSTM buněk. Prahová hodnota je značená b^f, U^f představuje vstupní váhy a W^f představuje váhy mezi rekurentními stavy.

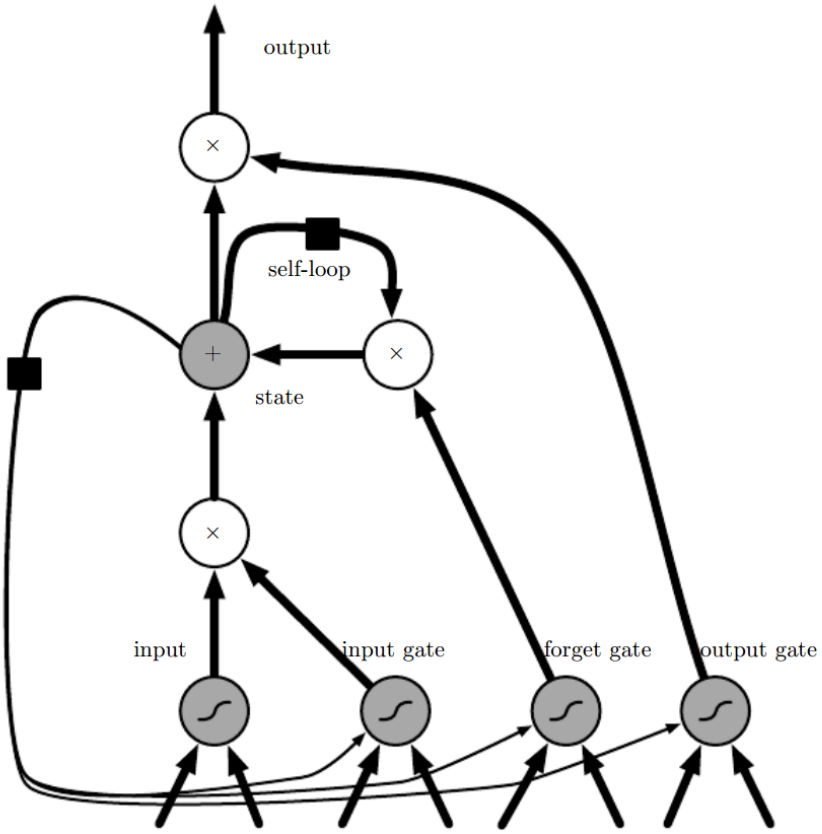
Druhý uzávěr g_i^t neboli „input gate“ se vypočítá podobně jako uzávěr první, avšak obsahuje vlastní množinu parametrů:

$$g_i^t = \sigma(b_i^g + \sum_j U_{i,j}^g x_j^t + \sum_j W_{i,j}^g h_j^{t-1}) \tag{1.6}$$

Tento uzávěr se podílí na tom, co za informace pustí dál, aby se uložily k vnitřnímu stavu.

Vnitřní stav s_i^t se posléze aktualizuje následující rovnicí, do které kromě uzávěrů f_i^t a g_i^t vstupuje předchozí vnitřní stav s_i^{t-1}.

$$s_i^t = f_i^t * s_i^{t-1} + g_i^t \sigma(b_i + \sum_j U_{i,j} x_j^t + \sum_j W_{i,j} h_j^{t-1}) \tag{1.7}$$



Obrázek 1.6: LSTM buňka [29]

Výstup LSTM buňky je ovlivňován posledním výstupním uzávěrem q_i^t neboli „output gate“, který určí, které části výstupu pustí dál. Rovnice výstupního uzávěru má stejnou podobu jako u předchozích dvou:

$$q_i^t = \sigma(b_i^q + \sum_j U_{i,j}^q x_j^t + \sum_j W_{i,j}^q h_j^{t-1}) \tag{1.8}$$

Samotný výstup se vypočítá pomocí následující rovnice:

$$h_i^t = tanh(s_i^t) q_i^t \tag{1.9}$$

[29]

LSTM rekurentní neuronové sítě budou použity v experimentech.

1.3.5 GRU

Gated Recurrent Unit (GRU) se objevila v roce 2014 a jedná se o obdobou výše popsané rekurentní neuronové sítě LSTM. Podobně jako to bylo u LSTM GRU obsahuje taktéž uzávěry, které modulují tok informací napříč jednotkou.

Na rozdíl od LSTM buněk GRU obsahuje uzávěry jen dva. GRU jednotka je vyobrazena na obrázku 1.7.

Prvním uzávěrem z_i^t v čase t a buňce i je tzv. „update gate“ a ten kontroluje, kolik informace se má zapomenout z předchozího stavu a kolik nové se má přidat. Rovnice uzávěru vypadá takto:

$$z_i^t = \sigma(b_i^z + \sum_j U_{i,j}^z x_j^t + \sum_j W_{i,j}^z h_j^{t-1}) \tag{1.10}$$

Značení v rovnici je stejné jako značení v rovnicích u LSTM 1.3.4.

Druhý uzávěr r_i^t neboli „reset gate“ pokud se přiblíží k 0, tak se buňka vyresetuje a bude se chovat tak, jako se chovala nenaučená na začátku trénování. Opět má rovnici podobnou předchozím uzávěrům s jediným rozdílem ve vlastních parametrech.

$$r_i^t = \sigma(b_i^r + \sum_j U_{i,j}^r x_j^t + \sum_j W_{i,j}^r h_j^{t-1}) \tag{1.11}$$

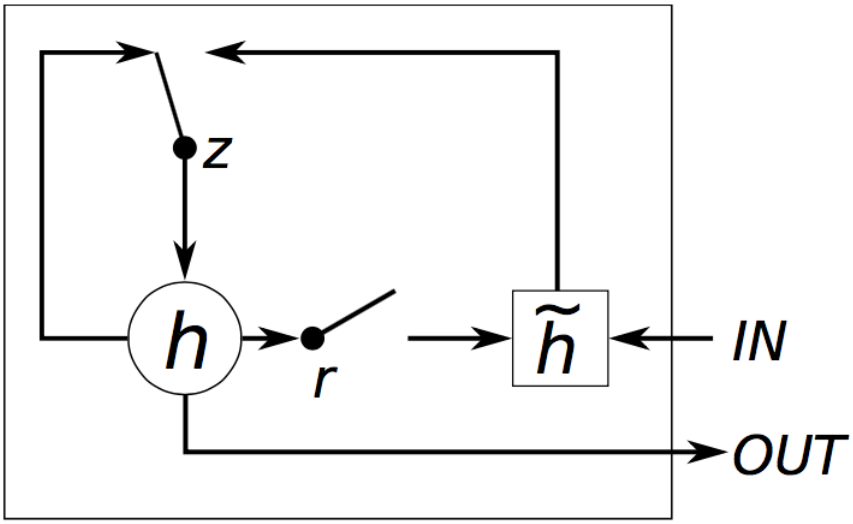
Výsledný stav se vypočítá pomocí následující rovnice:

$$h_i^t = (1 - z_i^t) h_i^{t-1} + z_i^t \widetilde{h}_i^t, \tag{1.12}$$

kde h_i^t a $\widetilde{h}_i^t$ odpovídají předchozímu obsahu paměti a novému kandidátu na obsah paměti.

$$\widetilde{h}_i^t = tanh(\sum_j U_j x_j^t + r_i^t \odot \sum_j W_j h_j^{t-1}) \tag{1.13}$$

Na rozdíl od LSTM GRU neobsahuje výstupní uzávěr, a tudíž v každém kroku poskytuje veškerý svůj obsah paměti do sítě. [32]



Obrázek 1.7: GRU buňka [32]

Kapitola 2

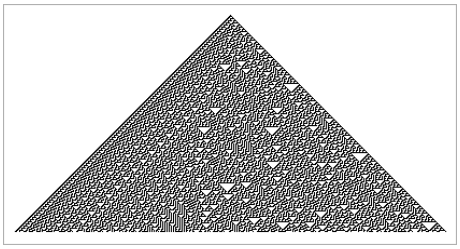
Analýza

Každý projekt, který se někdy věnoval generování hudby se vždy musel rozhodnout jakým způsobem bude hudbu generovat a jak bude vypadat výstup. Způsobů generování hudby je mnoho. V této kapitole budou zmíněny projekty, které generují hudbu nejenom pomocí neuronových sítí, ale i pomocí celulárních automatů, evolučních algoritmů i Markovských řetězců. Avšak na druhé straně hudba se dá rozumně zaznamenat pouze pomocí dvou způsobů, a to pomocí not nebo přímo zvukových vln. To bude také klíč, podle kterého bude kapitola rozdělena.

Notový zápis má pevnou strukturu a z větší části se ukládá ve formátu MIDI, kam se dá zaznamenat frekvence tónu, jeho délka, intenzita i jakým nástrojem se má tón zahrát. Následně při přehrávání MIDI souboru na zvukové kartě jsou všechny tóny přesně zahrané bez jakýchkoliv nepřesností. V přehrávání se nevyskytuje žádný šum.

Noty se kromě MIDI formátu zaznamenávají i v dalších formátech, které stojí za zmínění, jelikož několik projektů na této notaci stavělo. Jeden způsob je tzv. ABC notace. V základě tato notace používá písmena od a do G, které reprezentují noty a další znaky, které symbolizují délku noty nebo vertikální posunutí noty. Projekty potom postupovaly při generaci tím způsobem, že tyto symboly generovaly jako text.

Při generování zvukových vln se převážně postupuje tak, že se vezmou záznamy skladeb zahrané umělcem a z těch se potom síť učí. Proto při generování zvukových vln se při přehrávání vyskytují i zvuky, které nemají se samotnou skladbou přímou spojitost (šoupání židle). Oproti notovému zápisu má generování zvukových vln tu výhodu, že se ve výstupu objevuje osobitost umělce.



Obrázek 2.1: Příklad struktury sestavené celulárním automatem [12]

2.1 Současná řešení generující notový zápis

2.1.1 The Illiac Suite

Prvním zástupcem projektu, který byl zaměřen na generování hudby pomocí notového zápisu se datuje k roku 1955. V tomto roce postdoktorální student Lejaren Hiller a kolega Leonard Isaacson naprogramovali univerzitní počítač Illiac tak, aby skládal hudbu. První dílo vytvořené počítačem Illiac bylo dokončeno v roce 1957 pod názvem Illiac Suite for String Quartet. Jedná se o jeden z prvních projektů na vytvoření hudby pomocí elektronického počítače vůbec.

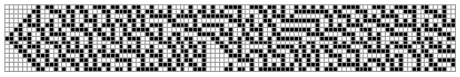
Počítač fungoval na základě naprogramované množiny pravidel, pomocí které generoval náhodná čísla, která určovala všechny potřebné hudební elementy. Určovala, jaký má následovat tón, délku tónu i nástroj, který má tón zahrát. [11]

2.1.2 WolframTones

Druhým projektem, který generuje notový zápis je projekt s názvem WolframTones. Ten staví na myšlence prezentované v 80. letech 19. století Stephenem Wolframem v knize „A New Kind of Science“[13], kde uvedl, že v počítačovém světě i ty nejjednodušší pravidla můžou tvořit složitou strukturu. Tato myšlenka vznikala v rámci Wolframových experimentů na systémech typu jednodimenzionálního celulárního automatu.

Jednodimenzionální celulární automat funguje na principu řádky s bílými a černými buňkami a množiny pravidel. Pravidla určují, kde má následující řádka mít bílé buňky a kde černé, a to v závislosti na rozložení barev mezi buňkami na řádce výše. Celkem existuje 256 základních pravidel, mezi kterými jsou i pravidla, která vytvoří takovou strukturu buněk, až se zdá, že byla vytvořena náhodně (viz 2.1).

Proces tvoření hudby celulárním automatem probíhá tak, že se podobně podle výběru základních pravidel vygeneruje větší složitější struktura. Celý vytvořený obrázek se překlopí na stranu o 90° a vyjme se z něj jen pás buněk uprostřed, který potom poslouží jako ekvivalent notového zápisu. Podle výšky se každá černá buňka namapuje na příslušný tón. Výsledný zápis, který vznikl ze stejného pravidla jako obrázek výše, může vypadat takto 2.2. [12]



Obrázek 2.2: Notový zápis sestavený celulárním automatem [12]

Výhodou projektu WolframTones ve tvoření hudby je fakt, že tvoření složitější struktury za pomocí pravidel ručí za jistou pravidelnost ve struktuře, přestože se na pohled může zdát náhodná.

2.1.3 Emily Howell

Emily Howell je název počítačového programu vytvořeném profesorem hudby Davidem Copem vyučujícím na University of California, Santa Cruz. Emily Howell je projekt postavený na projektu taktéž od Davida Copa s názvem EMI (Experiments in Musical Intelligence). EMI spočíval na tom, že byl krmen skladbami v notovém zápisu a jeho hlavním cílem bylo nalézat vzory a opakovat je v různých obdobách. Zjistil, že toto opakování je možné tvořit nejen na jeho pracích, ale i na pracích světoznámých skladatelů. Tímto způsobem vznikla řada nových skladeb, které se blížily dílům, ze kterých skladba dekompozicí vznikla.

