TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Generování hudby pomocí neuronových sítí

Bakalářská práce

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

Autor práce: **David Černý**
Vedoucí práce: Ing. Karel Paleček Ph.D.

Liberec 2020

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání bakalářské práce

Generování hudby pomocí neuronových sítí

Jméno a příjmení: **David Černý**
Osobní číslo: M16000017
Studijní program: B2646 Informační technologie
Studijní obor: Informační technologie
Zadávající katedra: Ústav informačních technologií a elektroniky
Akademický rok: **2019/2020**

Zásady pro vypracování:

1. Seznamte se s problematikou automatického generování hudby pomocí umělých neuronových sítí.
2. Sestavte dataset hudebních děl a vhodně upravte pro snadné zpracování neuronovými sítěmi.
3. Navrhněte vhodnou architekturu neuronové sítě pro automatické generování hudebního obsahu.
4. Kvalitativně porovnejte navržený systém s volně dostupným softwarem.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30-40 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] Goodfellow, I., Bengio, Y., Courville, A. Deep learning. MIT Press, 2016
[2] Bishop, C. Pattern Recognition and Machine Learning. 2006. ISBN 13: 978-038731073
[3] Karpathy, A., Johnson, J., Li, F. Convolutional neural neworks for visual recognition. dostupné online: http://cs231n.stanford.edu/

Vedoucí práce:	Ing. Karel Paleček, Ph.D. Ústav informačních technologií a elektroniky
Datum zadání práce:	9. října 2019
Předpokládaný termín odevzdání:	18. května 2020

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

prof. Ing. Ondřej Novák, CSc.
vedoucí ústavu

V Liberci dne 18. října 2019

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

1. června 2020 David Černý

Generování hudby pomocí neuronových sítí

Abstrakt

Cílem práce je navrhnout a realizovat funkční architekturu neuronové sítě pro generování hudebního obsahu.

Klíčová slova: Neuronové sítě, generativní modely, generování hudby, hudba

Music generation using neural networks

Abstract

Purpose of thesis, is to design and implement neural network architecture for generation of musical content.

Keywords: Neural networks, generative models, music generation, music

Poděkování

Děkuji Ing. Karlovi Parlečkovi Ph.D. za pomoc při vedení bakalářské práce.

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

GAN	Generative adversarial networks
KL	Kullback-Leibler
MOS	Mean opinion score
MLP	Multi layer perceptron
LSTM	Long short-term memory
RNN	Recurrent neural network
RVAE	Recurrent Variational autoencoder
TTS	Text to speech
VAE	Variational autoencoder

1 Úvod

V posledních letech došlo k výraznému posunu technologií umělé inteligence a strojového učení. Tento posun umožnil realizaci nových a zajímavých aplikací, které nám jsou k užitku téměř denně. Jednou z nových aplikací je generování hudebního obsahu. Myšlenka generování hudby pomocí umělých neuronových sítí není nová a byla často zpracovávána vědeckými týmy v minulosti, ovšem nyní, díky posunu výše zmíněných technologií, přicházejí slibné výsledky. Vznikají projekty pro zastoupení zaniklých nebo rozpuštěných populárních skupin, jako třeba Beatles, pomocné produkty pro skladatele, nebo celé virtuální orchestry, vše uvnitř jednoho programu.

Ve své práci se věnuji generování hudby pomocí umělých neuronových sítí. Hudbu jsem si vybral právě proto, že historicky jde o disciplínu, kterou plně ovládlo jen omezené množství lidských skladatelů. Přesto jsou zde pokusy a často úspěšné, ve kterých se umělá inteligence komplexní pravidla hudební teorie naučila a dokázala vytvořit něco zcela nového.

Využívat budu určitou nadstavbu nad rekurentními neuronovými sítěmi, neboť právě tyto architektury ukazují slibné výsledky nad zpracováním sekvenčních dat. Prozkoumám aktuální technologie pro strojové učení hudby, vyberu si a vhodně upravím dataset hudebních děl a pokusím se aplikovat jednu z používaných technologií na svoji množinu dat tak, aby byl výstup modelu co nejkvalitnější.

Po domluvě s vedoucím práce, jsme zadání zúžili na polyfonní výstup. Tedy ve výstupní skladbě je dovoleno použití více not ve stejný časový okamžik. Tato definice umožňuje vytvoření skladby s doprovodem. Práce nebude brát v potaz možnost více nástrojů v jedné písni. Veškerý výsledek je zpracováván pouze pro piano. Síť bude pracovat autonomně, tedy bez lidského vstupu. Fyzickým výstupem je MIDI soubor obsahující vytvořený výstup. Hudební styl specifikován není, záleží tedy na stylu zvolených, trénovacích dat.

2 Generování hudby pomocí umělých neuronových sítí

Generování hudby představuje jen velmi omezenou podmnožinu celkové aplikace neuronových sítí, ovšem prvopočátky řešení toho problému lze najít již v roce 1957.

> It was a 17 seconds long melody named "The Silver Scale" by its author Newman Guttman and was generated by a software for sound synthesis named Music I, developed byMathews at Bell Laboratories. [25]

Music I a další jeho nástupci dosahovali lokálních úspěchů převážně díky širokým stochastickým algoritmům využívající Markovových řetězců, nebo převodem not/akordů do jazykového modelu a aplikováním gramatických pravidel. Teprve značný posun ve vývoji umělých neuronových sítí dovolil tyto aplikace povýšit na více obecné zpracování.

Jako první pokusy o zpracování a generování hudby pomocí neuronových sítí byly práce od P. Todd ([30]) a J. Lewis ([19]) z roku 1988. Oba pokusy byly svými autory považovány za neúspěch, ovšem principy jejich práce jsou do dnešní doby stále validní. Práce J. Lewise v překladu "Stvoření zdokonalením" používá MLP síť a pomocí gradient descent iterativně aktualizuje hudební vstup. Stejná myšlenka stojí za slavnou sítí DeepDream. P. Todd oproti tomu viděl jako perspektivní rekurentní sítě typu RNN a adresoval problém zpracování hudby jako sekvenční úlohu. Rekurentní sítě jsou základem téměř jakékoli architektury generující hudební obsah dodnes. Jejich tehdejší neúspěch je dnes dáván za vinu limitům výpočetní techniky, která je pro trénování neuronových sítí zásadní.

Z těchto prvních pokusů následně čerpalo informace mnoho dalších akademiků. Nejznámější je společná práce od D. Eck a J. Schmidhuber: Finding temporal structure in music: blues improvisation with LSTM recurrent networks [9]. Zde autoři navrhují řešení zásadního problému hudebních dat. Nedostatek soudržnosti a struktury pro snadné zpracování neuronovou sítí. Jejich řešením je aplikace v té době nové architektury rekurentních sítí, LSTM. LSTM jsou do dnes jedny z nejpopulárnějších architektur pro zpracování sekvenčních dat. V té době se ovšem jednalo o jednu z prvních aplikací tohoto druhu sítě. Navíc se D. Eck a J. Schmidhuberovi podařilo vygenerovat opravdu přesvědčivé Jazzové výstupy.

Všechny tyto práce a několik dalších využívali jako vstup sítě určitý typ symbolické reprezentace hudby (2.2). Tím je myšleno vstup popisující hudbu jako sekvenci not, akordů, pomlk apod. Tedy reprezentace mnohem bližší našemu chápání hudby. Prvním výskytem jiného přístupu je práce od Matija Marolt, Alenka Kavčič a Marko

Privosnik [20], kteří použili spektogram hudebních děl pro trénování své sítě. Bohužel v tomto období nebyl výzkum v odvětví neuronových sítí prioritou. To se změnilo až v roce 2009, kdy se začalo objevovat velké množství prací zkoumající hudbu v souvislosti se strojovým učením.

Mimo generování hudby lze nalézt i aplikaci neuronových klasifikátorů na tagování hudby nebo rozpoznání akordů a jejich progresí. Použití spektogramů a symbolické reprezentace byly hlavními standardy většiny architektur. Právě v této době lze objevit první pokus využití konvolucí pro získání relevantních příznaků z obrazové reprezentace vstupních dat. V tomto případě jde o reprezentaci pomocí spektogramu [17]. Práce dlouho sloužila jako základ pro pokročilé aplikace využívající stejnou datovou strukturu. Je nutno říci, že i přes to, že byly spektorgramové reprezentace velmi populární, dnes je jejich výskyt mnohem méně častý. Konvoluční sítě jsou ovšem běžně využívány hlavně pro piano-roll (2.2.1), což je jedna ze symbolických reprezentací hudby.

Významným milníkem je práce z roku 2014 od Sander Dieleman a kolegů, která se snaží využít čistou audio waveform reprezentaci hudby pro klasifikaci a tagování písní ([7]). Práce nepředčila všudypřítomné spektogramové modely, ovšem pozdější výzkum ukázal velký potenciál waveform vstupu při využití dostatečně velkého datasatu. I z toho důvodu nebyla Sander Dielemanova práce tak úspěšná, jak si zasloužila. Waveform reprezentace je např. využívána ve velmi úspěšné síti WaveNet.

Obecně lze rozdělit tuto problematiku na dva hlavní proudy. Algoritmická kompozice využívá generativní modely pro vytvoření nového hudebního obsahu tak, aby cílený výsledek byl co nejsofistikovanější vzhledem k našemu pocitu z hudby. Dnešní výzkum se pojí s testováním a návrhem nových, efektivních architektur, jako je např. GAN (3.2) nebo VAE (3.3.2). Jednou z hlavních hnacích sil celého výzkumu je vytvoření generátoru, jehož výstup by nebyl omezen hudebním standardem MIDI, který je využíván ve valné většině projektů, ale který by byl schopen svůj výstup komponovat do přímého audio waveform. Oproti tomu ostatní disciplíny v oblasti hudby a neuronových sítí se spíše zajímají o získání relevantních informací hudebních dat jakékoli reprezentace. Obecně nejpopulárnější ovšem bývá waveform forma zpracování dat pro získání informací. Waveform lze totiž použít v mnoha odvětvích a existují ambice na obecně funkční klasifikátor zvuků, kde bude forma získání informací z dat klíčová.

2.1 Úspěšné generátory

2.1.1 WaveNet

WaveNet je hluboká, plně konvoluční síť navržena firmou DeepMind v roce 2016 [31]. Model je navržen pro generování čistého audio výstupu a jeho hlavním úspěchem je překonání state of the art na poli TTS (Text to speech), kterou v tu chvíli zastávala síť od Googlu. Text to speech je disciplína převodu psaného textu do mluveného obsahu. Právě TTS a generování vysoce kvalitních mluvených projevů je hlavním cílem WaveNet. Pozoruhodné ovšem je, že tento model může být i úspěšně natrénován na

hudebních datech a je schopen vysoce kvalitního hudebního výstupu.

Do úspěchu WaveNetu byla většina modelů pro TTS trénována k použití velké databáze krátkých mluvených úryvků a přesvědčivého projevu docílili jejich spojováním. Tyto modely trpí nepřirozenými tóny, kadencí a přechod mezi jednotlivými fragmenty je často velmi znát. Druhou možností modelování TTS je použití parametrických modelů, které se dokáží naučit pouze díky parametrizovaným, učeným funkcím syntetizovat plný výstup. Ovšem většina parametrických modelů trpěla náhodnými variacemi v tónině a spojovací TTS systémy převažovaly. Právě WaveNet dokázal jako parametrický model překonat MOS (Mean opinion score - standardní metrika pro kvalitu audio systémů) nejúspěšnější Google sítě pro TTS založené na spojování fragmentů.