Reakce na EMI byly smíšené. Negativní hodnocení však převršily pozitivní, a tudíž se Cope rozhodl s EMI skončit a založit v roce 2003 projekt Emily Howell, který bude vycházet pouze z výtvorů EMI a nikoli ze skladeb umělců.

Proces generování not programu Emily Howell se podobá reinforcement learningu, jelikož během generování David Cope sám sedí u interaktivního rozhraní a sám hodnotí výstupy Emily, čímž se snaží Emily naučit, co se mu líbí a naopak. [14]

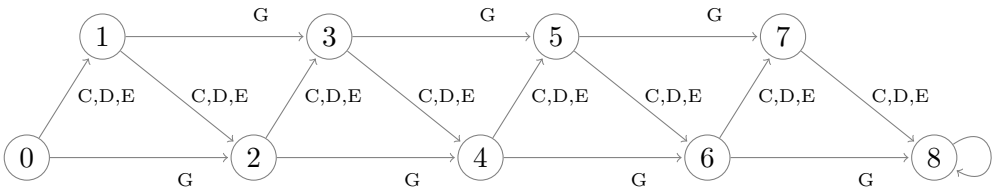
Co se týče mého osobního názoru na skladby generované programem Emily Howell, musím je označit za působivé, což může být hlavně důsledkem toho, že u Emily sedí sám David Cope a sám hodnotí výstupy.

2.1.4 Melomics

Melomics je výzkumný projekt vyvíjený na Univerzitě v Málaga ve Španělsku, který má za svůj hlavní cíl plnou automatizaci procesu skládání profesionální hudby.

Melomics na rozdíl od většiny projektů generujících hudbu se nesoustředí na imitování již zajetých stylů, ale jenom na základě naučených pravidel, jak se má hudba skládat se sám pouští do vlastní nezaujaté tvorby, které vedou ke tvoření vlastních stylů.

Na technologiích od Melomics běží počítačový cluster Iamus, což je počítačový skladatel specializovaný na klasickou hudbu současné doby. Iamus používá evoluci k vytvoření složitých hudebních struktur. Jak se populace postupně vyvíjí, mutace prováděné na genomu mění celou hudební strukturu.



Obrázek 2.3: Konečný automat, který přijímá sekvence tónů (C, D, E, G) dlouhých 8 dob [16]

Fitness funkce hodnotí jedince podle toho, jak moc odpovídají hudebním pravidlům a základním estetickým principům. Postupným vývojem se hudební struktury stávají složitějšími a vznikají kompletní skladby.

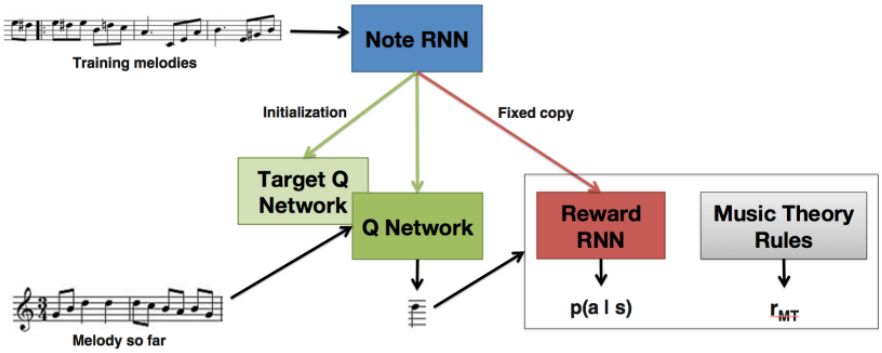
Evoluční algoritmus použitý pro tyto účely vychází z oboru biologického výzkumu s názvem „Evoluční-vývojová biologie“ (evo-devo). Ta studuje, jak funguje dynamika vývoje a snaží se určit změny fenotypů v závislosti na změnách v genotypu. Evoluční změny interpretuje jako malé mutace v genomu organismů, které ovlivňují jejich vývojové procesy, což nakonec může vést k novým vlastnostem a upraveným formám organismů. Z tohoto čerpá Melomics a svým způsobem si osvojilo efektivní nepřímé kódování, kde relativně malý genotyp dokáže popsat velký a složitý fenotyp.[15]

2.1.5 FlowComposer

FlowComposer je webová aplikace, jejímž hlavním úkolem je pomáhat uživatelům se skládáním skladeb. V současné době se tvůrci projektu FlowComposer chlubí, že se již projekt využívá v profesionálním hudebním světě.

Uživatelé specifikují hudební styl a aplikace sama vygeneruje novou kompletní skladbu. Uživatelé také mohou využít již vytvořený kus skladby a z něj seskládat zbytek. Úkolem aplikace je dodržovat tři typy pravidel při generování skladby. Základním pravidlem je, aby skladba byla rozdělena do ucelených hudebních celků udržujících rytmus. Dále se aplikace musí řídit pravidly, které zadal uživatel, a nakonec musí být splněno pravidlo, které zaručuje, aby se skladba blížila ke zvolenému stylu.

Generování je založeno na Markovově řetězci, kde náhodný průchod vyprodukuje sekvenci tónů. Markovův řetězec je omezen taktem, který způsobuje, že každá vygenerovaná sekvence má určitou délku. Stavy řetězce představují dobu v taktu a přechody mezi stavy určují hudební element. Jako příklad poslouží obrázek 2.3, kde je zobrazen konečný automat, který přijímá sekvence tónů s délkou 8 dob. Mezi dobami se přechází pomocí tónů C, D, F, které mají délku na 1 dobu, anebo tónem G, který má délku dva. [16]



Obrázek 2.4: Diagram spojení sítí Magenty pro použití RL při generování hudby[19]

2.1.6 Magenta

Posledním a zároveň z mého pohledu nejvýznamnějším z výčtu projektů generující notový zápis je projekt s názvem Magenta vytvořený týmem z Google Brain. Magenta se snaží o propojení umělců s výzkumníky a strojovým učením za takovým cílem, aby se posunuly hranice state of the art v oblasti generování hudby a umění. I z toho důvodu se Magenta stala od června 2016 open-source projektem.

Jako svou největší výzvu si Magenta předsevzala automatické generování skladby, která má zajímavý příběh a prvky překvapení. Magenta chce uspět v generování dlouhých sekvencí, které budou dávat smysl a které si s sebou budou nést myšlenku přes celou délku skladby. Magenta pro generaci not používá vlastní odrůdy RNN, a to LoopbackRNN a AttentionRNN. [18]

V současné době také začlenila do generování hudby i RL, kterým ladí výstup z RNN na základě hudební teorie. Proto, aby se z generování hudby mohl stát problém pro RL, každá nově vygenerovaná nota v nové skladbě se bere jako samostatná akce. Modul RL Tuner se stará o ladění nově generovaných skladeb. RL Tuner je rozdělen do 3 částí: Q-Network, Target Q-Network a Reward RNN. Q-Network a Target Q-Network jsou sítě potřebné pro Deep Q-Learning. Reward RNN se používá k poskytování části odměny používané při trénování modelu. Je tam z důvodu, aby se nově generovaná skladba jednoduše neřídila jenom pravidly z hudební teorie pro dosažení nejvyššího skóre, ale aby používala i naučené sekvence. RL tuner je napojený na Note RNN, která je natrénována na vstupních notách. Diagram je možné vidět na obrázku 2.4. [19]

2.2 Současná řešení generující zvukové vlny

2.2.1 GRUV

Prvním zástupcem projektů, který generuje přímo zvukový signál je projekt s názvem GRUV od inženýrů ze Stanfordské Univerzity. Jejich projekt spočíval v užití rekurentních neuronových sítí naučených na hudebních skladbách zakódovaných ve WAV formátu.

Model je reprezentován vektory, které představují jednotlivé zvukové stopy rozdělené do bloků, které jsou převedeny pomocí diskrétní Fourierovy trasformace. K experimentům využívají dvě konfigurace sítě, kde mezi skrytými vrstvami používají v jedné konfiguraci LSTM jednotky a v druhé GRU. Generování probíhá způsobem, že se vytvoří počáteční vstupní sekvence, na jejímž základě se vygeneruje nová sekvence zvukových vln, které by měly následovat. Tyto nově vygenerované sekvence posléze poslouží pro generaci nových. Po několika iteracích pak vzniká skladba. [23]

2.2.2 WaveNet

Největším zástupcem projektů, které generují přímo zvukové vlny je projekt WaveNet. WaveNet je pod záštitou skupiny Alphabet a patří do týmu Deepmind. WaveNet se nevěnuje generování hudby jakožto hlavnímu cíli, ale naopak generování hudby vzniklo jen jako vedlejší projekt.

Jako svůj hlavní úkol se WaveNet snaží o konverzi text-to-speech (TTS), tedy vytvořit mluvené slovo z textu tak, aby co nejvíc napodobovalo mluvené slovo člověka. Konkrétněji se jedná o parametrické TTS, které neskládá jednotlivé nahrané fragmenty mluveného slova dohromady, tak aby postavilo větu, ale naopak využívá parametrický model, pomocí kterého potom mluvené slovo poupravuje. Poupravuje například tón nebo zabarvení mluveného slova. WaveNet se chlubí tím, že je považován v oblasti TTS za state of the art a v hodnocení uživatelů na otázku, jak jim mluvené slovo připadá přirozené se WaveNet umístil o 0.3 bodů z 5 za ohodnocením lidských nahrávek. [21]

Generování hudby je, jak už bylo zmíněno, menší vedlejší projekt. Výzkumníci nechali WaveNet trénovat se na dvou datasetech. První dataset sestavoval 60 hodinový záznam sólo hry na piano získaného z portálu YouTube. Druhý dataset obsahoval tzv. MagnaTagATune dataset, který se skládá z 200 hodin hudby. Set je rozdělen do 29 sekundových klipů, kde každý je označen typem žánru, nástroji, rychlostí, hlasitostí a náladou, která z hudby vyzařuje.

WaveNet dokázal vytvořit nové hudební sekvence, kdy oproti algoritmům vytvářející noty šlo zachytit i dozvuky tónů, které po sobě zanechávají pianisté. WaveNet z datasetu hraní na piano vygeneroval 10 sekundové úryvky hraní, které jdou reálně zahrát člověkem. Tím je myšleno to, že ve skladbě není moment, kdy by hráč měl stisknout více kláves nebo takovou kombinaci kláves,

která by byla neproveditelná. Na druhou stranu přechody mezi klávesami jsou rychlé a časté, s čímž by už mohl mít obyčejný klavírista problém. [22]

Co se týče implementace, tak WaveNet čelil problému, kdy měl vytvořit model ze zvukových vln, které obsahují v jedné sekundě i více než 16 000 tiků. Na tento problém byla použita síť PixelCNN vytvořena van der Oordem, jednoho ze členů projektu. [20]

2.3 Notový zápis vs. zvukové vlny

Při výběru, jestli se má práce zabývat generováním notového zápisu nebo přímo zvukových vln byl kladen důraz zejména na náročnost učení a skladnost. Velikost celé skladby v notovém zápisu se pohybuje v řádu kilobajtů. Na druhé straně analogový signál nabývá tisíci násobně větších rozměrů a na délce sekundového intervalu se vyskytuje kolem 16 000 vzorků[20]. To má za následek to, že se pomaleji přenáší data na výpočetní uzly a celkově je zapotřebí více výpočetní síly než v případě notového zápisu.

Dalším faktorem výběru je usměrňování výsledků. Notový zápis udrží vygenerované sekvence v mezích a brání tak síti, aby generovala zvuky, které nemají s hudbou nic společného.

Z těchto důvodu se práce bude soustředit na generování hudby pomocí notového zápisu.

Kapitola 3

Data

Tato kapitola popisuje MIDI formát, ve kterém jsou uložené všechny skladby, které byly použity v práci a samotné zdroje.

3.1 MIDI formát

Účelem MIDI souborů je poskytnutí způsobu, jak se můžou časově označená MIDI data sdílet mezi různými programy na různých počítačích. MIDI soubory obsahují jeden nebo více toků MIDI dat, kde každá jednotlivá akce je označena časovou značkou.

Obsah MIDI souboru je strukturován jako série bloků dat, které můžou nabývat dvojího typu:

- Hlavička
  Hlavička obsahuje pár globálních vlastností, které platí pro celý soubor. Popisuje typ formátu MIDI souboru, typ jednotek, ve které se udávají tiky a počet následujících stop.

- Stopa
  Stopa obsahuje sekvence akcí, které jsou seřazené podle času.

Mezi akcemi se nachází Meta akce, které přidávají informace ke stopě. Jsou to informace popisující tempo stopy, její název nebo jakým nástrojem se má stopa hrát. Z pohledu práce nejdůležitější akce jsou MIDI akce, které svými voláními „Note on“ a „Note off“ vymezují interval, ve kterém má specifikovaný tón hrát. Těchto tónů je dohromady 128. [33]

3.2 Zdroje

Všechny skladby, které jsou v práci použity pro naučení umělých neuronových sítí pochází z kolekce 130 000 MIDI souborů, které dal dohromady uživatel

stránek Reddit.com, který vystupuje pod jménem „Midi Man“ [34]. Mezi těmito soubory jsou skladby různých žánrů od popu, přes klasickou hudbu až po elektrickou taneční hudbu.

Z této kolekce byly využity MIDI soubory, které obsahovaly klasickou hudbu pro klavír. Mezi těmito soubory se nachází skladby známých skladatelů jako například od Wolfganga Amadea Mozarta, Ludwiga van Beethovena, Josepha Franze Haydna a dalších.

Kapitola 4

Realizace

Tato práce má za cíl implementovat a otestovat různé metody založené na neuronových sítí. Jako způsob reprezentace hudby, která se bude experimenty generovat, byla vybrána notová notace, která se bude zaznamenávat do souborů ve formátu MIDI.

V této kapitole bude nejprve popsáno vývojové prostředí včetně knihoven, které byly použity při učení a generování nových skladeb a v následující části bude lehce nastíněna implementace projektu.

4.1 Vývojové prostředí

Při výběru vhodného programovacího jazyka, který by byl použit na experimenty, byl kladen zejména důraz na možnosti, jaké jazyk nabízí. Jako vítěz při porovnání jazyků (R, Octave,...) vyšel Python. V prvé řadě je pro něj vytvořeno značné množství již zaběhlých nástrojů a knihoven pro práci s umělými neuronovými sítěmi. Dalším faktorem je to, že Python je objektově orientovaný jazyk, takže sám o sobě poskytuje vysokou modularitu a přehlednost. To vede k tomu, že celý systém se podobá stavebnici a jednotlivé kousky se dají jednoduše vyměnit, což při experimentování přijde vhod. Příkladem může být snadné měnění typů modelů nebo importu dat.

V neposlední řadě nezanedbatelný vliv na výběr jazyka měl i potřebný čas k naučení se jazyka, v čemž Python nemá problém. Navíc vývojové prostředí PyCharm, které vytváří skupina JetBrains, maximálně usnadňuje vývoj v každém možném směru.

Python byl použit ve verzi 2.7, jelikož některé použité knihovny nebyly v době psaní práce ještě přizpůsobené nové verzi 3.0.

V následujících sekcích budou popsány nástroje a knihovny, které byly v práci užity.

4.1.1 virtualenv

Virtualenv je nástroj, který slouží k vytvoření izolovaného Python prostředí. Pomocí nástroje virtualenv uživatel vytvoří někde u projektu složku, která bude obsahovat vlastní instalační soubory, které nejsou sdílené s jinými virtuálními prostředími.

Tento nástroj vyřešil všechny problémy, které vznikaly tím, že na jednom stroji byly nainstalovány určité verze knihoven a některá z knihoven v projektu potřebovala jinou verzi již nainstalovaných knihoven. Díky virtualenv projekt mohl využívat jen ty knihovny, které potřeboval.

4.1.2 IPython

IPython je nástroj pro snadné experimentování v pythonu. Používá se stejně pomocí příkazů jako standardní interaktivní příkazová řádka Pythonu, avšak obsahuje několik funkcionalit navíc. Mezi ty patří například vypsaní všech podrobností o proměnné včetně dokumentace nebo jednoduché zavolání funkce shellu přímo z interaktivní příkazové řádky. [24]

4.1.3 NumPy

NumPy je jedním ze základních balíčků pro vědeckou činnost v jazyce Python. Mezi jeho hlavní přednosti patří N-dimenzionální matice objektů a sada vysoce abstrahovaných optimalizovaných matematických funkcí pro práci s těmito maticemi. V experimentech slouží zejména jako kontejner pro načtené skladby. [25]

4.1.4 python-midi

Python-midi je knihovna od Gilese Halla, která slouží ke snadné práci se soubory ve formátu MIDI. Mimo načítání a ukládání poskytuje funkce i na samostatné tvoření. [26]

4.1.5 Matplotlib

Matplotlib je knihovna na vykreslování grafů. Poskytuje rozhraní, pomocí kterého se snadno vytváří histogramy, křivky a různé další typy grafů. V práci byla knihovna použita hlavně pro zobrazení křivky minimalizace chybové funkce. [27]

4.1.6 Keras

Keras představuje jednu z nejdůležitějších knihoven pro tuto práci. Je to knihovna, která poskytuje rozhraní pro přívětivou práci s umělými neuronovými sítěmi.

Obsahuje plně konfigurovatelné samostatné moduly, které se na sebe napojují a dohromady tvoří model. Jako jednotlivé moduly se zde nachází typy vrstev umělých neuronových sítí, aktivační funkce a jiné. Každý modul navíc obsahuje značné množství parametrů, které se dají podle potřeb upravovat. [28]

4.1.7 TensorFlow

TensorFlow je open-source knihovna určená na Machine Learning. Je to knihovna původně vytvořena výzkumníky pracujících v týmu Google Brain v rámci výzkumné organizace s názvem Machine Intelligence. [35]

V této práci slouží zejména jako základ pro Keras, který je zodpovědný za low-level operace pro optimalizovanou manipulaci s vektory.

4.2 Implementace

V této sekci bude obecně popsána implementace programu použitého při trénování modelů, generování nových skladeb a aplikování metrik. Hlavní důraz byl při implementaci kladen na celkovou modularitu projektu, aby bylo možné jednoduše a efektivně experimentovat s různými nastaveními modelů.

4.2.1 Config

Třída Config představuje hlavní komponentu celého projektu. Obsahuje proměnné, které nastavují celý experiment a jsou k dispozici během celého procesu. Mezi proměnnými se nachází například hodnota, která určuje počet epoch při učení sítě, nebo počet trénovacích vzorků v dávce, které v rámci jednoho průchodu projdou sítí. Kromě toho Config obsahuje čtyřmístné číslo, které reprezentuje konkrétní typ modelu. První číslo udává velikost dávky, druhé typ optimalizační metody a třetí topologii sítě.

Ukládání výsledků několika stovek úloh paralelně na výpočetních uzlech je díky Configu přehledné. To je z důvodu, že Config za pomocí nastavených proměnných poskytuje unikátní popisek, který slouží jako název pro cílovou složku s výsledky.

Při experimentování stačí jenom upravovat hodnoty přímo v Configu a nemusí se sahat na zbytek kódu.

4.2.2 Import a export

Import MIDI souborů byl z velké části vyřešen pomocí knihovny pythonmidi 4.1.4, která umožňuje načíst jednotlivé MIDI soubory a uložit je do vektoru tiků, který reprezentuje časovou osu skladby. Jeden tik je uložen jako bitové pole se 128 prvky, kde každý prvek v poli zastupuje jeden konkrétní

tón. Jedničky v poli představují skupinu tónů, které mají v daný tik být zahrány.

Následně se ze všech tiků vytvoří slovník, který obsahuje seřazené všechny možné kombinace tónů, které se ve skladbě vyskytují. Tóny ve slovníku se posléze namapují na čísla, se kterými se bude následně pracovat. Zpětná konverze na tón proběhne až při samotném ukládání nově vygenerovaných tiků do MIDI souboru.

Při načítání MIDI souborů do vektorů tiků bylo potřeba si předem zvolit konstantní velikost časových úseků, do kterých se bude dílo dělit, aby se na jednu stranu snížily technické nároky na výpočet, ale na druhou stranu, aby se neztratilo příliš tónů. Konstantní velikost se nastavila na velikost 128.

4.2.3 Učení

Učení s učitelem probíhá tím způsobem, že se spustí skript main.py se dvěma argumenty. První určuje, jaký typ modelu se má při učení použít a druhý obsahuje cestu ke vstupním MIDI souborům. Data specifikovaná cestou se načtou a vytvoří se model podle Configu a podle rozměrů vstupních dat.

Model se vytvoří jednoduše pomocí vysoce abstrahovaného aplikačního rozhraní poskytovaného knihovnou Keras 4.1.6.

V momentě, kdy vzniká model, data k naučení v podobě vektorů tiků jsou již načtená v objektech Data. Vstupní matice, která se bude ládovat do modelu, bude nabývat rozměrů N:SEQ_LEN:1, kde N je celkový počet tiků - SEQ_LEN a SEQ_LEN je hodnota určená objektem Config – v rámci experimentů je hodnota zafixovaná na velikost 64. Číslo 1 v rozměru matice odpovídá počtu použitých kanálu. Následně se postupně přes všechny tiky posouvá klouzavé okno o velikosti SEQ_LEN a s každým posunutím se vybere sekvence tiků určená klouzavým oknem. Každý tik ze sekvence se namapuje na hodnotu ze slovníku, který byl zmíněn výše, a spolu se uloží do vektoru. Ten se následně přidá jako řádek vstupní matice.

S každým novým řádkem vstupní matice vznikne i pro něj požadovaný výstup, který představuje následující tik za sekvencí z klouzavého okna. Tik byl opět namapován na hodnotu ze slovníku. Finální jednorozměrné pole y správných výstupů se posléze transformuje do matice tříd o rozměrech N:max(y), kde třídy jsou reprezentovány pomocí one-hot encoding. Počet výstupů modelu se rovná hodnotě max(y).

Model se zkompiluje za specifikace ztrátové funkce a metody, jakou se má ztrátová funkce minimalizovat. Následně se modelu poskytnou parametry, které specifikují samotné učení (počet epoch) a učení započne. Ještě před učením se modelu nastaví, že při každém zlepšení chyby se uloží stav modelu do samostatného souboru. Ten se potom použije při generování nové skladby.

4.2.4 Generování

Po dokončení trénování modelu se vezme uložený stav modelu s nejnižší hodnotou chyby a jeho cesta se pošle jako argument při spouštění skriptu generate.py. Spolu s cestou se jako argument pošle ještě typ modelu, ze kterého uložený stav vznikl. Následně se vytvoří model a k němu několik počátečních tónů neboli náplně, která poslouží jako základ nové skladby k vygenerování. Počáteční tóny se vytvoří náhodně nebo ze specifikované části jiného MIDI souboru.

Po vytvoření modelu i náplně se můžou začít generovat tóny nové. Generování probíhá tím způsobem, že se aktuální náplň předhodí modelu a ten předpoví, který tón by měl s největší pravděpodobností nadcházet. Tento tón se vezme, připne se na konec náplně a tón na začátku se naopak odebere. Tímto způsobem se postupně vygeneruje celá skladba.

Po vygenerování dostatečného množství tónů se čísla reprezentující tóny namapují zpět na vektory tónů a uloží se do nově vytvořeného MIDI souboru.

4.2.5 Metriky

Většina použitých metrik je uložena v balíčku metrics. Kromě toho jsou další skripty na vytváření grafů a reportů z naměřených dat uložené v domovském adresáři ve složce scripts.

Při implementaci metrik se narazilo na problém, jak nějak rozumně porovnat dva MIDI soubory, jestli se na nich nachází stejná skladba nebo ne. Problém je v tom, že skladby můžou mít stejnou melodii, ale v tónině můžou být od sebe posunuté. Taktéž může nastat situace, že tóny budou ve stejné tónině, ale v jeden moment se vygenerovaná skladba opozdí o jeden tón a už originál nedohoní. U obou případů by porovnávání tón po tónu selhalo.

Místo toho byla implementována jednoduchá metrika, která porovnává poměrové zastoupení zahraných tónů ve vygenerované skladbě a originálu. Posunutí vygenerované skladby o tóninu se není třeba bát, jelikož model nemá odkud by se to naučil.

Při vykreslování grafů byla použita knihovna matplotlib 4.1.5.

Kapitola 5

Experimenty

Kapitola nejprve popíše výpočetní stroje, které se v rámci experimentů užívaly, i způsob, jakým se jednotlivé úlohy pouštěly. Posléze se přiblíží obecné probíhání experimentů, jaké nastavení se použilo a také jaké typy modelů se v experimentech objevovaly.

Následuje výčet experimentů, které obsahují obecný popis pokusu, jeho výsledky a závěrečnou diskuzi, která probere výsledky a nabídne možné vylepšení.

5.1 Výpočetní stroje

Veškeré náročné počítání probíhalo na výpočetních uzlech Metacentra. Samotné generování nových skladeb na již natrénovaných modelech probíhalo na obyčejném osobním počítači.