Autoři WaveNetu spojují hlavní část svého úspěchu s esenciálním generováním jednoho časového kroku po druhém. V audio systémech, kde se obvykle pracuje s 16 000 vzorky za sekundu, je tento proces velmi výpočetně náročný a proto ne příliš populární. Mimo vstupní signál je model podmíněn mnoha dalšími vstupy, kterými dokáže přepínat mezi jednotlivými řečníky (včetně pohlaví) a dokonce ovládat výšku tónu, nebo emocionální zabarvení.

Pozoruhodné je, že po natrénování modelu nad daty klasické hudby hrané na klavír, se síť naučila generovat vysoce kvalitní a přesvědčivé hudební výstupy. WaveNet je v tomto ohledu opravdu multifunkční síť pracující s jakýmkoli audio vstupem, je-li pro něj natrénována. Samotné trénování sítě ovšem podle autorů vyžaduje alespoň 50 hodin audio nahrávek.

V současné době se autoři snaží vylepšit již úspěšnou funkci modelu, tzv. content-swaping. Content-swaping je funkcionalita pro výměnu hlasu na existující audio nahrávce za jiný, obsažený v trénovacích datech modelu. S touto funkcí by bylo možné převést jakýkoli mluvený projev do projevu jiného člověka.

2.1.2 Magenta

Magenta je otevřený projekt od Google Brain, zaměřující se na neuronové sítě řešící obecně kreativní úlohy. Hlavním zaměřením všech projektů a prací vytvořených v rámci Magenty je interaktivita uživatele s výsledným produktem. Mezi výstupy patří aplikace a zařízení řešící úlohy spojené s hudbou nebo obrazy tak, aby je mohli umělci interaktivně využívat při své práci. Magenta také vytváří open source knihovnu, založenou na populárním python frameworku pro neuronové sítě tensorflow, která poskytuje podporu vývojářům při řešení hudebních, nebo jinak kreativních úloh pomocí neuronových sítí.

Výstupem magenty tedy není jediná síť generující hudbu nebo zvuky. Mezi některá konkrétní řešení patří např. Magenta studio, desktopová aplikace generující melodii se specifickými vlastnostmi jako je délka, zabarvení tónů, nebo jejich variace. Tyto restrikce jsou zadány uživatelem v jednoduchém grafickém uživatelském rozhraní. Magenta studio pracuje s několika nástroji a je přímo kompatibilní se standardem MIDI, díky čemuž může být studio propojeno přímo s hudebními zařízeními nebo nástroji.

Dalším zajímavým produktem je NSynth (Neural Synthesizer). Jedná se o zařízení

produkující waveform výstup spojené kompozice, uživatelem vybraných zvuků. Pomocí dotykového displeje lze nastavit, které nástroje se bude NSynth snažit propojit dohromady. Výsledkem jsou nové tóny, které uživatel může skládat do melodií a dokonce připojit přímo na mixážní pult. NSynth je možno propojit i s konkrétním nástrojem a komunikovat s ním díky formátu MIDI. Neuronová síť, která tvoří jádro celého projektu, je naučena získávat důležité vlastnosti zvuku z jeho waveform reprezentace a spojovat je z několika vstupů do jednoho originálního výstupu.

Na podobném principu jako NSynth ovšem více abstraktně pracuje i MusicVAE. Projekt kombinující dvě různé melodie do jedné. Síť je opět naučena získávat reprezentativní příznaky melodií a vytvořit jejich společnou kompozici.

Většina konkrétních řešení je docílena pomocí state of the art technologií pro generování sekvenčního obsahu neuronovými sítěmi za pomoci zpětnovazebního učení (reinforcement learning). Myšlenka tohoto procesu je v poskytování odměn, nebo penalizací modelu, podle specifických metrik jeho výstupu. V případě hudebních aplikací jde nejčastěji o termíny hudební teorie. Například síť může dostávat penalizace za vygenerování noty mimo definovanou stupnici, příliš velkou autokorelaci výstupu nebo velmi časté opakování jedné noty. Právě opakování stejných not je obtížně řešitelný problém ve všech modelech generující hudbu. Odměny sítě může dostávat, pokud výsledná kompozice začíná tonální notou, nebo pokud jsou vygenerované noty součástí motivu písně. Pravidla a hodnoty odměn/penalizací musí být cíleně navržena pro maximální efektivitu sítě a jejího trénování. Právě tyto příklady zpětnovazebního učení byly použity v LSTM modelu Magenty z roku 2016.

2.1.3 MuseNet

MuseNet je OpenAI projekt publikovaný v roce 2019. Jde o model hudebního generátoru pracující s deseti různými nástroji najednou, vytvářející až čtyř minutové skladby o různých stylech. Hlavním rozdílem MuseNetu od ostatních architektur je jeho uživatelsky volitelná forma stylu, která bude aplikovaná na výstup. Navíc síť neumí pracovat sama o sobě a potřebuje alespoň jednu ze dvou možných interakcí uživatele. V prvním případě je MuseNetu předán krátký hudební vstup a označení stylu výstupu. Model se pokusí v předané skladbě pokračovat a přenést ji do vybrané stylizace tak, aby původní informace na vstupu definovala motiv skladby. Tímto způsobem je možno zahrát Bacha nebo Shuberta např. v pop nebo country žánru. V druhém případě uživatel aktivně interaguje se sítí a upravuje různé parametry ovlivňující výstup generování.

Množství možných hudebních stylů MuseNetu je mnoho a lze použít i stylizaci jednotlivých slavných autorů. Na příklad Bach, Mozart, ale dokonce i Beatles jsou použitelné jako podmíněné stylizační vstupy.

2.2 Datový prostor

Základním prvkem, který ovlivňuje jak výslednou architekturu, tak i celé zpracování vstupního datasetu je jeho reprezentace. Obecně musíme síti poskytnout dostatečné

množství hudebních děl, ze kterých se pokusí naučit strukturu a pravidla hudby. Síť se poté snaží pomocí těchto naučených informací vytvořit svoji vlastní skladbu. Prvotní otázka tedy zní, v jakém formátu trénovací data síti poskytneme. V zásadě jde o rozhodnutí mezi symbolickou, nebo audio reprezentací.

Audio reprezentací je myšlen velmi low-level přístup ke kódování vstupu. Vstupem tak může být waveform audio signálu nebo spektogram či chromagram z něj získaný. I přesto, že neuronové sítě obvykle fungují nad více abstraktním datovým prostorem, který síti umožňuje lépe porozumět souvislostem mezi prvky vstupu, tak tento přístup byl často používaný a měl velmi dobré výsledky v sítích, jako je třeba WaveNet (2.1.1).

Symbolická reprezentace je více podobná našemu, abstraktnímu chápání hudby. Na hudební dílo je nahlíženo jako na sekvenci symbolů (tokenů). Symboly může být myšlena nota, akord, pomlka apod. Toto kódování vstupu mnohem více přibližuje zadání myšlence generování textu, neboť nyní máme konstantní množinu vstupů a mění se pouze jejich výskyt. Na množinu symbolů lze tedy nahlížet jako na abecedu, kde každá nota na rozsahu instrumentu je jedním písmenem. Bohužel je zde zásadní rozdíl oproti jazykovým modelům a to právě skutečnost, že v hudebním díle se vyskytuje v jeden časový okamžik vícero symbolů najednou. Např. v akordu C-dur jsou přítomny noty C, E a G, což z definice není pro jazykové modely možné.

Vzhledem k velmi častému používání symbolické reprezentace hudby oproti audio reprezentaci ve většině publikovaných modelů jsem se rovněž rozhodl pro symbolickou reprezentaci, konkrétně ve formě piano roll.

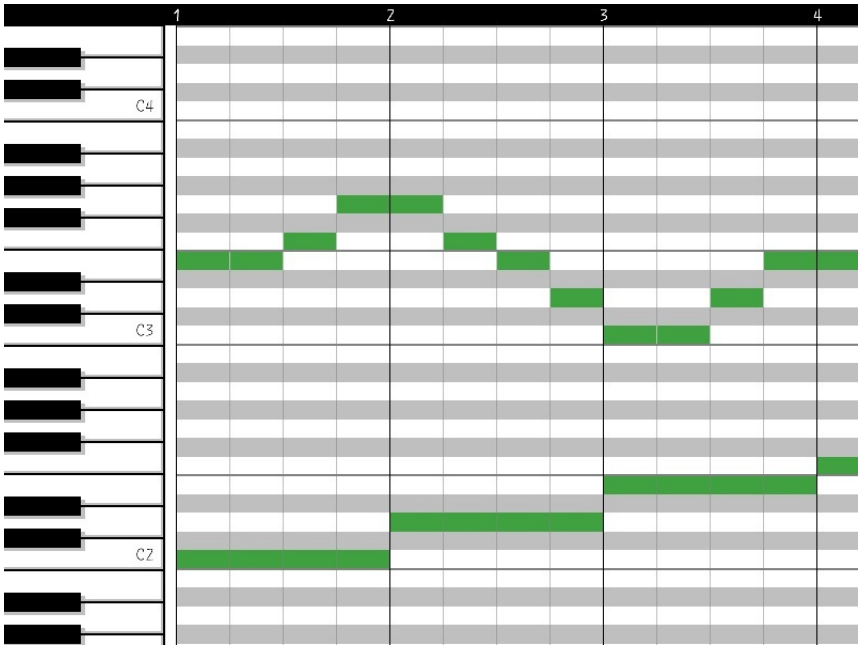
2.2.1 Piano roll

Piano roll je obvykle obdélníková matice velikosti $[mxn]$, kde m je tónová osa písně a n je časová osa písně. Původně se jedná o formát inspirovaný automatizovanými piány. Vstupem je děrovaná role papíru, kde každá díra reprezentuje spínač pro zahrání specifické noty.

Vyjdeme-li z obrázku 2.1, lze si piano roll představit jako matici jedniček a nul, kde 1 označuje hranou notu a 0 žádnou akci, tak jak je to vyjádřeno na obrázku pomocí barev. V některých případech je v piano roll uvažována i třetí dimenze, tedy síla tónu. Tuto dimenzi můžeme vyjádřit výměnou hodnoty 1 v případě přítomnosti tónu za škálu popisující sílu úderu.

V těchto případech je velmi užitečný MIDI formát. MIDI je technický standard používaný pro popis komunikačních protokolů mezi různými audio zařízeními a PC. Hudba uložená v mid formátu je čtena jako série událostí, kde událost může být zahrání noty, změna stupnice apod. Mimoto každá nota uložena v MIDI má přiřazeno číslo udávající sílu tónu. Tato hodnota je vyjádřena číslem ze škály 0-127. Stejná škála se používá i pro označení hraného tónu na instrumentu.

Velice důležité při vytváření piano rollu je zvolení optimálního časového kroku. Během vzorkování písně, pro získání jednotlivých sloupců piano rollu můžeme vzorkovat fixním krokem. Pokud bude krok příliš krátký, bude např. jedna celá nota zabírat mnoho místa ve výsledné matici, což může být problém při následném trénování. Pokud bude krok příliš velký, některé krátké tóny se v piano rollu vůbec



Obrázek 2.1: Piano roll příklad [15]

nemusí objevit. Existují 2 obecně implementovaná řešení.