5.1.1 Metacentrum

Projekt Metacentrum vznikl v roce 1996 a od roku 1999 je součástí aktivit sdružení CESNET. Hlavním úkolem Metacentra je rozšiřování a koordinace národní gridové infrastruktury v České republice vhodnými propojeními stávajících a nově pořizovaných výpočetních a úložných prostředků akademické komunity v ČR. To přispívá k využívání většího množství výpočetních zdrojů pro řešení velmi náročných výpočetních úloh. Virtuální organizace MetaCentrum potom spravuje vlastní i svěřené výpočetní prostředky a datová úložiště akademických center, které jsou dostupné všem vědeckovýzkumným pracovníkům a studentům vysokých škol v České republice. [36]

Členství v Metacentru je bezplatné pro akademické pracovníky a studenty a po vyplnění přihlášky se uživatel může připojit na jeden ze vstupních uzlů. Po připojení a po zkopírování potřebných dat a zdrojových kódů se skripty pro provedení výpočtu přidávají do fronty pomocí příkazu qsub. Ve frontě potom úkony čekají do té doby, dokud se neuvolní výpočetní uzel, který splňuje

všechny konfigurační požadavky specifikované uživatelem. Mezi těmito požadavky je například počet procesorů, či jestli uzel má mít k dispozici i grafický čip.

5.1.2 Spouštění úloh na Metacentru

Metacentrum poskytuje otevřenou frontu, kam si může uživatel zařadit vlastní úlohy na vypočítání. Počet úloh na uživatele není omezen, což dovoluje paralelní vykonávání experimentů. Do fronty si uživatel vkládá úlohy pomocí qsub příkazu, kde argumenty specifikuje, jaké minimální požadavky musí výpočetní uzel splňovat, aby si mohl úlohy z fronty vzít a vypočítat. Příklad volání qsub příkazu, které potřebuje k práci 4 procesory, 12 GB paměti a který se bude vykonávat 24 hodin vypadá následovně.

```
qsub -l select=ncpus=4:mem=12gb
     -l walltime=24:00:00
     -v MYVAR="$var"
     script.sh
```

Pomocí přepínače *v* se do úlohy definované v tomto případě bash skriptem „script.sh“ vkládají argumenty.

5.2 Obecné probíhání experimentů

Každý jednotlivý experiment se spouští pomocí skriptu, který nainstaluje potřebné Python balíčky, překopíruje data nezbytná pro chod programu a následně program s příslušnými argumenty spustí. První argument reprezentuje typ modelu, který se má při experimentu v rámci jednoho úkolu použít a druhý argument popisuje cestu k datům se skladbami uloženými v MIDI formátu. Třetí argument je volitelný a představuje cestu k úložišti, kam se mají výsledky ukládat.

Experiment se spustí pro každý typ modelu i typ dat v oddělených úlohách. Po doběhnutí experimentu program po sobě zanechá složku, ve které zůstanou uloženy poslední konfigurace vah použitého modelu, které vykazovaly postupně nejlepší úspěšnosti. Kromě toho ve složce s experimentem zůstane csv soubor, který obsahuje podrobné informace o průběhu učení modelu.

Po doběhnutí všech úloh v rámci jednoho experimentu se agregují všechny složky s výsledky. Z nich se posléze po pročištění získají jen ty nejlepší.

5.2.1 Nastavení experimentů

Experimenty probíhaly na výpočetních uzlech a bylo jim pevně nastaveno cílové úložiště, kam se mají výsledky ukládat. Při počítání úloh na Metacentru si totiž jednotlivé výpočetní uzly ukládaly výsledky různě po úložištích a ty se potom musely agregovat. Někdy se však stalo, že na určitých úložištích

přetekla velikost mezivýsledků přidělenou kapacitu a úloha byla zabita. Z toho důvodu si úlohy samy specifikují cílové úložiště.

Modely se trénují po dobu 500 epoch. Při trénování model informuje množinu posluchačů o veškerém vývoji. Mezi posluchače se řadí následující:

- ModelCheckpoint

  Tento posluchač se stará o to, že v momentě, kdy se síti podaří zmenšit výstup ztrátové funkce oproti poslednímu nejlepšímu mezivýsledku, tak se aktuální nastavení vah uloží jako nový mezivýsledek do souboru ve formátu hdf5. Ten se dá potom zpětně načíst a pomocí něj generovat nové skladby.

- CSVLogger

  CSVLogger zaznamenává veškeré nastavené měření pro každou epochu během učení. Mezi měřeními je kromě ztráty například přesnost sítě.

- TensorBoard

  Posluchač TensorBoard si podobně jako CSVLogger zaznamenává vývoj učení pro každou epochu. Navíc však poskytuje možnost zobrazit si vývoj učení na interaktivních grafech.

- EarlyStopper

  Tato komponenta sleduje vývoj ztráty, a pakliže se nezlepší po 25 epoch, tak se trénování ukončí.

5.2.2 Použité modely

Faktorů, které ovlivňují úspěšnost modelů, je mnoho. Dá se upravovat struktura umělé neuronové sítě, tzn. jakým způsobem jsou vrstvy na sebe napojeny, kolik neuronů vrstva obsahuje, jestli je mezi vrstvami Dropout a pokud tam je, tak se dá dále hýbat i s jeho poměrem. Kromě toho ve vrstvách lze použít různé typy neuronů, u kterých se dá šoupat navíc s parametry.

Na druhé straně, když už je topologie sítě vytvořená, tak existují další komponenty, které lze měnit. Mezi ně patří například ztrátová funkce nebo způsob její minimalizace.

Zkrátka možností je mnoho a množina, která by obsahovala všechny typy modelů, by byla příliš rozsáhlá a bylo by velmi náročné na čas i na výpočetní prostředky spouštět každý experiment s každým modelem v samostatné úloze. Proto bylo nutné počty těchto modelů omezit.

5.2.2.1 Struktura ANN

Při experimentech byly použity různé konfigurace modelů, které vznikaly jako permutace následujících parametrů:

- Topologie sítě
  - Rekurentní vrstva → výstupní vrstva
  - Rekurentní vrstva → rekurentní vrstva stejného typu → výstupní vrstva
  - Rekurentní vrstva → DO → rekurentní vrstva stejného typu → výstupní vrstva. Dropout nabýval hodnoty 0,01 nebo 0,05
- Typ RNN
  - Obyčejná plně propojená RNN, kde výstup je napojen zpět na vstup
  - Rekurentní síť typu LSTM
  - Rekurentní síť typu GRU
- Počet rekurentních uzlů
  - 64
  - 128
  - 256
  - 512
  - 1024

Sítě se nadále budou zapisovat následujícím stylem:

- $XX(N) \rightarrow DO(d) \rightarrow XX(N)$
- $XX(N) \rightarrow XX(N)$
- $XX(N)$

, kde XX reprezentuje typ RNN, N reprezentuje počet uzlů a d poměr Dropout. Příkladem může být síť GRU(128) → GRU(128), která se skládá z dvou vrstev se 128 GRU jednotkami v každé vrstvě.

5.2.2.2 Výstupní vrstva

Výstupní vrstva obsahuje aktivační funkci Softmax a k ní příslušnou ztrátovou funkci cross-entropy pro více tříd. Důvod pro užití právě této dvojice bez dalšího experimentování vychází od C. M. Bishopa. Ten ve své knize[37] píše, že aktivační funkce výstupních neuronů spolu s typem ztrátové funkce závisí na problému, který má být řešen. Pro regresní problémy uvádí, že by se měly použít lineární výstupy spolu s počítáním chyby podle kvadratické odchylky. Pro binární klasifikaci by se měla použít logistická sigmoida v kombinaci s cross-entropy ztrátovou funkcí a pro vícetřídní klasifikaci by se měla

Tabulka 5.1: Porovnání RMSprop a SGD s měnící se mírou učení α a velikostí dávky

Metoda	α	Velikost dávky	∅ epocha	∅ ztráta
RMSprop	0,005	128	188	0,0209
RMSprop	0,005	256	186,5	0,02415
SGD	0,1	64	80,5	0,05155
RMSprop	0,005	64	84	0,05535
SGD	0,05	32	75,5	0,0648

použít kombinace aktivační funkce Softmax s cross-entropy ztrátovou funkcí. V tomto případě se jedná o vícetřídní klasifikaci, jelikož každému vstupu se přiřadí jeden tón z množiny všech zahraných tónů, a proto byla použita tato kombinace aktivační funkce Softmax a ztrátové funkce vícetřídní cross-entropy.

5.2.2.3 Optimalizační metoda a velikost dávky (batch size)

O výběru optimalizační metody rozhodovala nejen schopnost dostatečné konvergence, ale taky její rychlost. Vybíralo se z optimalizačních metod SGD a RMSprop uvedených v podkapitole 1.2.2. Kromě samotných metod se upravovala i míra učení α, která nabývala postupně hodnot $\alpha = \{0{,}1;\ 0{,}01;\ 0{,}05;\ 0{,}001;\ 0{,}005\}$.

Paralelně při výběru optimalizační metody probíhal i výběr velikosti dávky vzorků, kterou se síť bude učit během jedné iterace. Počet vzorků v dávce nabýval velikostí 2^b, kde $b = 0..9$.

V jednom běhu se postupně modely snažily přeučit jedno dílo od Mozarta.Všechny kombinace optimalizačních metod, jejich míry učení a velikostí dávek se vyzkoušely na dvou konkrétních typech ANN:

- GRU(128) → GRU(128)
- LSTM(128) → LSTM(128)

Výsledky u dvou rozdílných topologií se vzaly a udělal se z nich průměr. Nejlepších pět kombinací optimalizační funkce, míry učení a velikosti dávky je zobrazeno v tabulce 5.1.

Jako nejlepší kombinace se ukázalo spojení optimalizační metody RMSprop s mírou učení nastavenou na 0,005 a k tomu velikost dávky při učení rovna 128 prvkům. Toto nastavení se bude používat v dalších experimentech.

5.3 Vygenerování skladby 1:1

Prvním experimentem je naučit síť jednu konkrétní skladbu tím způsobem, aby po předhození prvních několika tónů byla síť schopná dogenerovat zbytek skladby, který by byl identický s originálem.

Experiment se bude provádět na všech typech naimplementovaných modelů a pustí se na skladbách od různých umělců. Ve výsledcích se bude zkoumat, jaká síť si vedla nejlépe a kteří autoři mají takové skladby, které jsou pro umělé neuronové sítě snadnější k naučení než jiné. Taktéž se bude zkoumat samotná schopnost vygenerování skladby totožné s originálem. Jako počáteční náplň pro generování bude použitý začátek skladby.

Identita vygenerované skladby s originální se bude porovnávat pomocí poměru tiků, kde byl zahrán určitý tón vůči všem tikům ve skladbě. Tyto poměry se vypočítají pro každý ze 128 tónů jak pro originální skladbu, kterou se model snažil naučit, tak i pro nově vygenerovanou skladbu. Poměry obou skladeb se vykreslí do jednoho grafu a spočítá se, jak moc se překrývají, tzn. jak moc jsou poměry podobné. Výsledné číslo bude popisovat identitu vygenerované skladby.

Porovnávat skladby tón po tónu na časové ose řekne méně o podobnosti skladeb než technika výše. Ano, pokud by byly skladby identické od začátku do určitého tiku v budoucnosti a měřila by se uplynulá doba vs. celková doba skladby jako výsledný poměr, tak by to dávalo smysl. Z poslouchání vygenerovaných skladeb však vyšlo najevo, že přestože se skladby ze značné míry podobají, sem tam se stane, že ustřelí pár nesouvislých tónů. To má za výsledek to, že kdyby se měřila identita tón po tónu, tak by na těchto skladbách selhala.

Každý model se zkusil naučit skladbu od tří autorů a dohromady proběhlo 165 experimentů v prvním běhu a 15 experimentů v druhém běhu. V prvním běhu se vybralo nejlepších 5 modelů pro každou skladbu a s těmito modely se experimenty spustily znovu, aby se alespoň částečně odstranil důsledek náhodného navolení vah při startu učení. Výsledky se zprůměrovaly a sepsaly do tabulek. U dvou nejlepších modelů pro každou skladbu je zobrazen i vývoj minimalizace ztrátové funkce na příslušných grafech.

5.3.1 Použitá data

Modely se pokusily naučit jednu náhodně vybranou skladbu od W. A. Mozarta, L. van Beethovena a F. J. Haydna a z ní následně vytvořit její identickou kopii. Následující tabulka 5.2 popisuje celkový počet tiků skladeb a unikátních kombinací stisknutých kláves v jednom tiku.