Prvním řešením je vzorkovat s periodou délky nejkratší noty objevující se v písni. Tento proces vyžaduje prvotní analýzu písně pro získání tempa a následné propočítání délky nejkratší noty, která je z pravidla šestnáctková. Problém tohoto řešení leží ve změně tempa v průběhu písně, což je velmi častý jev, kdy musí být vzorkovací perioda přepočítána a není pevně stanovena nad celou skladbou. Dále některé pokročilejší hudební prvky jako třeba triplety nemohou být přesně přečteny. Často se tedy volí mnohem menší perioda. Mnohdy se též volí přístup postupného vzorkování písně podle struktury, např. podle taktů. V symbolické notaci je každá píseň sekvencí jednotlivých taktů.

Takt je tedy základní rytmická jednotka hudebního díla. Každý takt je dále dělen na několik dob a délka jedné doby je rovna délce určité noty. Pro označení taktů obecně slouží zlomek přirozených čísel, kde dělenec označuje počet dob v taktu a dělitel délku jednoho taktu, např. 4/4 takt značí, že jeden takt má 4 doby o délce čtvrťové noty. Tento popis taktu se opět může v průběhu písně libovolně měnit, čímž činí změnu dynamiky skladby, tedy ani vzorkování v rámci taktu není stejné napříč celou písní. Z toho důvodu je v MIDI formátu dále uváděn parametr ticks per beat (tiky za dobu), čímž se dále dělí doba taktu do menších jednotek a vzorkuje se podle pevně stanoveného počtu tiků na jednu dobu. Toto řešení umožňuje síti nahlížet na každý takt jako na individuální stavební prvek písně a díky tomu se lépe učí konceptům dynamiky a rytmu.

Druhým řešením je použití pevné časové konstanty např. 10ms pro vzorkování. Toto řešení není často používané, protože nedokáže rozpoznat, kdy začínají/končí strukturní prvky písně jako např. takty. Výhodou tohoto přístupu je možnost zachycení expresivity hraní, pokud data pocházejí z představení individuálních umělců,

neboť v průběhu hraní expresivní umělec sám mění délky not a akordů.

Ve výsledku ovšem piano roll trpí jednou závažnou, ovšem ne neřešitelnou nevýhodou. Po převedení písně do této matice není možno poznat rozdíl mezi jednou dlouze hrající notou a několika na sebe navazujícími zahrání té samé noty. V některých projektech je tento problém zanedbáván, v jiných je adresován pomocí přidání nového symbolu pro držení noty. Pokud je tedy nota stisknuta, objevuje se číslem 1, nebo silou úderu (hlasitostí) a v dalším vzorku je jí přiřazen symbol značící, nota je stále hrána.

2.2.2 Další symbolické reprezentace

Mimo piano roll je samozřejmě používáno velké množství jiných reprezentací, každá mající svoje výhody a nevýhody za určitých okolností. Nejjednodušší možností je one-hot kódování, které podobně jako piano roll vzorkuje píseň podle zvolené strategie, ale místo matice pracuje s vektorem jednoho vzorku jako ze základním prvkem a posílá vzorek za vzorkem do neuronové sítě.

Některé projekty také zvolily postup přímého kódování midi zpráv o událostech, jako vstupů pro síť. MIDI zprávy mají několik standardních typů, z nichž nejdůležitější jsou události začátku a ukončení zahrání jedné noty. Ke každé zprávě jsou přiloženy data o notě, hlasitosti zahrání, čase události apod. Čili seznam těchto zpráv popisuje všechny důležité informace o písni a dohromady nám dává kompletní hudební dílo. Hlavní nevýhodou této reprezentace je, že není stavěna pro informace několika událostí najednou. MIDI vše měří v čase a tři události v jeden časový okamžik jsou tedy zakódovány jako tři po sobě jdoucí události se stejným časovým parametrem.

Dalším populárním řešením je ABC notation - textová notace pro hudbu. ABC notation popisuje pravidla, kterými lze velmi jednoduše převést hudební dílo do psaného textu a rovněž použít textové architektury a sítě k jejímu zpracování. Hlavní nevýhodou je opět nemožnost vyjádření několika tónů ve stejný časový okamžik, na což nejsou jazykové modely stavěny, jak již bylo avizováno v kapitole 2.2.

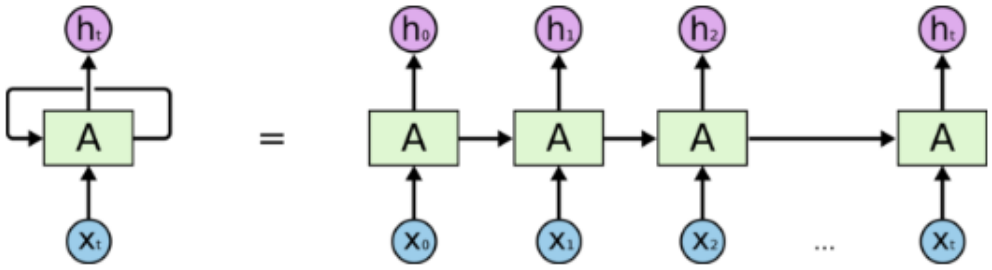
Další reprezentace jsou zřídkakdy používány a pokud ano, mají velmi specifický důvod, např. používají značkovací jazyk, nebo pouze popis hraných akordů.

3 Architektury neuronových sítí pro generování hudby

Jak již bylo avizováno v Úvodu, hudba je forma sekvenčních dat, tedy informací vyvíjejících se v čase. Obyčejná neuronová síť typu MLP není stavěna na takovou míru vnitřních závislostí a i když je v těchto případech aplikovatelná, je potřeba obrovské množství vnitřních parametrů, což činí síť velmi těžce trénovatelnou. Existují způsoby, jak trénování výrazně zrychlit a zároveň dosáhnout lepších výsledků.

3.1 Rekurentní neuronové sítě

Rekurentní neuronové sítě byly poprvé uveřejněny v roce 1986 a měly být přímým řešením problémů se sekvenčním učením. Hlavní myšlenkou bylo vytvoření vnitřního rekurentního spojení, které by sloužilo jako jakási paměť perceptronu. S pomocí vnitřní paměti aktualizující se vždy při dalším časovém kroku, t si síť může držet dočasné závislosti mezi prvky sekvence a predikovat následující prvek v čase $t + 1$. Pokud tento nový predikovaný prvek použijeme jako další vstup sítě, můžeme libovolně dlouho nechat síť generovat nový obsah jako třeba text nebo hudbu.



An unrolled recurrent neural network.

Obrázek 3.1: Rekurentní síť [24]

V čase t je vždy pomocí vstupu $X[t]$ a vnitřního stavu h_{t-1} vypočten nový stav $h[t]$, který je zároveň výstupem sítě. Z toho důvodu potřebuje RNN dvě sady vah (W^{xh} a W^{hh})), kterými rozlišuje mezi zpracováním vstupu a vnitřního stavu. V některých případech je možno se setkat s návrhem sítě typu RNN, která nepoužívá stav $h[t]$ jako svůj výstup a přidává do množiny parametrů další váhovou matici

W^{hy}, kterou transformuje vnitřní stav na skutečný výstup z_t. Celková funkce standardní RNN:

$$h_t = g(W^{xh} * x_t + W^{hh} * h_{t-1}) \tag{3.1}$$

Kde všechny váhové matice mohou být dle libosti rozšířeny o bias vektor a:

- h_t je nový vnitřní stav sítě a zároveň její výstup.
- funkce g je aktivační ne-lineární funkce, z pravidla Tanh nebo ReLU.
- W^{xh} je váhová matice, zpracovávající vstup sítě.
- x_t je vstup sítě v čase t.
- W^{hh} je váhová matice, zpracovávající vnitřní stav předešlého časového kroku.
- h_{t-1} je vnitřní stav sítě v čase $t-1$ nebo inicializační stav v případě kroku $t = 0$.

V takové implementaci je zřejmé, že síť průběžně aktualizuje svoji paměťovou buňku a drží si informace o kontextu dat, ovšem její obsah musí být uchováván ve formátu, který odpovídá výstupu predikce dalšího prvku v sekvenci. To je velmi limitující, neboť síť má velmi omezenou množinu hodnot popisující kontext sekvenčních dat. Z tohoto důvodu se za síť typu RNN z pravidla zapojuje síť typu MLP, která mapuje skrytý stav sítě na opravdový výstup, nebo je použita rozšířená architektura RNN:

$$h_t = g(W^{xh} * x_t + W^{hh} * h_{t-1}) \tag{3.2}$$

$$z_t = f(W^{hy} * h_t) \tag{3.3}$$

Kde rovnice 3.2 je identická s rovnicí 3.1. Oproti tomu rovnice 3.3 má parametry:

- z_t je výstup sítě.
- Funkce f může být další aktivační funkcí sítě, nebo výstupní aktivace pro klasifikaci např. Softmax.
- W^{hy} je váhová matice, mapující vnitřní stav na výstup.
- h_t je nový vnitřní stav sítě.

Opět všechny váhové matice mohou, ale nemusí obsahovat bias vektor.

S použitím RNN sítí získáváme kýžené výsledky u zpracování sekvenčních dat. Jejich použití se velmi rozmohlo ve všech typech projektů řešící podobné problémy. Ovšem i zde jsou limity.

3.1.1 Problém mizejícího gradientu

Problém mizejícího gradientu je častou příčinou výrazného zpomalení schopnosti sítě se učit. Problém je zanedbatelný pro mělké sítě, ergo pro sítě obsahující malé množství vrstev. Ovšem s rostoucí hloubkou roste i nutnost problém adresovat.

Základním učícím algoritmem pro neuronové sítě je zpětná propagace, kdy každá použitá matematická funkce je zpětně zderivována a použita podle řetízkového pravidla pro propagaci gradientu na váhové matice nebo jiné parametry sítě. S rostoucí hloubkou sítě je gradient zpětné propagace často stále menší a proto má i menší důraz na trénování. Tedy za jednu trénovací iteraci se váhy změní jen velmi málo.

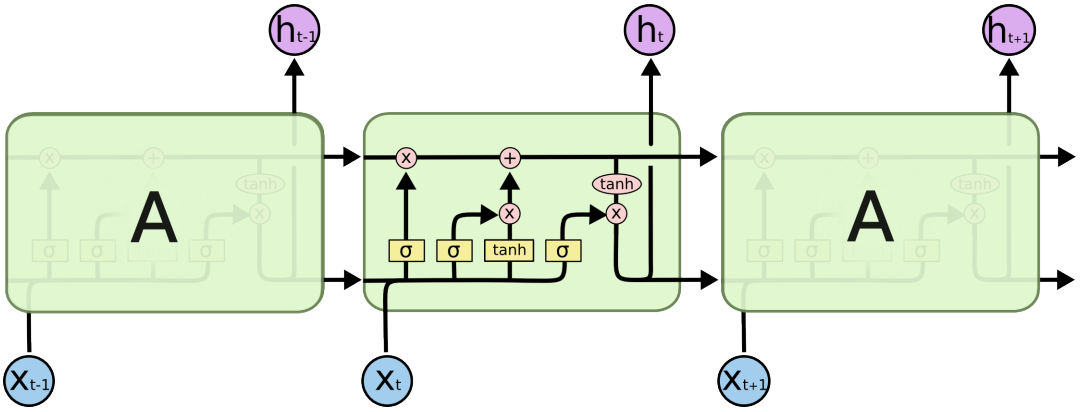
Hlavní příčinou mizejícího gradientu jsou často špatně zvolené aktivační funkce, např. Sigmoid, kde i velmi znatelný nárůst ve vstupních hodnotách pozmění výsledek funkce jen omezeně, což přímo implikuje nízký gradient. Pokud je takováto funkce použita za sebou v několika vrstvách, gradient se stále rychleji zmenšuje, neboť následné násobení jeho prvků v dalších krocích zpětné propagace problém jen zhoršuje.