Tabulka 5.2: Použitá data v rámci generování identické skladby

Umělec	Celkový počet tiků	Celkový počet kombinací
W. A. Mozart	2984	316
L. van Beethoven	3196	763
F. J. Haydn	368	78

5.3.2 Výsledky

Experiment se provedl na náhodně vybrané skladbě od Wolfganga Amadea Mozarta, Ludwiga van Beethovena a Franze Josepha Haydna ve dvou bězích. Všechny použité skladby jsou přiložené v MIDI formátu na CD.

5.3.2.1 Wolfgang Amadeus Mozart

Úspěšné z hlediska stabilního minimalizování chybové funkce se ukázala skladba od Mozarta, kde v tabulce 5.3 se všechny uvedené modely dostaly pod hranici ztráty 0,06. Na grafu 5.1 je možné vidět vývoj ztráty průměrně nejlépe konvergujícího modelu v jeho obou bězích. Tam lze pozorovat to, že v rámci obou běhů úspěšně konverguje k minimální chybě.

Tabulka 5.3: Úspěšnost modelů při učení skladby od Wolfganga Amadea Mozarta

Popis RNN	ø chyba	ø epocha
LSTM(128) → LSTM(128)	0,0155	221
LSTM(64) → DO(0,05) → LSTM(64)	0,0234	264,5
GRU(64) → DO(0,01) → GRU(64)	0,0268	215,5
LSTM(256)	0,0305	126,5
GRU(128) → DO(0,01) → GRU(128)	0,0561	217

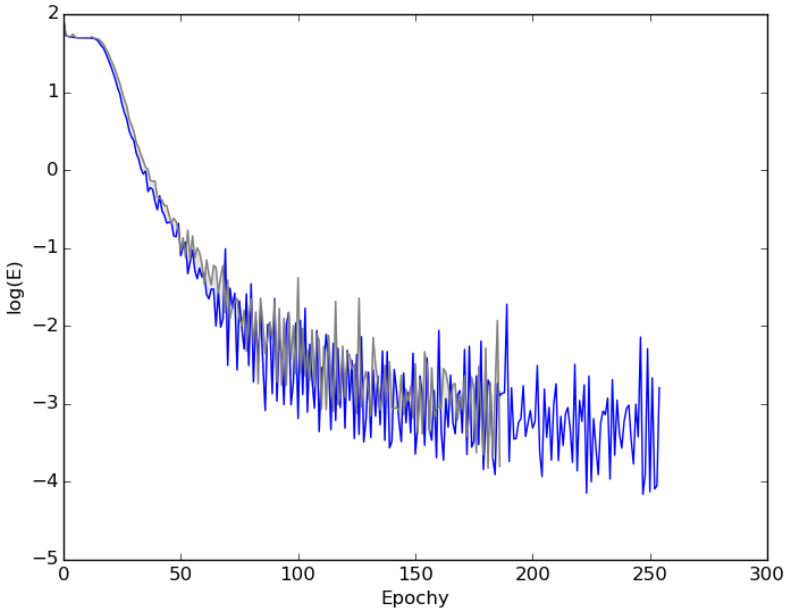
Zajímavým výsledkem experimentu je obsazení tří míst z pěti modely, které obsahují regularizační metodu Dropout na vstup druhé rekurentní neuronové vrstvy. Dropout, jak už bylo zmíněno v kapitole popisující učení s učitelem 1.2.3.1, má sloužit k tomu, aby se síť nepřeučila. To však bylo naopak cílem tohoto pokusu. Jako důvod dobrých výsledků způsobených aplikováním metody Dropout, je možná to, že díky tomu se síť více naučila obecné vlastnosti skladby a nelpěla tolik na jejich nepravidelnostech. To potom vedlo k menší globální chybě.

Poslední graf 5.2 popisuje úspěšnost experimentu na naučení skladby od Wolfganga Amadea Mozarta a její následného vygenerování. Na tomto grafu lze pozorovat vcelku uspokojivý výsledek rovný 83 %.

5.3.2.2 Ludwig van Beethoven

Úspěšnost modelů se u skladby Ludwiga van Beethoven objevila podobná jako u skladeb Mozarta. Taktéž se v tabulce 5.4 objevují modely s regularizační technikou DO. Zajímavým fenoménem je absence ANN, které by obsahovaly více jak 256 uzlů ve vrstvě. Zdálo by se, že více uzlů poskytne síti více prostoru na naučení, avšak v tomto případě se nestalo.

Na grafu 5.3 je zobrazen vývoj minimalizace ztrátové funkce na nejlepším z modelů ve dvou bězích. Ten obsahoval dvě vrstvy se 128 GRU uzly a DO na



Obrázek 5.1: Vývoj chyby dvou běhů modelu LSTM(128) → LSTM(128), který se snažil přeučit skladbu od Wolfganga Amadea Mozarta

vstupu druhé vrstvy. V horším ze dvou běhů lze pozorovat, že i přestože učení trvalo skoro dvojnásobně více epoch, tak se nepodařilo najít lepší minimum než u lepšího z běhů.

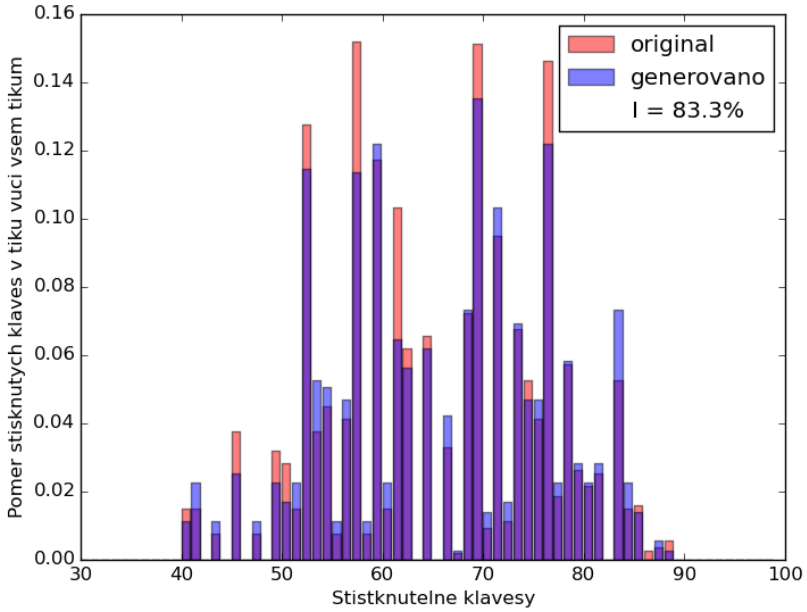
Tabulka 5.4: Úspěšnost modelů při učení skladby od Ludwiga van Beethovena

Popis RNN	ø chyba	ø epocha
GRU(128) → DO(0,05) → GRU(128)	0,0334	128
LSTM(128) → DO(0,01) → LSTM(128)	0,0392	216
LSTM(128) → LSTM(128)	0,0483	192
GRU(64) → DO(0,01) → GRU(64)	0,0684	250
GRU(64) → DO(0,05) → GRU(64)	0,0729	332,5

Podobně jako u pokusu vygenerovat identickou skladbu od Mozarta tak i při generování identické skladby od Beethovena model překročil hranici 80 % identického poměru zahraných tónů vůči originálu.

5.3.2.3 Franz Joseph Haydn

Výsledky na skladbě od Franze Josepha Haydna se liší od prvních dvou. Je to zejména v tabulce 5.5 s nejlépe vycházejícími modely. První 4 místa zaujaly



Obrázek 5.2: Porovnání distribuce tónů originální skladby od Wolfganga Amadea Mozarta a vygenerované skladby

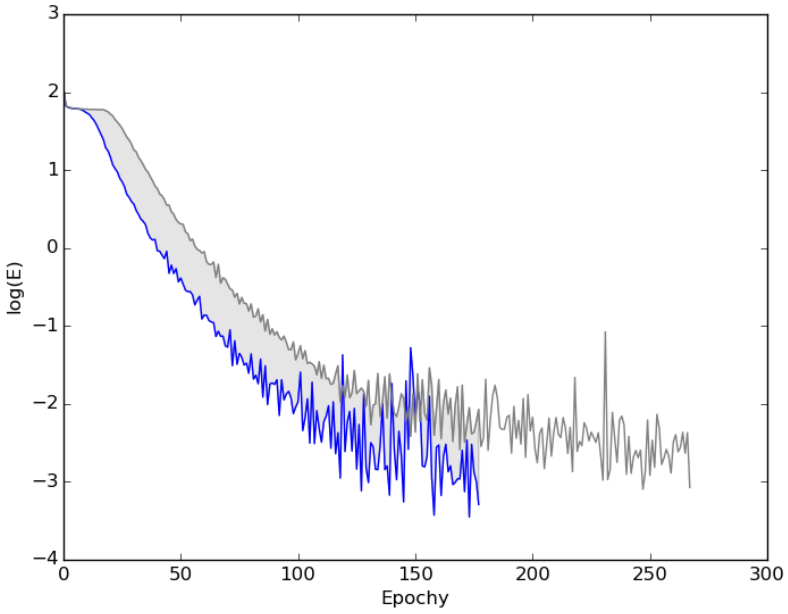
modely, které neobsahují DO. Poslední model ho obsahuje a podle průměrné chyby 2,1 jde vidět, že se do top nejlepších modelů dostal náhodou.

Vývoj během dvou běhů nejlepšího z modelů je vyobrazen na grafu 5.5. Oproti předchozím dvěma grafům je zde vidět mnohem silnější oscilace.

Tabulka 5.5: Úspěšnost modelů při učení skladby od Franze Josepha Haydna

Popis RNN	ø chyba	ø epocha
GRU(64) → GRU(64)	0,0162	457
GRU(512)	0,0195	189,5
GRU(512) → GRU(512)	0,02	331
LSTM(512)	0,0225	413
LSTM(128) → DO(0,05) → LSTM(128)	2,1014	271,5

Nejzajímavějším výsledkem z tohoto měření je nejhorší výsledek, co se týče schopnosti vygenerovat identickou skladbu. Na grafu 5.6 je možné vidět výsledek pouze 75 %.



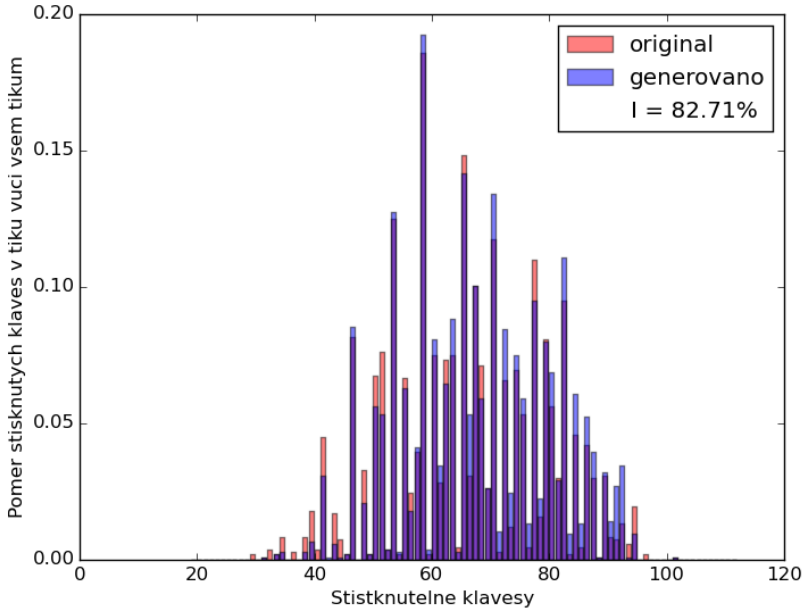
Obrázek 5.3: Vývoj chyby dvou běhů modelu GRU(128) → DO(0,05) → GRU(128), který se snažil přeučit skladbu od Ludwiga van Beethovena

5.3.3 Diskuze

Na první pohled zaujme značná oscilace ve výsledcích, hlavně tedy při trénování sítí na skladbách od Haydna. Tyto skoky z části přisuzuji velikosti dávky vzorků, na kterých se síť učí v průběhu jednoho běhu. Pokud by se velikost dávky blížila k celkovému počtu vzorků v trénovací sadě, oscilace by se nejspíše snížila, jelikož by každý krok po gradientu byl vypočítaný pro všechny vzorky.

Z výsledků je patrno, že ani jednomu modelu se nepovedlo dostat na hranici nulové chyby. Zvyšování počtu uzlů ve vrstvě ani rozšiřování sítě o další vrstvu globálně nepřinášelo lepší výsledky. Nejspíše by se našla nějaká konfigurace modelu, která by se dokázala přeučit skladbu do takové úrovně, aby byla schopná vygenerovat naučenou skladbu 1:1. Problémem s touto sítí by však bylo, že by se dokázala naučit pouze jednu konkrétní skladbu, ale nedala by se použít jako obecné řešení. Nejspíše se zde narazilo na hranice možností LSTM/GRU uzlů, za které se už nedá dostat.

K nalezení perfektní konfigurace pro každou skladbu by se dal využít evoluční algoritmus, který by se snažil modifikovat strukturu sítě, aby dosáhla ještě nižší chyby než původní generace. Další možností, která by teoreticky byla schopná se skladbu přeučit by byla umělá rekurentní neuronová síť rozšířená o zásobník s označením StackRNN[38]. Síť StackRNN má navíc oproti



Obrázek 5.4: Porovnání distribuce tónů originální skladby od Ludwiga van Beethovena a vygenerované skladby

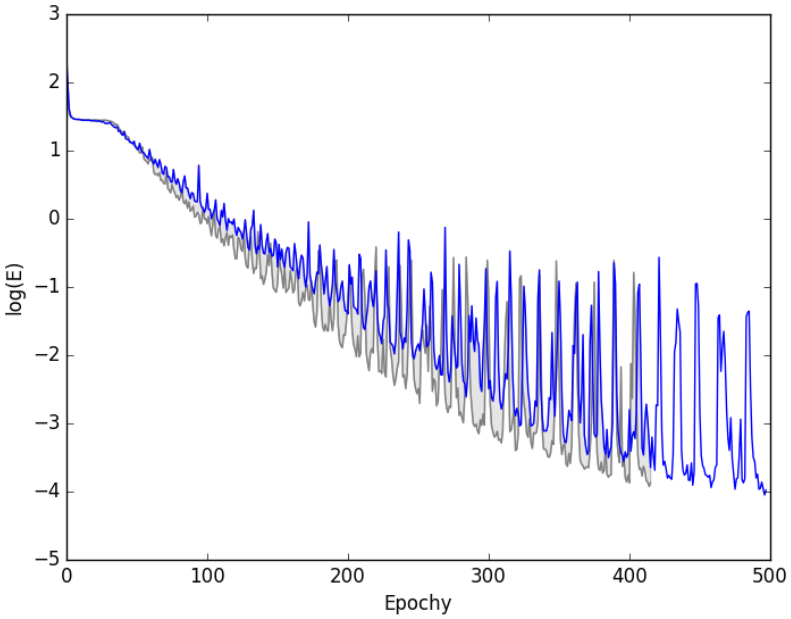
obyčejné RNN možnost naučit se ovládat nekonečnou strukturovanou paměť ve formě zásobníku nebo seznamu.

Zajímavým výsledkem bylo i nedokonalé vygenerování identické skladby Haydna, i přestože to byla skladba zdaleka nejkratší a nejméně náročná na počet kombinací zahraných tónů v jednom tiku. Při poslechovém porovnání originální a vygenerované skladby se u originálu často vyskytují pasáže, kdy autor zahraje jakousi kudrlinku. Tam místo jednoho nebo dvou tónů jich zahraje ve velmi rychlém sledu 10. Tyto pasáže se však už nenacházejí ve vygenerovaném díle. Toto dávám za vinu slabšímu výsledku na grafu 5.6.

5.4 Vygenerování nové skladby od jednoho autora

Tento experiment již nespočívá v co možná nejdokonalejším reprodukci naučených skladeb, ale naopak se snaží naučené skladby zobecnit a vytvořit skladbu novou, která by v nejlepším případě byla nerozeznatelná od profesionální tvorby. Proto se při učení nejelo až do samotného minima, kterou by optimalizační metody dokázaly najít, ale učení se vždy po dni učení přerušilo.

Opět se využijí skladby od Wolfganga Amadea Mozarta, Ludwiga van Beethovena a Franze Josepha Haydna, avšak v tomto případě se jich použije několik. Dalším rozdílem oproti předchozímu experimentu je to, že se již vů-



Obrázek 5.5: Vývoj chyby dvou běhů modelu GRU(64) → GRU(64), který se snažil přeučit skladbu od Josepha France Haydna

bec nebudou v pokusech vyskytovat modely s obyčejnými RNN, jelikož při pokusech o reprodukování skladby skončily s propastným rozdílem za modely s uzly LSTM nebo GRU.

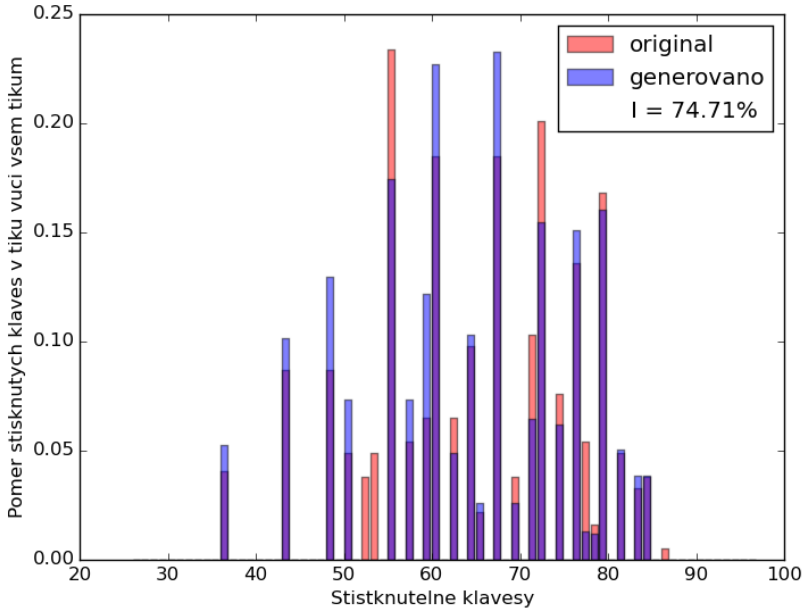
Další novinkou oproti předchozímu je možnost vygenerovat počáteční náplň nejenom z trénovacích skladeb ale i z jiných, nezařazených do trénovacího vzorku. Počáteční náplň lze tvořit i náhodnou směsí tónů.

5.4.1 Použitá data

V rámci experimentu vygenerování nové skladby od jednoho autora bylo postupně použito 34 skladeb od W. A. Mozarta, 24 od L. van Beethovena a 40 od F. J. Haydna. Při pokusech se vždy pracovalo jen se skladbami patřícími konkrétnímu autorovi. Detailnější informace o celkovém množství tiků a unikátních kombinací jsou popsány v tabulce 5.6. V tabulce je vidět, že Beethoven je v počtu unikátních kombinací stisknutých kláves v jednom tiku nejrozmanitější. Na druhém konci spektra potom stojí F. J. Haydn.

5.4.2 Výsledky

Ve výsledcích bude vyobrazení vývoje minimalizace ztrátové funkce pěti nejlepších modelů a konkrétní hodnoty napsané v tabulkách.



Obrázek 5.6: Porovnání distribuce tónů originální skladby od Josepha France Haydna a vygenerované skladby

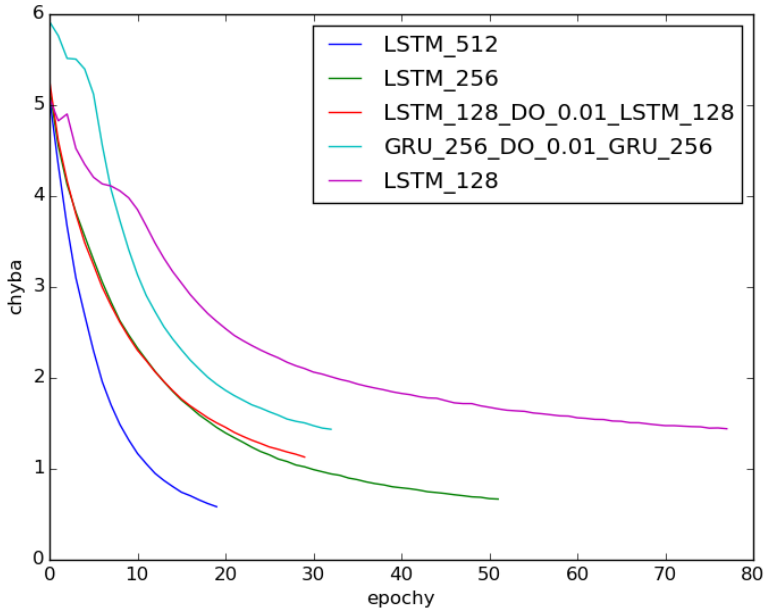
Tabulka 5.6: Použitá data v rámci generování skladby od jednoho autora

Umělec	Celkový počet tiků	Celkový počet kombinací
W. A. Mozart	104346	3503
L. van Beethoven	76851	8457
F. J. Haydn	54786	1863

Tabulka 5.7: Úspěšnost modelů při učení skladeb od Wolfganga Amadea Mozarta

Popis RNN	chyba	epocha
LSTM(512)	0,5838	20
LSTM(256)	0,6680	52
LSTM(128) → DO(0,01) → LSTM(128)	1,1298	30
GRU(256) → DO(0,01) → GRU(256)	1,4354	33
LSTM(128)	1,4412	78

Postupně na grafech 5.7, 5.8 a 5.9 lze pozorovat, že všechny modely úspěšně konvergují a konvergovaly by i dále, kdyby nebyly přerušeny. Zajímavým zjištěním je, že potřebná velikost sítí k lepšímu naučení skladeb se oproti předchozímu experimentu nezvětšila a že stále stačily i jednodušší modely pouze



Obrázek 5.7: Vývoj chyby nejlepších modelů, které se trénovaly na skladbách od Wolfganga Amadea Mozarta

s jednou rekurentní vrstvou. To potvrzují i tabulky 5.7, 5.8 a 5.9, kde se nejmenší naměřená ztráta modelů pohybuje pod hranicí 1.

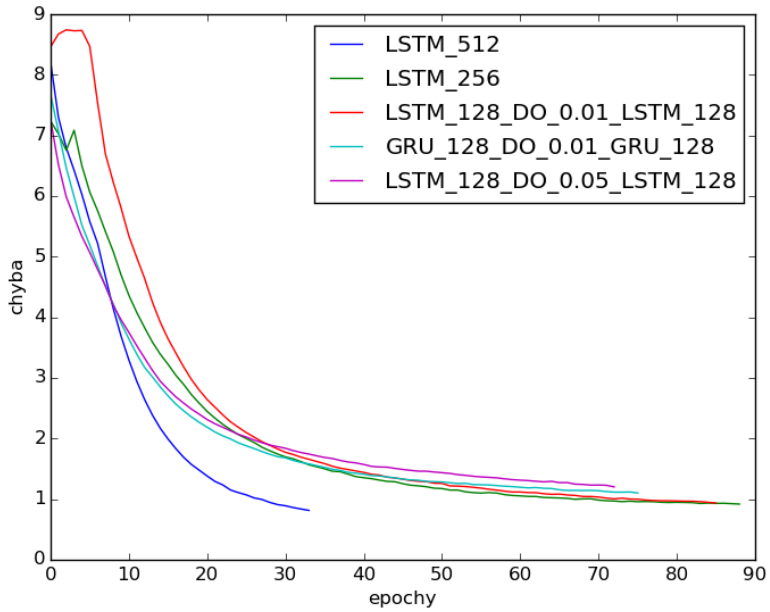
Tabulka 5.8: Úspěšnost modelů při učení skladeb od Ludwiga van Beethovena

Popis RNN	chyba	epocha
LSTM(512)	0,8147	34
LSTM(256)	0,9191	89
LSTM(128) → DO(0,01) → LSTM(128)	0,9359	86
GRU(128) → DO(0,01) → GRU(128)	1,1029	76
LSTM(128) → DO(0,05) → LSTM(128)	1,2028	73

Při generování nových skladeb byla použita náplň ze skladeb použitých v trénovacím setu i ze skladeb mimo trénovací test. Z nich byla subjektivně od každého umělce vybrána nejzajímavější a vložena do dotazníku k nezávislému ohodnocení.