Ani sítě typu RNN toho nejsou ušetřeny, neboť při generování obsahu je vždy použit vygenerovaný prvek sekvence jako další vstup. Zpětná propagace tedy putuje sítí přes celou délku sekvence. Pokud např. chceme generovat několik odstavců dlouhý text, při zpětné propagaci je chyba z poslední věty efektivně propagována pouze několik málo znaků, i když její příčina může ležet i o odstavec zpět.

Obecným řešením mizejícího gradientu je lepší volba aktivačních funkcí, jako např. funkce ReLU, která udržuje vysoké hodnoty vstupu neměnné. Nepříliš často implementovaným řešením je samozřejmě i zmenšení počtu vrstev (v případě RNN délky sekvence, kterou síť zpracovává). Standardním řešením bývá také batch normalizace, která udržuje vstupy v optimálním rozsahu, neboť aktivační funkce nechávají často mizet extrémní hodnoty. Nakonec velmi prospěšná může být i změna architektury na síť řešící tento problém již svým designem. V případě konvolučních sítí jde o populární ResNet architekturu a v kontextu rekurentních sítí jde snad o ještě slavnější sít LSTM.

Z našeho pohledu se síť LSTM chová naprosto stejně jako síť RNN. Jediným rozdílem je dvojnásobná velikost vnitřního stavu. LSTM používá dva vnitřní stavy a rozlišuje mezi nimi v rámci jejich aktualizace. Zatímco první, skrytý stav, popisuje kontext nejaktuálnějšího prvku sekvence a jejich nejbližších sousedů, druhý stav (stav buňky) se snaží udržet a postupně lehce upravovat kontext celé sekvence. Díky této a dalším vlastnostem síť typu LSTM přímo adresuje problém mizejícího gradientu a dnes je používána jako standard pro zpracování sekvenčních dat.

LSTM velmi výrazně urychlila posun ve zpracování sekvenčních dat pomocí umělých neuronových sítí a je stále používána i v nejnovějších projektech. Časem k LSTM přibylo velké množství dalších rozšíření, které ve většině případů vylepšují skóre na velmi specifických aplikacích. Ovšem existují i obecné architektury vycházející z LSTM, které zaznamenaly velký úspěch. Z nich pravděpodobně nejvýznamnější je Gated Reccurrent Unit (GRU).



Obrázek 3.2: LSTM řetězec [24]

3.2 Generative adversarial networks

Generative adversarial networks neboli GAN je generativní model neuronové sítě, poprvé publikován v práci Ian J. Goodfellow z roku 2014 [13]. Původní výzkum tohoto modelu byl zasvěcen rozpoznání a zabránění špatné klasifikace při použití adversarial examples. Časem se ovšem ukázalo, že GAN může být mnohem užitečnější v oboru generování dat.

Základní myšlenka leží v soutěži dvou proti sobě postaveným sítím. První síť zvaná diskriminátor $D(x)$ má binárně-klasifikační úlohu. Z příchozích dat x musí určit, která data pocházejí z pravděpodobnostního rozdělení x originálního datasetu a která ne. Tím implikujeme, že mimo originální dataset bude diskriminátor přijímat i jiná data, konkrétně data z rozdělení p_g. Tato data jsou podvodná. Diskriminátor tedy v určitém slova smyslu funguje jako jakýsi kontrolor, rozhodující mezi pravostí a podvržeností jemu příchozích dat.

Druhá síť zvaná generátor $G(z)$ má za úkol vytvořit data, která diskriminátora oklamou a budou klasifikována jako data původní. Vstupem generátoru je náhodná veličina p p_z, kde p_z je rozdělení náhodných čísel (konkrétní volba záleží na implementaci). Jeho hlavním úkolem tedy je simulovat co nejpřesněji původní datové rozdělení x. Pokud ho získáme, můžeme vyloučit síť $D(x)$ a pomocí $G(z)$ generovat uvěřitelná data.

Teoreticky pracujeme se dvěma pravděpodobnostmi:

- $D(x)$ je pravděpodobnost, že data pocházejí z rozdělení x a jsou tedy pravá.
- $1 - D(G(z))$ je pravděpodobnost, že data pocházejí z rozdělení $G(z)$ a jsou tedy podvržená.

Pokud budeme maximalizovat pravděpodobnost $D(x)$, nebude mít generátor dostatek prostoru pro svoje pokusy a učení skončí nezdarem. Pokud naopak budeme minimalizovat $\log(1 - D(G(z)))$, výstupy generátoru nebudou mít dostatečnou kvalitu pro naši aplikaci. Celý tento problém se dá vyjádřit pomocí min-max hry:

$$\min_G \max_D V(D,G) = E_{x \sim p_{data}(x)}[\log(D(x))] + E_{z \sim p_z(z)}[\log(1 - D(G(z))] \tag{3.4}$$

Jinými slovy mluvíme o hře dvou hráčů D a G. Výslednou hodnotou $V(D, G)$ myslíme maximální zisk hráče D při (pro D) nejhorším možném kroku hráče G. Konečně tento výraz naprosto naplňuje jak podstatu hry, tak i snahy obou hráčů. V praxi ovšem není možné GAN trénovat pouze za pomocí rovnice 3.4. Problém byl již naznačen v popisu věrohodností pro maximalizaci/minimalizaci. Generátor z počátku učení nebude mít dostatek kapacity pro kvalitní výsledky a diskriminátor naprosto jednoduše rozliší mezi originálem a podvrhem. Generátor tak nebude mít dostatek prostoru a učení skončí nezdarem.

Ian J. Goodfellow ve své originální práci tento problém adresuje pomocí změny počátečního učení G v maximalizaci $\log(D(G(z)))$ místo minimalizace $\log(1-D(G(z)))$, což je objektivně stejná funkce, ovšem poskytuje mnohem silnější gradient z počátku učení.

Trénování GAN pomocí stochastic-gradient descent není navíc tak přímočaré jako ve většině sítí, neboť můžeme aktualizovat váhy jen jedné sítě zároveň a musíme hlídat rovnováhu mezi nimi, aby soutěž byla vyrovnaná. Z toho důvodu se zavádí nový hyperparametr k, který určuje kolik iterací gradient-descentu má proběhnou na aktualizaci vah D oproti jedné u G. Hodnota tohoto hyperparametru se může lišit od aplikace k aplikaci.

V praxi se má originální verze GANu velmi nestabilní učení. Může za to převážně nutnost vyváženosti trénování mezi D a G. I přes použití alternativní loss funkce pro generátor během prvních fází učení dochází k velkým variacím v gradientu, tedy problém se nevyřeší zcela.

3.2.1 Wasserstein GAN

Wasserstein GAN byl poprvé uveden v práci Martina Arjovsky v roce 2017 [5]. Jde o vylepšení architektury GAN z hlediska zjednodušení učení pomocí vyhlazení průběhu gradientu v celé síti. Toho lze dosáhnout změnou loss funkce celé sítě a tím pádem i odlišným přemýšlením nad celým problémem.

Wasserstein vzdálenost, také známá jako Kantorovich–Rubinstein vzdálenost, je metrika určující vzdálenostní funkci dvou pravděpodobnostních rozdělení. Jejím použitím dosáhneme jemnějšího gradientu v celém průběhu učení bez ohledu na aktuální výsledky D a G. Tedy i pokud diskriminátor naprosto vždy rozezná falšované vstupy, generátor se stále učí a gradient ve zpětné propagaci nezmizí. Po zapracování Watterstein vzdálenosti do GAN získáme:

$$W(P_r, P_g) = \sup_{||f||_L \leq 1} E_{x \sim P_r}[f(x)] - E_{x \sim P_\theta}[f(x)] \tag{3.5}$$

Kde f je zatím neznámá funkce, která musí dodržovat omezení 1-Lipschitz funkce, tedy:

$$|f(x_1) - f(x_2)| \leq |x_1 - x_2| \tag{3.6}$$

Pro nás výhodné je, že 1-Lipschitz funkci můžeme modelovat neuronovou sítí, která základní omezení implementuje. Omezení lze pochopit jako udržení funkčních hodnot dvou neznámých v intervalu jejich vlastnímu rozdílu a funkce tedy nesmí výslednou hodnotu "příliš přepálit".

Nová síť bude vyměněna za diskriminátor. Ovšem strukturou může být naprosto identická. Hlavním rozdílem je loss funkce a výstup. Ten již nyní nebude vytvořen sigmoid aktivací, tedy výstupem sítě bude skalár. Hodnota skaláru může být interpretována jako skóre reálnosti vstupních dat, neboť vše co ji odlišuje od D, je sigmoid funkce. Nová síť již tedy nerozhoduje na základě pravděpodobnosti, ale hodnotí kvalitu vstupu. Z toho důvodu je pojmenována Critic.

	Diskriminátor/Kritik	**Generátor**
GAN	$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [\log D(x^{(i)}) + \log(1 - D(G(z^{(i)})))]$	$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D(G(z^{(i)})))$
WGAN	$\nabla_{w} \frac{1}{m} \sum_{i=1}^{m} [f(x^{(i)}) - f(G(z^{(i)}))]$	$\nabla_{\theta} \frac{1}{m} \sum_{i=1}^{m} f(G(z^{(i)}))$

Tabulka 3.1: WGAN loss v porovnání s GAN

Nakonec zajištění omezení 1-Lipschitz funkce je implementováno v trénování sítě pomocí konstanty c. Ta je dalším hyperparametrem sítě a určuje maximální hodnotu, o kterou se mohou váhy sítě zvětšit/zmenšit v jedné iteraci gradient descentu. Praktická implementace probíhá funkcí clip, která je přítomna ve většině frameworků pro implementaci neuronových sítí.

3.3 Autoenkodéry

Autoenkodér je speciální architektura neuronové sítě, která je velmi populární v řadě výzkumných projektů, ale také v menších projektech, nezávislých nadšenců do neuronových sítí. Základní myšlenka autoenkodéru sice není zaměřena na generování obsahu, ale architektura byla mnohokrát vylepšena a augmentovaná na jiné formy úloh. Dnes se dá říci, že autoenkodéry jsou spíše rodinou architektur pro neuronové sítě, než jeden nezávislý model.

Autoenkodér je neuronová síť se stejně velkou vstupní a výstupní vrstvou. Nutností je ovšem ještě skrytá vrstva, která je oproti vstupu a výstupu záměrně zmenšena. Z toho důvodu se jí říká "bottleneck layer"(vrstva s úzkým hrdlem). Cílem autoenkodéru je naučit se funkci identity a tedy přenést s co nejmenší chybou vstupní data na výstup. Zde úkol záměrně komplikuje ona vrstva s úzkým hrdlem. Autoenkodér vyhodnocuje výsledek přímím porovnáním vstupu a výstupu. Je tedy aplikováno učení bez učitele. Vyhodnocující chybová funkce se může lišit od implementace k implementaci, neboť je velmi závislá na typu a povaze dat. Obecně lze vždy použít např. mean square error (střední kvadratická chyba).