5.4.3 Diskuze

Při poslechu vygenerovaných skladeb bylo zajímavé poslouchat, jak písnička sklouzává od jednoho tématu k jinému. V jeden moment přehrává tóny, které



Obrázek 5.8: Vývoj chyby nejlepších modelů, které se trénovaly na skladbách od Ludwiga van Beethovena

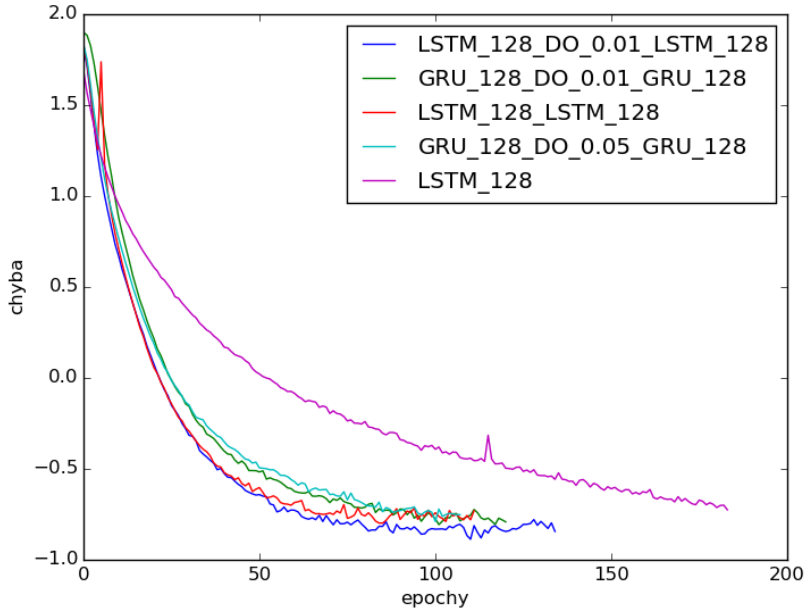
Tabulka 5.9: Úspěšnost modelů při učení skladeb od Josepha France Haydna

Popis RNN	chyba	epocha
LSTM(128) → DO(0,01) → LSTM(128)	0,4119	111
GRU(128) → DO(0,01) → GRU(128)	0,4463	102
LSTM(128) → LSTM(128)	0,4501	86
GRU(128) → DO(0,05) → GRU(128)	0,4541	101
LSTM(128)	0,4842	184

se vyskytují v jedné z trénovacích skladeb a v dalším momentu se přes několik nesouvisejících tónů přesune k jiné skladbě. Dále z poslechu vyplynulo, že model nemá tendenci sklouzávat ke stále stejným tónům. I v momentech, kdy se na chvíli vygenerovaná skladba zasekne v jednom doznívajícím tónu, se po chvilce zase rozjede, až to zní, jako by nás chtěl model napínat, co přijde dál.

Problémem však bylo, že se skladba nedokázala držet jedné myšlenky, ale neustále se měnily nálady, jako by docházelo k argumentu mezi několika lidmi.

Oproti grafům z experimentu zabývajícím se generováním identické skladby jde vidět, že míra oscilace značně ustoupila. Z toho lze usuzovat, že na samotné oscilaci se podílí i velikost vstupních dat, a tudíž jí v předchozím experimentu nejvíce trpěl pokus o vygenerování identické skladby od Haydna, která měla pouze 368 tiků.



Obrázek 5.9: Vývoj chyby nejlepších modelů, které se trénovaly na skladbách od Josepha France Haydna

5.5 Vygenerování nové skladby ze směsi skladeb od různých autorů

V tomto experimentu, jak už název napovídá, se jednotlivé modely budou snažit dostatečně se naučit na skladbách od různých autorů a posléze pomocí vlastních sil vytvořit unikátní dílo. Modely se opět nenechaly sklouznout do svého optima, ale po uplynutí časové doby se utnuly.

5.5.1 Použitá data

Při generování nových skladeb ze směsi skladeb od různých autorů se použily skladby opět od tria Mozart, Beethoven a Haydn. Celkem bylo použito 18 děl od Beethovena, 20 od Haydna a 21 od Mozarta. Celkový počet tiků skladeb a unikátních kombinací stisknutých kláves v jednom tiku je zobrazen v tabulce 5.10. V rámci jedné epochy experimentu se pak modely učily na všech zmíněných skladbách.

5.5.2 Výsledky

Jako výsledek je zobrazen graf 5.10, který obsahuje pět modelů vykazující nejlepší výsledky, tak jako to bylo v předcházejícím experimentu. Konkrétní

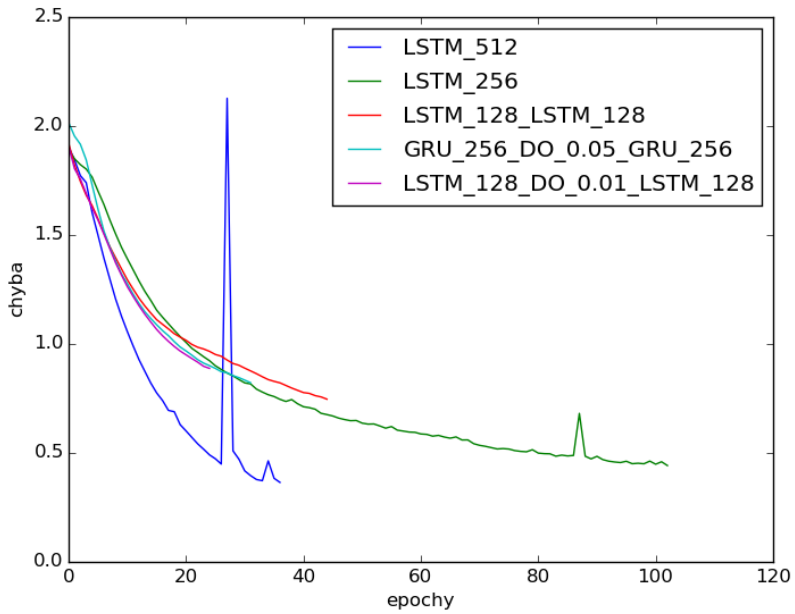
Tabulka 5.10: Použitá data v rámci generování skladby ze směsi skladeb od různých autorů

Umělec	Celkový počet tiků	Celkový počet kombinací
W. A. Mozart	61965	2648
L. van Beethoven	52761	7114
F. J. Haydn	13758	521

hodnoty těchto modelů jsou zmíněny v tabulce 5.11.

Tabulka 5.11: Úspěšnost modelů při učení skladeb od Haydna, Mozarta a Beethovena dohromady

Popis RNN	chyba	epocha
LSTM(512)	1,4344	38
LSTM(256)	1,5549	104
LSTM(128) → LSTM(128)	2,0954	46
GRU(256) → DO(0,05) → GRU(256)	2,2507	33
LSTM(128) → DO(0,01) → LSTM(128)	2,3804	26



Obrázek 5.10: Vývoj chyby nejlepších modelů, které se trénovaly na skladbách od Haydna, Mozarta a Beethovena dohromady

5.5.3 Diskuze

Opět můžeme z grafu 5.10 vypozorovat konvergenci modelů a absenci oscilace tak, jako to bylo u předchozího experimentu. Ustřelení u modelu LSTM(512) se stalo nejspíše z toho důvodu, že při klesání podle gradientu se vstoupilo mimo a ztráta tímto vystřelila vzhůru.

Při vytváření počáteční sekvence z dat, které byly součástí trénovacího setu, se stávalo, že se skladby až příliš držely naučených skladeb. Proto byly do dotazníku vloženy skladby, které vznikaly za pomocí náplně úplně od jiného autora nebo z náplně náhodně vygenerované.

V tomto experimentu se i několikrát při vytváření nových skladeb stalo, že při použití náplně od jiného autora, než na kterých byly modely trénované, se skladbě podařilo se zaseknout na dvou neustále se opakujících se tónech. Přisuzuji to tomu, že v náplni byly použité kombinace tónů, které viděl model poprvé, a tudíž nedokázal rozmanitě pokračovat a zacyklil se.

Kapitola 6

Testování a vyhodnocení

Tato kapitola obsahuje subjektivní vyhodnocení vygenerovaných skladeb a zmínění několika vlastností, které vytvořeným skladbám chybí. Následuje seznámení se s dotazníkem, kde skupina 38 hodnotitelů oznámkovala pět vybraných skladeb.

6.1 Kritické zhodnocení

Při generování skladeb velmi záleželo na počáteční sekvenci, která se do generovacího procesu vložila a na které potom stavěl celý zbytek skladby. Někdy se povedlo vytvořit takovou počáteční sekvenci, která skladbu navodila do příjemného tempa a která držela skladbu v jednom tématu po dobu pár desítek vteřin.

Na druhou stranu pár desítek vteřin z mého pohledu nestačí. Skladbám chybí myšlenka, která by prostupovala celou skladbou. Úseky, které se hrály na začátku ani zdaleka nepřipomínají to, co se hraje ke konci.

Dále vygenerovaným dílům chybí rytmus. Někdo by to mohl nazvat jako dílo virtuóza, který popisuje souboj složitých pocitů v hlavě, nicméně mně tam rytmus a jistá pravidelnost chybí.

Osobně se domnívám, že tento program by se dal efektivně využít v tom případě, že nějaký profesionální umělec, který má již vytvořeno několik skladeb, by si chtěl nechat našeptávat nějaké nápady na další možné obraty a sekvence, které by vycházely z jeho písniček. K tomuto účelu by si naučil některý z nabízených modelů na svých skladbách a pak by mu podsouval různé počáteční sekvence a z nich by potom tvořil.

6.2 Dotazník

V zájmu objektivního ohodnocení vygenerovaných skladeb byl vytvořen dotazník, který obsahuje kromě popisu experimentu i seznam 5 skladeb, které

byly vygenerované v rámci všech proběhlých experimentů a které mi přišly osobně nejzajímavější. Skladby Neural Mozart, Neural Beethoven a Neural Haydn vzešly z pokusu o vygenerování skladby pomocí modelů naučených na skladbách od jednoho autora. Neural Mix 1 a Neural Mix 2 byly vytvořeny pomocí modelů naučených na mixu skladeb od zmíněných třech autorů.

Zpovídaná osoba každou skladbu ohodnotila na stupnici 1-10, kde hodnota 1 reprezentuje skladbu sestavenou ze směsi náhodných tónů a 10 představuje profesionální dílo. Dotazník je spolu s použitými skladbami přiložen na CD.

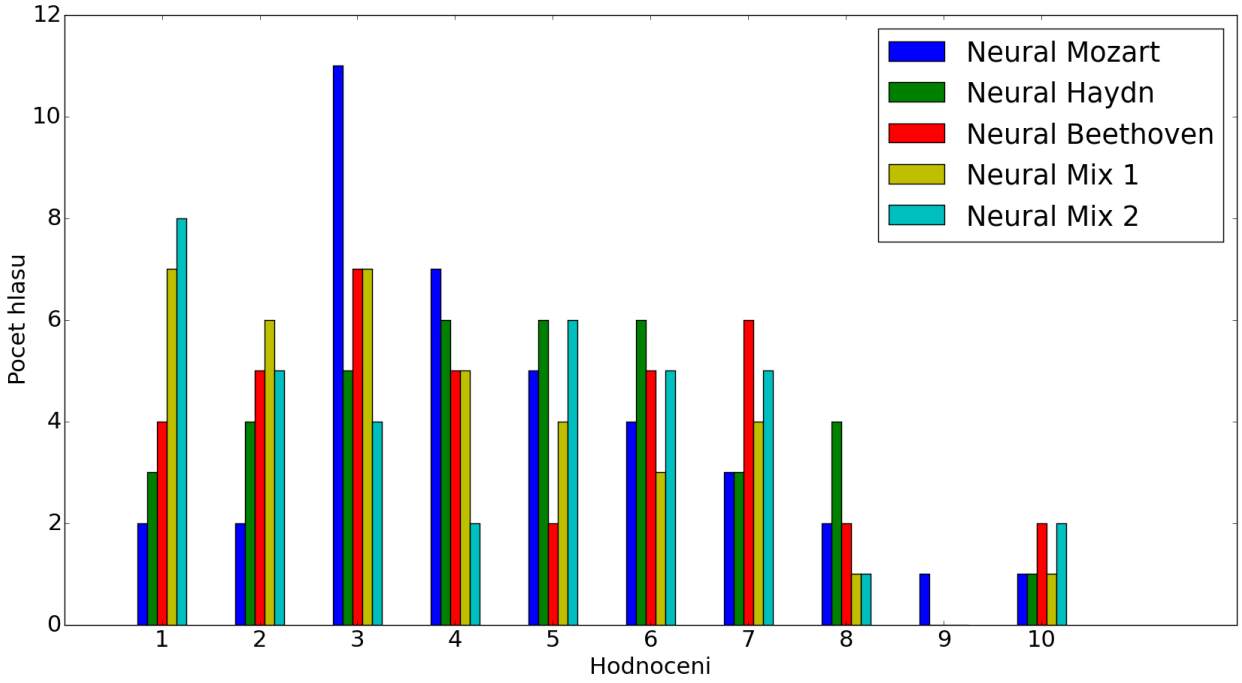
Celkem dotazník vyplnilo 38 lidí a výsledné průměrné hodnocení skladeb je popsáno v nadcházející tabulce 6.1.