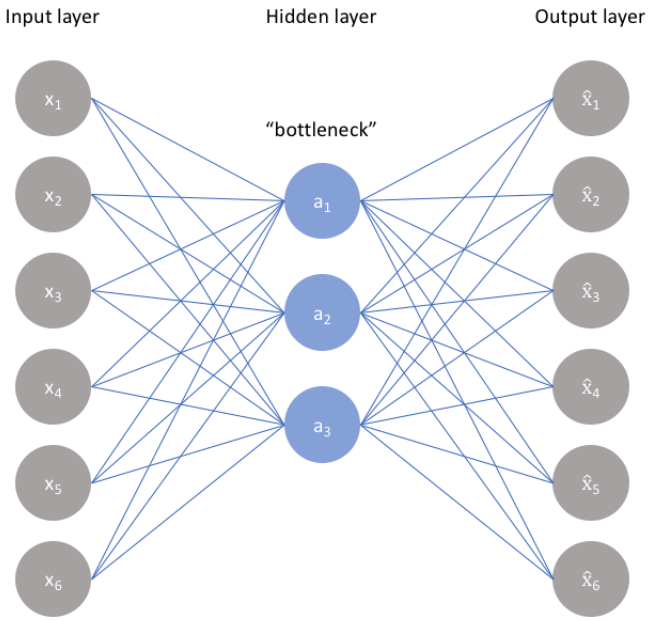
$$MSE = \frac{1}{n} \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 \tag{3.7}$$

Kde:

- Y jsou originální data.
- $\hat{Y}$ jsou výsledná data.

Ovšem v případě, kde máme data čistě v rozsahu $< 0, 1 >$, můžeme použít např. Cross-entropy funkci. Dále lze využít např. mean absolute error (střední absolutní chyba) apod.

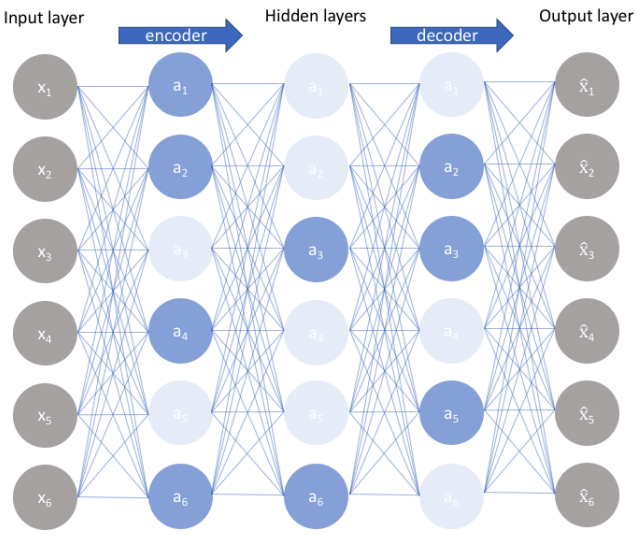
Autoenkodér musí beze změny přenést informace skrz celou síť. Díky bottleneck vrstvě je ovšem nutné je zakódovat do menšího datového prostoru. Tento zmenšený prostor je nazýván latent space. Na druhé straně sítě se informace dekódují opět zpět a porovná se výsledek vzhledem k loss funkci. Tím se architektura rozpadá na dvě sítě. Kodér se snaží vstup zakódovat do latentního prostoru a dekodér má za úkol je opět rekonstruovat. Autoenkodér se tedy učí automatizovaně nejlepší možnou kompresi dat, neboť komprese je navržena na specifický dataset. Velikost skryté vrstvy je hyperparametr, který se upravuje během učení. Po dokončení učení se dekodér ze sítě odpojí a kodér se použije pro extrakci příznaků z datasetu, které se poté mohou použít jako vstup do další neuronové sítě. Autoenkodér je tedy jakousi alternativou k metodám extrakce, ovšem ve většině případů jde o kódování, které se naučí velmi podobné metodě PCA.



Obrázek 3.3: Autoenkodér [16]

3.3.1 Řídký autoenkodér

Řídký autoenkodér (Sparse autoencoder) je architektura vycházející z idei klasického autoenkodéru, ovšem s jiným přístupem k problému. Řešení je navrženo pomocí omezení řídkosti aktivace skrytých neuronů. Tato funkce je implementována přidaným výrazem v chybové funkci a lze díky ní odhalit zajímavé struktury vstupních dat i při velkých rozměrech skrytých vrstev.



Obrázek 3.4: Řídký autoenkodér [16]

Jak je ilustrováno na obrázku 3.4, během jednoho průchodu architekturou, jsou aktivovány pouze některé neurony skrz všechny skryté vrstvy s tím, že je vyžadováno, aby neurony byly ve většině průchodu neaktivní.

Aktivita neuronu je posuzována s ohledem na aktivační funkci. Pokud používáme např. Sigmoid aktivaci je neuron neaktivní, když je hodnota jeho aktivace blízko nuly. Naopak aktivní je, když je blízko jedné. V případě tanh aktivace jde o hodnoty -1 a 1 a podobně lze získat hranice aktivace u ostatních aktivačních funkcí.

Po implementaci omezení aktivací do neuronové sítě v podstatě nutíme každý jednotlivý neuron, aby kontroloval unikátní aspekt vstupu a aktivoval se pouze v případě jeho výskytu. Aspekty mohou být hrany na určitých pozicích v obrázku, jejich orientace, nebo např. velké jasové složky. Tímto způsobem můžeme získat velmi užitečné informace a vlastnosti vstupu, které pomůžou jiné neuronové síti v efektivním učení. Při použití na hudební vstup můžeme mít neurony kontrolující stupnici nebo tempo písně. Následná generační síť podle těchto vstupů může vytvořit píseň novou.

Nejjednodušším omezením aktivací je např. L1 regularizace:

$$J_{sparse}(W,b) = J(W,b) + \beta \sum_{j=1}^{s2} |a_j^{(h)}| \tag{3.8}$$

kde:

- $J_{sparse}(W,b)$ je celkový loss řídkého autoenkodéru.
- $J(W,b)$ je loss rekonstrukce originálního obsahu.
- β je hyperparametr určující míru regularizace.
- $s2$ je počet neuronů ve skryté vrstvě.

- j je index určující jeden specifický neuron sítě.
- a^h je vektor aktivací vrstvy h.

Síť je tedy nucena minimalizovat vedle funkce rekonstrukční chyby ještě absolutní hodnoty veškerých aktivací přes všechny skryté vrstvy. Toto je opravdu jen velmi jednoduché a ne příliš funkční omezení. Řešení není použitelné pro všechny aktivační funkce a ne vždy zaručí aktivaci pouze omezené množiny neuronů. Z těchto důvodů se mnohem častěji používá KL divergence.

KL divergence je obecně používaná metrika pro měření rozdílnosti mezi pravděpodobnostními rozděleními. Pokud přemýšlíme nad jedním neuronem skryté vrstvy z hlediska jeho aktivace, máme pouze dva možné stavy. Aktivován nebo neaktivován. To samé lze vyjádřit pomocí náhodné veličiny, pocházející z Bernoulliho rozdělení s prozatím neznámou pravděpodobností. Pokusíme se tedy spočítat odhad průměrné pravděpodobnosti aktivace neuronu v síti $\hat{\rho}$.

$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^{m} [a_j^{(h)}(x^{(i)})] \tag{3.9}$$

Kde $a_j^{(h)}(x^{(i)})$ je výstup aktivace neuronu j ve vrstvě h za vstupu $x^{(i)}$. Nyní představíme nový hyperparametr ρ, což je naše cílová pravděpodobnost aktivace neuronu. Tato záleží čistě na designu sítě. Pokud nyní máme náš aktuální odhad průměrné aktivace a cílové Bernulliho rozdělení s parametrem ρ (pro informační účely řekněme, že jsme zvolili ρ jako 0.2), můžeme spočítat rozdíl mezi nimi pomocí KL divergence, vyvážit ji hyperparametrem β a přičíst ji k celkovému lossu. Celkový loss tedy bude:

$$K_{sparse}(W, b) = J(W, b) + \beta \sum_{j=1}^{s2} KL(\rho || \hat{\rho}_j) \tag{3.10}$$

kde $KL(\rho||\hat{\rho}_j)$ je:

$$KL(\rho||\hat{\rho}_j) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_j} \tag{3.11}$$

Zde $s2$ určuje počet neuronů skryté vrstvy. KL divergence je rovna nule v případě rovnosti obou rozdělení, tedy když $\rho = \hat{\rho}_j$. V tu chvíli je řídká aktivace neuronů zajištěna podle zadaného parametru ρ. Je důležité říci, že toto řešení řídkého autoenkodéru je ovšem pouze použitelné v případě aktivace pomocí funkce Sigmoid.

3.3.2 Variační autoenkodér

Variační autoenkodér se od ostatních autoenkodérů velmi odlišuje, neboť jeho úlohou není získání reprezentativních příznaků ze vstupních dat. Je určen pro generativní účely.

Princip

Princip variačního autoenkodéru leží ve využití latentního prostoru pro jiné účely než datová komprese nebo extrakce příznaků. Pokud by se nám podařilo zaručit, že latentní prostor bude mít určité vlastnosti, vhodné pro jeho modelování pravděpodobnostním rozdělením, pak by stačilo po ukončení trénování z rozdělení latentního prostoru navzorkovat náhodnou veličinu (označujme z) a její dekódování by vytvořilo nový obsah. Hlavní vlastností, kterou by latentní prostor měl mít, je dostatečná regularita. Autoenkodér má během učení kompletní kontrolu nad vnitřní organizací latentního prostoru a jediný jeho úkol je zaručit co nejnižší rekonstrukční chybu za každou cenu. To přirozeně vede k jisté úrovni přeučení nad trénovacími daty. Síť okamžitě využije svoji svobodu v organizaci datového prostoru k maximální přesnosti. V případě variačního autoenkodéru by ovšem přeučení vedlo k silné inspiraci nového obsahu trénovacím datasetem.

Dostatečnou regularitu latentního prostoru lze zajistit během trénování přidáním dalšího členu do loss funkce, který zabrání přeučení a zajistí vlastnosti vhodné pro generační účely. Jedinou další změnou variačního autoenkodéru od obyčejného je jeho kódování vstupu. Namísto zakódování vstupního bodu do bodu latentního prostoru je kódováno do pravděpodobnostního rozdělení nad latentním prostorem. Vstup bude zakódován do parametrů určitého pravděpodobnostního rozdělení. Důvodem k tomu je mnohem jednodušší a intuitivnější vyjádření potřebné regularizace. Ačkoliv teoreticky můžeme zvolit jakékoli rozdělení, v praxi se používá z pravidla gaussovo (normální). Vstupem dekodéru je střední hodnota a kovarianční matice vstupních dat.

Regularizační vlastnosti latentního prostoru

Od prostoru z potřebujeme zajistit, aby se prostor datový a latentní vzájemně reflektovaly, ale také abychom pro náhodně zvolené rozdělení latentního prostoru získali po dekódování obsah dávající smysl. Pokud tyto vlastnosti nezajistíme, kodér by vracel rozdělení s velmi nízkými rozptyly, čímž by maximálně přesně mapoval rozdělení na vstup a docházelo by k přeučení. Dalším problémem je skutečnost, že kodér vrací na každý vstup rozdělení se středními hodnotami velmi daleko od sebe. Mapuje se tak každý vstup opět na libovolné místo latentního prostoru a nezajistili bychom blízké vzdálenosti podobných vstupů.