Tabulka 6.1: Vyhodnocení 38 odpovědí ohledně pěti skladeb z dotazníku na škále 1-10, kde 1 reprezentuje náhodnou směs tónů a 10 profesionální dílo

Vygenerovaná skladba	Průměrné hodocení
Neural Mozart	4,53
Neural Beethoven	4,55
Neural Haydn	4,71
Neural Mix 1	3,79
Neural Mix 2	4,24

Jak jde vidět, nejvíce se lidem líbilo dílo s názvem Neural Haydn, které bylo vygenerováno pomocí modelu naučeném jenom na skladbách právě od Josepha France Haydna. Na druhou stranu nejhůře dopadl Neural Mix 1 spolu se skladbou Neural Mix 2. Zajímavé bylo pozorovat, že se hodnotící například na skladbách Neural Mix 2 a Neural Beethoven úplně neshodli a v dotazníku zavládly osobní preference. Na následujícím grafu 6.1 je zobrazeno hodnocení u všech skladeb v dotazníku.

K dotazníku byl přiložen i prostor na vlastní poznámky k experimentu. V poznámkách se objevovaly názory, že skladby postrádají myšlenku a účel. Dále, že díla nectí jednotnost tempa a základní principy harmonie. Kromě toho se objevil i komentář, ve kterém autor píše, že se mu skladby subjektivně nelíbí, ale setkal se již s řadou klasických děl, které na něj působily podobným dojmem – dle jeho názoru by bylo možné zaměnit vygenerovaná díla s některými počiny moderní vážné hudby. Vyskytly se i takové komentáře, které určité skladby pochvalovaly.



Obrázek 6.1: Hodnocení pěti skladeb v dotazníku

Závěr

V práci jsem prozkoumal výčet současných řešení včetně metod, které se věnují generování hudby ať už v podobě not nebo přímo v podobě analogového signálu. Dále jsem vytvořil program, který se za specifikace typu modelu a vstupních dat v MIDI formátu dokáže data naučit a na jejich základě posléze vygenerovat skladbu novou. Modely byly založené na rekurentních neuronových sítích používajících LSTM nebo GRU uzly.

Pomocí vytvořeného programu jsem provedl několik experimentů na klasické hudbě od Wolfganga Amadea Mozarta, Ludwiga van Beethovena a Josepha France Haydna. Nejprve jsem empiricky našel nejlepší konfigurace modelů, které se dokážou naučit skladby umělců a tyto konfigurace jsem posléze použil přímo v experimentech. První experiment spočíval v naučení konkrétní skladby a následného vytvoření jeho identické kopie. Výsledkem potom byla díla, která byla až z 83 % identická s originální skladbou. Další experimenty generovaly nové skladby.

Nově vygenerované skladby jsem následně vyhodnotil za použití dotazníku. Z dotazníku vyplynulo, že se vytvořené skladby s nejlepším hodnocením 4,71/10, zatím ještě nevyrovnají těm od profesionálních umělců.

Do budoucna by se práce dala vylepšit použitím evolučních algoritmů, pomocí kterých by bylo možné vyšlechtit takovou topologii sítě a takové nastavení parametrů, které by vykazovalo stabilně dobré výsledky pro obecně jakoukoliv skladbu v MIDI formátu. Také by mohlo být zajímavé vyzkoušet použít tzv. Generative Adversarial Networks[39] pro generování kompetitivních skladeb, kde by jedna síť skladby tvořila a druhá by určovala, jestli se dá nová skladba považovat za dílo umělce nebo ne.

Příloha A

Seznam použitých zkratek

ANN Artificial neural network

BP Back-propagation

BPTT Back-propagation through time

DO Dropout

LSTM Long short-term memory

MIDI Musical instrument digital interface

MSE Mean squared error

RL Reinforcement learning

RNN Rekurentní neuronové sítě

SGD Stochastic gradient descent

TTS Text to speech

WAV Waveform audio file format

Literatura

[1] Kubota, Taylor. *Deep learning algorithm does as well as dermatologists in identifying skin cancer* [online]. 2017 [cit. 2017-04-16]. Dostupné z http://news.stanford.edu/2017/01/25/artificial-intelligence-used-identify-skin-cancer/.

[2] The Tesla Team. *All Tesla Cars Being Produced Now Have Full Self-Driving Hardware* [online]. 2016 [cit. 2017-05-3]. Dostupné z: https://www.tesla.com/en_GB/blog/all-tesla-cars-being-produced-now-have-full-self-driving-hardware.

[3] Materna, Jiří. *Poezie umělého světa*. 2016.

[4] Gatys, Leon A., Ecker, Alexander S. a Matthias Bethge. *A Neural Algorithm of Artistic Style* [online]. 2015 [cit. 2017-04-16]. Dostupné z https://arxiv.org/pdf/1508.06576v1.pdf.

[5] Haykin, Simon S. *Neural networks and learning machines*. 3 ed. New York. ISBN 0131471392.

[6] Wikipedie. *Neuron* [online]. 2016 [cit. 2017-03-15]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Neuron.

[7] Hinton, Geoffrey E. *RMSprop: Divide the gradient by a running average of its recent magnitude* [online]. [cit. 2017-04-28]. Dostupné z: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[8] Hinton, Geoffrey E., Srivastava, Nitish, Krizhevsky, Alex, Sutskever, Ilya a Ruslan R. Salakhutdinov. *Improving neural networks by preventing co-adaptation of feature detectors* [online]. 2012 [cit. 2017-04-30]. Dostupné z: https://arxiv.org/pdf/1207.0580.pdf

[9] Rumelhart, D. E., Hinton, G. E. a R. J. Williams. *Learning Internal Representations by Error Propagation. Neurocomputing: Foundations of Research*. 1988. ISBN 0-262-01097-6.

[10] Werbos, Paul J. *Backpropagation Through Time, What It Does and How to Do It* [online]. 1990 [cit. 2017-03-15]. Dostupé z: `http://axon.cs.byu.edu/~martinez/classes/678/Papers/Werbos_BPTT.pdf`

[11] Inart 55. *History of Electroacoustic Music: The Birth of Computer Music The Illiac Suite* [online]. 2016 [cit. 2016-12-11]. Dostupné z: `http://www.personal.psu.edu/meb26/INART55/illiac_suite.html`.

[12] Wolfram Research, Inc. *HOW IT WORKS* [online]. 2016 [cit. 2016-12-11]. Dostupné z: `http://tones.wolfram.com/about/how-it-works`.

[13] Wolfram, Stephen. *A New Kind of Science 2012*. ISBN 1-57955-008-8.

[14] Cheng, Jacqui. *Virtual composer makes beautiful music—and stirs controversy: Can a computer program really generate musical compositions that are good ...* [online]. 2009 [cit. 2016-12-10]. Dostupné z: `http://arstechnica.com/science/2009/09/virtual-composer-makes-beautiful-musicand-stirs-controversy/`.

[15] Quintana, Carlos Sánchez, Arcas, Francisco Moreno, Molina, David Albarracín, Rodríguez, Jose David Fernández a Francisco J. Vico. *Melomics: a Case-Study of AI in Spain* [online]. 2013 [cit. 2017-01-27]. Dostupné z: `http://geb.uma.es/images/papers/AIMAGAZINE.pdf`.

[16] Papadopoulos, Alexandre, Roy, Pierre a François Pachet. *Assisted Lead Sheet Composition using FlowComposer* [online]. 2016 [cit. 2017-03-01]. Dostupné z `https://www.csl.sony.fr/downloads/papers/2016/roy-16b.pdf`.

[17] Google, Inc. *Magenta: Make Music and Art Using Machine Learning* [online]. 2016 [cit. 2016-12-10]. Dostupné z: `https://magenta.tensorflow.org/`.

[18] Waite, Elliot. *Generating Long-Term Structure in Songs and Stories* [online]. 2016 [cit. 2017-04-12]. Dostupné z `https://magenta.tensorflow.org/blog/2016/07/15/lookback-rnn-attention-rnn/`.

[19] Jaques, Natasha, Gu, Shixiang, Turner, Richard E. a Douglas Eck. *Tuning Recurrent Neural Networks with Reinforcement Learning* [online]. 2016 [cit. 2016-02-08]. Dostupné z: `http://arxiv.org/abs/1611.02796`.

[20] van den Oord, Aäron a kolektiv. *WaveNet: a Generative Model for Raw Audio* [online]. 2016 [cit. 2017-01-05]. Dostupné z: `https://arxiv.org/pdf/1609.03499v2.pdf`

[21] van den Oord, Aäron a Sander Dieleman. *WaveNet: a Generative Model for Raw Audio* [online]. 2016 [cit. 2017-02-05]. Dostupné z: `https://deepmind.com/blog/wavenet-generative-model-raw-audio/`.

[22] Oore, Sageev. *Human Learning What WaveNet Learned from Humans* [online]. 2016 [cit. 2017-12-04]. Dostupné z: `https://magenta.tensorflow.org/blog/2016/09/23/learning-music-from-learned-music/`.

[23] Nayebi, Aran a Matt Vitelli. *GRUV: Algorithmic Music Generation using Recurrent Neural Networks* [online]. 2015 [cit. 2017-02-04]. Dostupné z: `https://cs224d.stanford.edu/reports/NayebiAran.pdf`.

[24] Pérez, Fernando a Brian E. Granger. *IPython: a System for Interactive Scientific Computing, Computing in Science and Engineering* [online]. 2007 [cit. 2017-03-25]. Dostupné z `http://ipython.org`.

[25] van der Walt, Stéfan, Colbert, S. Chris a Gaël Varoquaux. *The NumPy Array: a Structure for Efficient Numerical Computation, Computing in Science & Engineering* [online]. 2011 [cit. 2017-03-25]. Dostupné z `http://scitation.aip.org/content/aip/journal/cise/13/2/10.1109/MCSE.2011.37`.

[26] Hall, Giles. *Python MIDI library* [online]. 2015 [cit. 2017-03-25]. Dostupné z `https://github.com/vishnubob/python-midi`.

[27] Hunter, John. *Matplotlib: a 2D graphics environment. 2007 Computing In Science & Engineering.* [cit. 2017-03-25].

[28] Chollet, François. *Keras* [online]. 2015 [cit. 2017-03-25]. Dostupné z `https://github.com/fchollet/keras`.

[29] Goodfellow, Ian, Bengio, Yoshua a Aaron Courville. *Deep Learning.* 2016 [cit. 2017-04-09]. Dostupné z `http://www.deeplearningbook.org`.

[30] Nielsen, Michael. *Why are deep neural networks hard to train?* [online]. 2017 [cit. 2017-03-25]. Dostupné z `http://neuralnetworksanddeeplearning.com/chap5.html`.

[31] Hochreiter, Sepp a Jürgen Schmidhuber. *Long Short-Term Memory.* Neural Computation, 9(8):1735–1780, 1997.

[32] Chung, Junyoung, Gulcehre, Caglar, Cho, KyungHyun a Yoshua Bengio. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling* [online]. 2014 [cit. 2017-04-09]. Dostupné z `https://arxiv.org/pdf/1412.3555.pdf`.

[33] Back, David. *Standard MIDI-File Format Spec. 1.1, updated* [online]. 2003 [cit. 2017-04-15]. Dostupné z `https://www.cs.cmu.edu/~music/cmsip/readings/Standard-MIDI-file-format-updated.pdf`.

[34] MidiMan. *The Largest MIDI Collection on the Internet, collected and sorted diligently by yours truly.* [online]. 2016 [cit. 2017-04-15]. Dostupné z `https://www.reddit.com/r/WeAreTheMusicMakers/comments/3ajwe4/the_largest_midi_collection_on_the_internet/`.

[35] Team Google Brain. *TensorFlow: a system for large-scale machine learnin* [online]. 2016 [cit. 2017-04-16]. Dostupné z `https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf`.

[36] *O MetaCentru* [online]. 2014 [cit. 2016-04-16]. Dostupné z `https://www.metacentrum.cz/cs/about/`.

[37] Bishop, Christopher M. *Pattern Recognition and Machine Learning.* Springer, 2006. ISBN 0-387-31073-8.

[38] Joulin, Armand a Tomas Mikolov. *Inferring Algorithmic Patterns with Stack-Augmented Recurrent Nets* [online]. 2015 [cit. 2017-04-29]. Dostupné z `https://arxiv.org/pdf/1503.01007.pdf`.

[39] Goodfellow, Ian J., Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron a Yoshua Bengio. *Generative Adversarial Networks* [online]. 2014 [cit. 2017-05-01]. Dostupné z `https://arxiv.org/pdf/1406.2661.pdf`.

Příloha **B**

Obsah přiloženého CD

- `MIDI` ..... složka s MIDI soubory
  - `mix`
    - `all`
    - `generated`
  - `one_author_multiple_songs`
    - `beethoven`
    - `haydn`
    - `generated`
    - `mozart`
  - `one_author_single_song`
    - `beethoven`
    - `haydn`
    - `mozart`
- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `implementation` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
  - `uzivatelska_prirucka.txt` ..... uživatelská příručka
- `text`
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `dotaznik.pdf` ..... dotazník na hodnocení skladeb