Regularizace musí být zajištěna jak na střední hodnotu, tak na kovarianční matici zvoleného rozdělení. Naštěstí normální rozdělení lze těmito parametry přímo definovat a není třeba brát v potaz jiné veličiny. Navíc můžeme lehce definovat náš cíl regularizace. Tím je nutnost udržet zakódované vstupy ve standardním normálním rozdělení $N(0, 1)$, tedy rozdělení se střední hodnotou 0 a standardní odchylkou 1.

Model

Kodér lze definovat jako $p(z|x)$, tedy rozdělení latentního prostoru vzhledem k danému rozdělení dat. Dekodér definuje opačně jako $p(x|z)$ a latentní prostor, který

bude využit ke generování pojmenujme $p(z)$. U $p(z)$ můžeme předpokládat standardní normální rozdělení $N(0,1)$. Vztahy mezi těmito rozděleními implikují Bayesův teorém:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} \tag{3.12}$$

Jelikož rozdělení dat $p(x)$ je nám neznámé a nedá se přímo pozorovat, dostáváme se do problému inference. Bayesovská inference vzniká v případě, kdy lze určitý problém formulovat Bayesovým teorémem. Toto se v praxi stává často a i přes to, že nám k výpočtu může obecně chybět popis jakéhokoli rozdělení v teorému. Nejčastěji se jedná právě o normalizační faktor $p(x)$. Tento problém lze adresovat několika aproximačními metodami. Konkrétně použijeme metodu Variační inference.

K předpokladu $z \sim N(0,1)$ přidáme další. Dekodér bude mít také normální rozdělení tak, že:

$$p(x|z) \equiv N(f(z), cI)$$

Kde f je deterministická funkce, kterou můžeme modelovat neuronovou sítí a c je konstanta násobící matici identity. V tuto chvíli použijeme pro řešení metodu Variační inference.

Cílem variační inference je aproximovat cílové rozdělení pomocí postupné optimalizace jedné rodiny pravděpodobnostních rozdělení. V tomto případě je zmíněná rodina opět gaussovo rozdělení. Definujeme $q_x(z)$ jako aproximaci $p(z|x)$:

$$q_x(z) \equiv N(g(x), h(x)$$

Kde g a h jsou parametrizované funkce pocházející z množin funkcí G a H, jejichž postupnou optimalizací můžeme s $q_x(z)$ manipulovat pro zajištění nejnižší hodnoty chybové funkce Variační inference. Jako chybovou funkci použijeme KL divergenci **??**. Naše optimalizační parametry jsou funkce g a h, parametry po optimalizaci nazveme g^* a h^*, rozdělením pro aproximaci je q_x. Po dosazení do aplikované Variační inference na Bayesovskou inferenci získáme:

$$(g^*, h^*) = \underset{(g,h)\in G\times H}{argmax} \left(E_{z\sim q_x}(\log p(x|z)) - KL(q_x(z), p(z)\right) \tag{3.13}$$

Po dosazení hustoty normálního rozdělení do logaritmu prvního výrazu získáme:

$$(g^*, h^*) = \underset{(g,h)\in G\times H}{argmax} \left(E_{z\sim q_x}(-\frac{||x - f(z)||^2}{2c}) - KL(q_x(z), p(z)\right) \tag{3.14}$$

Nyní máme téměř kompletní loss funkci. První výraz zajišťuje minimální vzdálenost vstupního bodu od dekódovaného, zatímco druhý výraz slouží pro regularizaci latentního prostoru. Jediné co nezapadá do celé definice je funkce f. Řekli jsme, že f bude aproximována pomocí neuronové sítě. Tato neuronová síť definuje dekodér. Stále nám ovšem chybí optimalizační výraz pro f. Teoreticky pro jakoukoliv funkci f modelovanou neuronovou sítí získáme optimální kodér a celá architektura fungovat bude. Ovšem pro maximalizaci rekonstrukčního výrazu chceme zahrnout f do optimalizačních parametrů loss funkce. Zvolená f bude ta, která maximalizuje

rekonstrukční člen výsledku Variační inference. Do regularizačního členu zahrnutá být nemusí, protože f definuje dekodér, který na latentní prostor nemá žádný vliv:

$$f^* = \underset{f \in F}{argmax} E_{z \sim q_x^*}(-\frac{||x - f(z)||^2}{2c}) \tag{3.15}$$

Kde F je množina funkcí pro výběr f. Nakonec spojíme všechny výrazy dohromady:

$$(f^*, g^*, h^*) = \underset{(f,g,h) \in f \times G \times H}{argmax} (E_{z \sim q_x}(-\frac{||x - f(z)||^2}{2c}) - KL(q_x(z), p(z)) \tag{3.16}$$

Funkce f, g, h nakonec modelujeme pomocí neuronových sítí. Kodér má sítě dvě. Jednu pro střední hodnotu a druhou pro standardní odchylku. Obě sítě ovšem v praxi nebývají naprosto nezávislé, proto je část sítě sdílena. Po získání parametrů normálního rozdělení je potřeba získat vzorky náhodné veličiny z. Zde je ovšem problém při zpětné propagaci, neboť vzorkování z normálního rozdělení není diferencovatelná operace a tudíž je zapotřebí využít re-parametrizační trik:

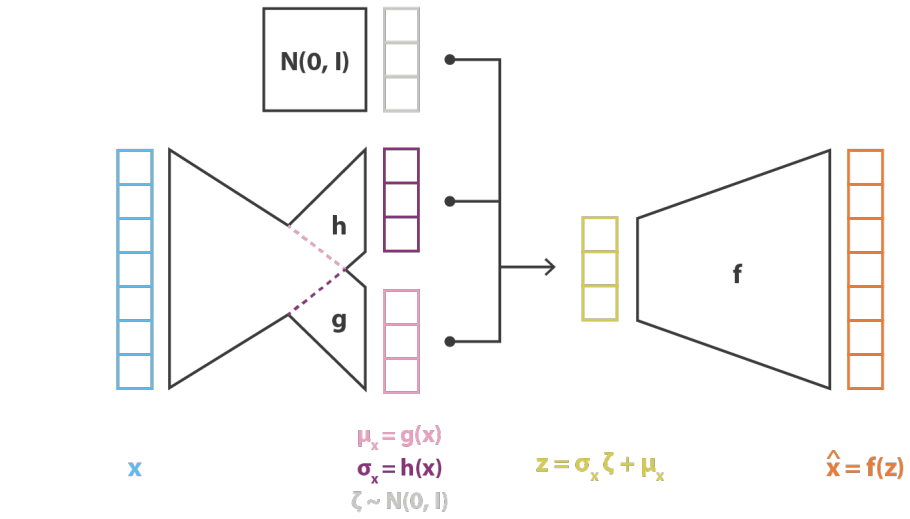
$$z = h(x)\varsigma + g(x) \tag{3.17}$$

Kde:

$$\varsigma \sim N(0, 1)$$

Tímto výpočtem lze zachovat gradient oproti sítím g a h. Gradient na rozdělení ς není pro funkcionalitu důležitý. Navzorkovaná proměnná z poté slouží jako vstup dekodéru, který funkcí f rekonstruuje původní obsah. Celá je ilustrována na obrázku 3.5.

Variační autoenkodér lze použít pro téměř jakoukoliv generační úlohu, neboť rodiny funkcí F, G, H reprezentují zvolené funkce neuronových sítí. Lze z nich zvolit standardní MLP, konvoluční síť nebo pro nás nejdůležitější síť rekurentní.



Obrázek 3.5: Variační autoenkodér [29]

4 Sestavení a upravení datasetu pro neuronovou síť

Pro svůj trénovací dataset jsem zvolil MIDI skladby na piano ze stránky vgmusic.com [3]. Autoři stránek uvádějí dostupnost jejich nashromážděného obsahu pro volné použití. Veškerá data na vgmusic pochází z prostředí videoher, valná většina z nich ze starších arkádových her. Dnešní hudba ve hrách je propracovaná a mnohokrát i komplexní, ovšem u starších titulů nebyl hudební doprovod prioritou. Sloužil hlavně pro vytvoření atmosféry. Z toho důvodu je většina těchto písní svou stavbou jednoduchá, ale přesto dobře strukturovaná. Neobsahuje vysoce komplexní hudební pravidla jako klasická hudba. Čím méně budou vstupní data komplexní, tím jednodušší bude pro síť trénovací fáze.

Prvním krokem byla analýza skladeb. Mezi důležité informace patřila délka písní, rozdělení hraných not, délky taktů, použité stupnice a instrumenty. Pro práci s midi formátem jsem použil python knihovny pretty-midi [27] a music21 [6]. Pomocí pretty-midi lze procházet každý .mid soubor přes dvě hierarchické úrovně. První jsou jednotlivé zvukové stopy. Každá stopa má definován svůj instrument, seznam tónů, změn ve stupnicích a délek taktů. Některé skladby definují pouze jednu stopu s piano instrumentem. Jiné, pravděpodobně navrženy pro výuku hry na piano, obsahují 2 stopy, každou pro jednu ruku při hraní na nástroj. Některé oproti tomu mají několik stop skrz které jsou tóny rozděleny. Pokud instrument stopy je mimo MIDI označení piana (1-8), je stopa ignorována.

4.1 Analýza

Dataset obsahuje 718 skladeb. Alespoň 23 z nich je kratší než 20 sekund a 60 písní je kratší než 40 sekund. Většinu z těchto skladeb jsem přesunul po pre-processingu do validačního datasetu. Zbytek písní je svou délkou optimální a délka více než poloviny přesahuje 100 sekund.

Analýza rozdělení hraných not získá všechny zahrané tóny v každé skladbě a vrátí jejich číselné označení podle MIDI kódu tónu. Každému je přiřazen celkový součet všech výskytů a jejich procentuální příspěvek v datasetu. Kompletní počet tónů ve všech skladbách se pohybuje kolem hodnoty 635 000. Finální přehled výsledku analýzy podle jednotlivých oktáv je v tabulce 4.1.

Z tabulky je patrná vysoká koncentrace většiny vyskytujících se tónů v intervalu 36 - 83. Zde se nachází celkem 91.8863% hraných not. Tyto informace jsou využity

Oktáva (MIDI)	Oktáva (hudební notace)	Výskyt (suma)	Výskyt (%)
0-11	C-1	1	0.0002
12-23	C0	180	0.0285
24-35	C1	28 643	3.8227
36-47	C2	113 183	17.8433
48-59	C3	154 967	24.4305
60-71	C4	189,057	29.8048
72-83	C5	125 644	19.8077
84-95	C6	21 349	3.3656
96-107	C7	1 276	0.2011
108-119	C8	17	0.0026
120-121	C9	0	0.0000

Tabulka 4.1: Rozdělení výskytu tónu podle oktáv

ve fázi preprocessingu pro transformaci tónů. Tóny spadající do rozsahu piana jsou podle midi značení v intervalu 21-108 (88 tónů). Zajímavým zjištěním je, že ne všechny tóny jsou součástí tohoto intervalu.

V důsledku výběru piano-roll (2.2.1), jako své datové reprezentace, bylo potřeba zvolit vzorkovací strategii a velikost jednoho rollu sekvence písně. Standardně se používají piano-rolly o velikosti reprezentující jeden takt skladby. Použití této délky pro časový rámec jednoho piano-rollu sekvence umožní síti lépe zachytit dynamiku a strukturu celé písně. Navíc, jelikož každá skladba končí svým posledním taktem, máme jistotu zachycení celé délky písně do sekvence rollů (matic) o stejné velikosti. Tím se eliminuje problém nezahrnutí některých not do piano rollu, nebo naopak nutnost doplnění posledního rollu nulami pro zachování velikosti. Tento přístup má ovšem několik problémů. Prvně, pokud chceme trénovat síť pomocí mini-batch dávkování je nutné, aby všechny piano-rolly měly stejnou velikost. To nás limituje ve výběru písní na dataset s konstantní délkou taktu, např. 4/4. Druhý problém rovněž omezuje dataset na výběr písní, ve kterých se délka taktu nemění a zůstává konstantní v průběhu celé skladby. Z těchto důvodů jsem implementoval analýzu délek taktů.

Analýza délek taktů v datasetu vysoce závisí na kvalitě jednotlivých midi souborů. Kvalitou je myšlena korektnost a úplnost důležitých metadat v souborech. Během průchodu jednotlivých souborů hledám první inicializaci a následné změny v proměnné time_signature, jejíž hodnota je rovna zlomkovému zápisu délky taktu. Bohužel celkem 117 souborů tuto proměnou naprosto ignoruje a ponechává ji prázdnou přes všechny stopy. Jiné soubory jich naopak definují několik, nejčastěji z důvodu inicializace stejné délky taktu pro každou stopu v souboru. Někdy jde ovšem o úmyslné změny v rámci jedné stopy a délka taktu je v průběhu písně proměnlivá. Výsledky jsou měřeny v jednotlivých stopách, nikoli v souborech. Soubory bez nastavené hodnoty time_signature jsou ignorovány.

- 4/4 takt: 732 stop
- 3/4 takt: 130 stop
- 6/8 takt: 61 stop
- 2/4 takt: 60 stop

Ostatní nalezené takty se vyskytují pouze velmi zřídka. Z výsledku analýzy vyplývá, že při mapování midi souboru do piano-roll sekvence nebude možno zajistit strukturu jednoho rollu na jeden takt, ovšem vzhledem k okolnostem jsem se rozhodl tento postup nadále dodržet. Hlavním důvodem je stále velmi velký poměr 4/4 taktu oproti všem ostatním. Navíc 4/4 takt je v hudbě považován za určitou výchozí hodnotu taktu. Z toho se dá předpokládat, že většina dat neuvádějící explicitně hodnotu proměnné time_signature, uvažuje 4/4 takt bez nutnosti zápisu jeho hodnoty do metadat. Nakonec jediná alternativa vzorkování a mapování piano-rollů je použití absolutní časové hodnoty v řádu milisekund, což nezaručuje zahrnutí veškerých hraných tónů.

Posledním důležitou vlastností datasetu je rozdělení stupnic jednotlivých skladeb. Tato analýza je důležitá z pohledu předzpracování dat pro trénování. Obdobně jako při analýze délky taktu i v tomto případě jsme limitování úplností metadat jednotlivých midi souborů. Každá midi stopa by měla obsahovat inicializaci stupnice. Zde lze obdobně předpokládat výchozí hodnotu v případě absence metadat a to C-dur respektive A-moll. Opět v průběhu stopy lze mezi jednotlivými stupnicemi přecházet, což se děje mnohem častěji než u délky taktu. Výsledek analýzy je zobrazen v tabulce 4.2.

Stupnice	**Výskyt (suma)**	**Stupnice**	**Výskyt (suma)**
C-dur	257	A-moll	6
C#-dur	30	; Hb-moll	4
D-dur	66	H-moll	3
D#-dur	90	C-moll	7
E-dur	35	Db-moll	4
F-dur	80	D-moll	10
F#-dur	9	Eb-moll	6
G-dur	83	E-moll	7
G#-dur	30	F-moll	9
A-dur	41	Gb-moll	5
A#-dur	40	G-moll	8
H-dur	30	Ab-moll	3

Tabulka 4.2: Rozdělení výskytu stupnic

Z tabulky 4.2 je zřejmá dominance durových stupnic oproti mollovým. Důležité ovšem je, že každá jednotlivá stupnice je zastoupena alespoň jednou. Nejfrekventovaněji se v datasetu vyskytuje stupnice C-dur.

4.2 Předzpracování

Cílem předzpracování datasetu bylo data co nejvíce zjednodušit bez ztráty relevantních informací a transformace midi souborů do .npz formátu obsahující číselný zápis piano rollů jednotlivých písní.

Základními úpravami hudebních dat pro efektivnější trénování je omezení notového rozsahu a transpozice skladeb do požadované stupnice. Po sestavení piano rollů jedné písně získáme vektor o dimenzi délka sekvence x časový rozsah x tónový rozsah. Kde tónový rozsah piana je 88 tónů. Konečné piano rolly jsou řídké matice (většina obsahu je prázdná nebo nulová). Důvodem je velmi malý poměr not, které jsou v daném časovém okamžiku t hrány vůči nepoužitým tónům. Navíc, jak je vidět, v tabulce 4.1 jsou některé tóny v celém datasetu hrány jen velmi zřídka a po většinu času jsou stále nulové. Řídké matice jsou obecně špatným trénovacím vstupem neuronových sítí. Většina jejich obsahu totiž neobsahuje žádné relevantní informace. Z těchto důvodů se originální notový rozsah nástroje často uměle zkracuje ve všech datech před začátkem trénování. Z obou stran rozsahu zvolíme úsek o určité velikosti, který bude odebrán a noty, které se v něm nacházejí, jsou následně posunuty o odpovídající počet oktáv směrem k validnímu rozsahu. Pomocí této úpravy zmenšíme vstup a řídkost vstupních matic se zmenší.

Plný rozsah piana je 88 tónů číslovaných podle MIDI formátu jako noty 21-108. Tóny nespadající do tohoto rozsahu jsem také transformoval do vyšších/nižších oktáv, z důvodu možné ztráty relevantních informací v případě jejich ignorování. Při výběru optimálního rozsahu jsem se řídil rozdělením tónů v datasetu a jinými projekty generující hudební obsah. Konečným zvoleným intervalem validních tónů je 36-96. Tedy celkem 60 tónů. Z každé strany tónového intervalu originálního datasetu je 14 tónů transformováno do optimálních oktáv. Těmito oktávami jsou C2-C6 včetně samostatné noty C7.

Transpozice stupnic písní se provádí ze dvou důvodů. Prvním je zjednodušení datasetu pro zefektivnění trénovací procedury. V tomto případě se běžně všechny skladby transponují do C-dur respektive A-moll. Tím se snižuje riziko vygenerování tónu, který ve hrané stupnici není validní. Opačně, pokud chceme síť účelně naučit všem známým stupnicím a tím zajistit správnou znalost jejich tónových rozdělení, transponuje se obvykle celý dataset do všech stupnic. Tím zároveň dojde i k obrovskému zvýšení počtu trénovacích dat. V práci jsem se rozhodl pro první variantu. Mou největší obavou během trénování sítě byla přílišná komplexnost vzájemných závislostí jednotlivých tónů a již výběrem datasetu jsem se snažil o co největší simplifikaci úlohy sítě.

Pro transpozici je potřeba zjistit stupnici, ve které se daná skladba nachází. V případě více použitých stupnic je třeba získat všechny jejich názvy a časy začátku/ukončení každé stupnice. V případě, kdy v midi souboru není nalezena žádná informace o použité stupnici, používám knihovnu music21, která nabízí možnost odhadu stupnice písně. Odhad je implementován algoritmem od Carol Krumhansl a Mark A. Schmuckler [1], který porovnává tónové rozdělení písně s tónovým rozdělením jednotlivých stupnic. Výsledkem je stupnice, která je rozdělení tónů skladby nejblíže.

Pro transpozici stupnic je nutno spočítat vzdálenost tóniky stupnice skladby vůči

tónice cílové stupnice. V tomto případě jsou cílovou tónikou noty C-dur/A-moll. Vzdálenost je počítána vůči nejbližší cílové tónice vzhledem k sousedním oktávám. Pokud je vzdálenost počítána směrem k nižšímu tónu, má kladnou hodnotu. V opačném případě je záporná. Po jejím získání je každý tón vstupní skladby posunut o tuto vzdálenost, kde znaménko vzdálenosti opět značí směr posunu. Tímto způsobem je tónika vstupu posunuta na tóniku cíle. Ostatní noty skladby jsou stejným způsobem posunuty. Po konečné transpozici získáme stejnou skladbu bez povýšených/snížených tónů, kromě tónů, které byly posunuty/sníženy bez ohledu na stupnici ve vstupní skladbě.

Konečné předzpracované písně znějí velmi podobně originálům. Hlavním rozdílem je celkový dojem, který píseň vytváří. To je způsobeno převážně změnou stupnice. Absence falešných tónů značí, že transpozice je implementována správně.

Nakonec je potřeba upravený dataset převést do vstupního formátu pro samotnou síť. Tímto formátem jsou .npz soubory, představující .zip archiv .npy souborů s podporou líného nahrávání. .npy formát je používán populárním výpočetním frameworkem Numpy, navrženým pro python. Vytvoření piano rollu probíhá v kooperaci s knihovnou music21, která vrací matici piano rollu právě v Numpy objektu. Ten lze snadno perzistovat do .npy/.npz souborů.

Při trénování neuronových sítí pomocí velkých datasetů je standardem využití mini-batch dávkování. Tato metoda umožňuje síť trénovat pomocí malých dávek několika paralelně příchozích vstupů. Chybová funkce je poté průměrována přes celou dávku a výsledný průměr je využit pro aktualizaci vah. Výhodou metody je výrazné zrychlení trénovací sekvence. Nevýhodou je určitá nepřesnost aktualizace vah z důvodu průměrování chyby.

Pro trénování sítě pomocí mini-batch dávkování, je nutno zajistit stejnou dimenzionalitu vstupních dat. Toho docílíme zkrácením všech skladeb v datasetu na dobu nejkratší trénovací písně, nebo rozdělení písní do několika konstantně dlouhých dílů. Rozhodl jsem se pro druhou možnost. Velikost jednotlivých dílů jsem v průběhu trénování několikrát upravoval pro co nejlepší výsledky. Poslední použitou hodnotou je doba deseti taktů. Vstup sítě bude mít dimenzionalitu velikost dávky x 10 x počet vzorků na takt x 60 (tónový rozsah skladby).

Velikost dávky je hyperparametr a v závislosti na kvalitě trénování je upravován. Pro vytvoření jednotlivých piano rollů je třeba ještě definovat počet vzorků na takt. Ve většině volně dostupných projektů lze nalézt použití 96 vzorků. Číslo 96 totiž lze beze zbytku dělit většinu standardních délek taktů, jako jsou čísla 2, 4, 8 a 16. Navíc je dostatečně velké pro zachycení šestnáctkové noty a tripletu ve většině taktů. Koncová dimenzionalita jednotlivých dílů skladeb je 10 x 96 x 60. V trénovací fázi je velikost vstupu rozšířena o velikost dávky. Bohužel LSTM měla s takto velkou vstupní sekvencí problémy. Hlavním problémem bylo neudržení kontextu dočasných závislostí v jednotlivých taktech. Vzhledem k tomu jsem snížil počet vzorků na 50. Tato hodnota se váže pouze ke čtení vstupu, nikoli k jeho vzorkování. Všechny relevantní informace jsou v piano rollech obsaženy a tóny jsou správně zachyceny. Jednotlivé rolly v trénování pouze nereflektují takty ani v případě 4/4 taktu.

5 Vytvoření a trénování neuronové sítě

Ve výběru architektury jsem se rozhodoval mezi GAN 3.2 potenciálně s nadstavbou sítě W-GAN 3.2.1 a variačním autoenkodérem (VAE) 3.3.2. Obě architektury jsou velmi efektivní za předpokladu správného učení. Obtížnost trénování je ovšem u obou sítí vysoká. V každé z nich spolu kooperují 2 vnitřní sítě. V GAN to jsou generátor a diskriminátor/kritik, u VAE to jsou kodér a dekodér. Zajistit správnou spolupráci vnitřních sítí je kritické, neboť pokud se jedna z nich učí rychleji než druhá, může začít ignorovat výstup druhé sítě a trénování skončí neúspěšně. VAE je obecně považována za méně složitou architekturu pro trénování a to z důvodu aktualizace vah obou sítí ve stejný okamžik. Oproti tomu GAN přidává nový trénovací hyperparametr k, který určuje poměr trénovacích iterací diskriminátoru vůči jedné iteraci generátoru. Zde je problém rovnováhy trénování komplexnější. Převážně z toho důvodu jsem se rozhodl pro implementaci VAE.

VAE ve své definici nespecifikuje typ sítí, ze kterých se skládá. Volba vnitřních sítí je závislá na typu dat. V případě generování hudby s piano roll vstupem jsou data sekvenční a obrazová. Pro sekvenční data jsou nejefektivnější rekurentní sítě, zatímco pro obrazová data jsou to sítě konvoluční.

Konvoluce mohou sloužit jako velmi efektivní extraktory relevantních příznaků z obrazových dat. Příznaky mohou být hrany, skvrny nebo barvy ve vstupu. Se zvyšující se hloubkou konvoluční sítě lze rozpoznávat i jednoduché objekty. V případě piano roll vstupu jsou tyto příznaky pro nás nedůležité. Konvoluce fungují na předpokladu velké míry závislosti mezi sousedícími prvky. Ovšem v případě hudby není vzdálenost not spolehlivou mírou závislosti. Harmonie tónů, ač řízena pevnými vzdálenostmi not, jako jsou tercie, kvinty nebo oktávy nelze s rostoucí vzdáleností ignorovat. Na druhou stranu, při použití velkého konvolučního jádra pro zachycení i velmi vzdálených harmonií, bychom riskovali ztrátu relevantních informací v důsledku rychlé redukce vstupu. Z těchto důvodů je vedena diskuse, zda jsou konvoluce v hudebním prostředí užitečné. V důsledku všech zmíněných problémů a nejistoty efektivity konvolucí jsem se rozhodl je v implementaci vynechat.

Oproti tomu rekurentní sítě jsou naprosto nezbytné pro rozpoznání a zapamatování dočasných závislostí hudebních dat. Kombinovaná architektura VAE a rekurentních sítí se nazývá variational recurrent auto-encoder [10], kde kodér i dekodér jsou nahrazeny rekurentní sítí. Zvoleným typem rekurence je LSTM.

Vstupem kodéru je dávka piano rollů o dimenzi [batch, 10, 50, 60]. Jednotlivé rolly každé dávky jsou použity jako vstup rekurentní sítě, která zpracuje celou sekvenci a jejím výstupem je poslední vnitřní stav h_n. LSTM nedokáže zpracovat vstup, pokud není ve formátu jedno-dimenzionálního vektoru. Vstup je tedy

ještě před zpracováním transformován do [batch, 10, 3000]. Tato transformace není příliš efektivní. Matice piano rollu velmi jasně reflektuje posloupnost hraných not přes několik časových kroků. Transformace do vektoru tyto informace zachová. Jsou však náročnější pro získání. Jsem přesvědčen, že právě tato transformace komplikovala LSTM trénování na piano rollech o 96 vzorcích na takt. Na h_n je aplikovaná aktivační funkce Tanh. h_n poté slouží jako vstup dvou nezávislých MLP sítí g a h, jejichž výstupem je střední hodnota a standardní odchylka normálního rozdělení vstupních dat. Výstupy sítí g a h následně slouží pro získání latentní proměnné z pomocí reparametrizace 3.17.

Vstupem dekodéru je proměnná z a velikost vstupních dat. z představuje jediné informace, které o vstupní skladbě máme k dispozici. Tato data je nutno udržet relevantní po celou dobu generativní fáze. Pokud z pouze transformujeme na inicializační stav LSTM, není zaručeno, že je síť sama nepřepíše během aktualizace stavu. Proto je z navíc připojováno ke každému sekvenčnímu vstupu LSTM a slouží nejen jako inicializace, ale také jako vstup. Díky tomu LSTM nemá potřebu zachovávat z ve svých vnitřních proměnných a může využít celý svůj prostor pro kontext již vygenerovaných dat. Transformace z na inicializační stav probíhá průchodem jednovrstvou MLP sítí s Tanh aktivací. V každém časovém kroku je rekurentní síti dekodéru předán vstup. Jeho obsahem je spojení matic výstupu dekódování v minulém časovém kroku a latentního vektoru z. V prvním iteraci je vstupem pouze vektor z, zbylý prostor je doplněn nulami. Po získání výstupu LSTM je opět aplikována Tanh aktivační funkce a výsledný vektor je mapován zpět na rozměry originálního piano roll v jednovrstvé MLP síti. Pomocí konečné Sigmoid aktivace převedeme reálné hodnoty do pravděpodobností zahrání jednotlivých not. Výstup dekodéru má stejnou velikost jako vstup kodéru pro možnost porovnání kódovaného a dekódovaného obsahu při trénování. Po výpočtu chybových funkcí dochází k finálnímu převodu pravděpodobností tónů do binárních hodnot. Strategií pro převod je hraniční hodnota 50 procent, podle které se rozhoduje o zahrání tónu ve výstupu. V generativní fázi je velikost vstupu simulována požadovanou délkou sekvence.

5.1 Trénování

Pro optimalizační algoritmus jsem zvolil Adam. Rozšíření originálního gradient descent algoritmu podporující jak Adaptivní gradient (AdaGrad), tak Root mean square propagaci (RMSProp). Adam byl použit i v původní práci o variačních rekurentních autoenkodérech [10] pro generování hudby. Beta parametry algoritmu jsou nastaveny také podle zmíněné práce. Beta1 o hodnotě 0.05 a Beta2 0.01. Tyto hodnoty jsou frekventovaně využívány i ve volně dostupných projektech využívající variační rekurentní autoenkodéry.

Prvotní learning rate je nastaven na 0.001. V průběhu trénování je postupně snižován. Ke snížení dojde vždy po 30 epochách. Nová hodnota learning rate je vždy 50% původní hodnoty. Celková délka trénovací sekvence je 200 epoch. Pro měření míry vzdálenosti latentního prostoru od $N \sim (0, 1)$ jsem použil KL divergenci. Chybová funkce rekonstrukce je měřena binární cross-entropy funkcí, která

oproti MSE funkci lépe kvantifikuje rozdíl binárních proměnných. Modle je prvních 20 epoch trénován pouze na rekonstrukční chybové funkci. Poté k celkové chybě přidána i funkční hodnota latentní chybové funkce násobena hyperparemetrem β. Beta je inicializována na 0 a v průběhu dalších sta epoch je postupně navyšována do hodnoty 100. Posledních 80 epoch model trénuje na neměnné chybové funkci.

Prvním zásadním problémem bylo vytvoření architektury popsané v kapitole 5. I přes velkou popularitu variačních autoenkodérů existuje velmi málo informací o jejich rekurentní implementaci. Nízká míra využití této architektury značně zkomplikovala veškeré pokusy o dohledání jakékoli relevantní informace. Čerpal jsem převážně z originální práce [10] a z několika málo volně dostupných zdrojových kódů projektů využívajících variační rekurentní autoenkodér. Většina implementací byla ovšem určena pro generování textu a byla pro toto zadání upravena. Nejvíce informací jsem získal z projektů VariationalRecurrentAutoEncoder od Arunesh Mittal [21] a pytorch_RVAE od Daniil Gavrilov [11]. Myšlenka připojení latentního vektoru ke každému vstupu rekurentní sítě dekodéru je převzata právě z projektu pytorch_RVAE [11], který ji převzal z originální práce citované na github stránce projektu.

Funkcionalitu modelu jsem ověřoval přeučením autoenkodéru na jedné samostatné skladbě. Pokud by síť dokázala po několika trénovacích iteracích rekonstruovat kompletní vstupní skladbu pomocí přeučení nad jedním datovým bodem, lze předpokládat korektnost implementace a spustit trénování nad celým datasetem. Tímto způsobem je možné vyhnout se zbytečnému, několikahodinovému trénování nesprávně implementovaného modelu. Problémem byla skutečnost, že i po kompletní rekonstrukci samostatné písně model selhával na kompletním datasetu. S rostoucí velikostí trénovacích dat bylo selhání znatelnější. Zde jsem udělal chybu, neboť i nevhodná architektura po opakovaném přeučení je schopna naučit se rozdělení jedné písně. Ovšem může naprosto selhat v obecném rozpoznávání a kódování hudebních závislostí. Po opakovaném přepracování celé implementace se mi podařilo problém odhalit. Ověření správné funkcionality modelu jsem poté testoval trénováním nad plným datasetem, ovšem pouze s jedním aktivním výrazem v loss funkci. V případě ověření schopnosti rekonstrukce původních dat jsem ze sítě vyřadil omezení latentního prostoru a naopak.

Velmi dlouhé trénovací časy značně zpomalovaly rychlost vývoje a optimalizace. 100 trénovacích epoch trvalo obvykle cca pět hodin. Teprve po této době jsem byl schopen porovnat výsledky a patřičně upravit hyperparametry pro další pokus. Obecně při trénování neuronových sítí je finální trénovací sekvence znatelně delší než 5 hodin, ovšem ve fázi otimalizace a testování se jedná o velké zdržení. Tento problém jsem adresoval použitím 1/5 datasetu jako testovací množiny dat, nad kterou mohu architekturu optimalizovat. Během optimalizace jsem byl nucen model opět mnohokrát upravovat a přepracovávat. Bez ohledu na hodnoty hyperparametrů a dimenzionalitu učených parametrů síť nedokázala konvergovat k optimálním hodnotám. Příčinou této vady nebyla chyba na straně sítě. Pro správnou funkcionalitu variačního autoenkodéru je nutné během trénovací procedury co nejlépe definovat význam jednotlivým bodům latentního prostoru. Toho lze docílit pouze za použití velkých datových objemů, neboť právě variace mezi trénovacími daty slouží k vy-

tvoření nových hodnot vektoru z, ve kterém se dekodér učí rozpoznávat hudebně relevantní informace. Při použití datasetu o malé velikosti zůstává latentní prostor z velké části nevyužit a náhodné vzorkování z $N \sim (0, 1)$ nelze dekódovat do smysluplného obsahu. Myšlenku testovacího datasetu pro účely optimalizace jsem zavrhl a pokračoval v pomalé optimalizaci nad plným datasetem.

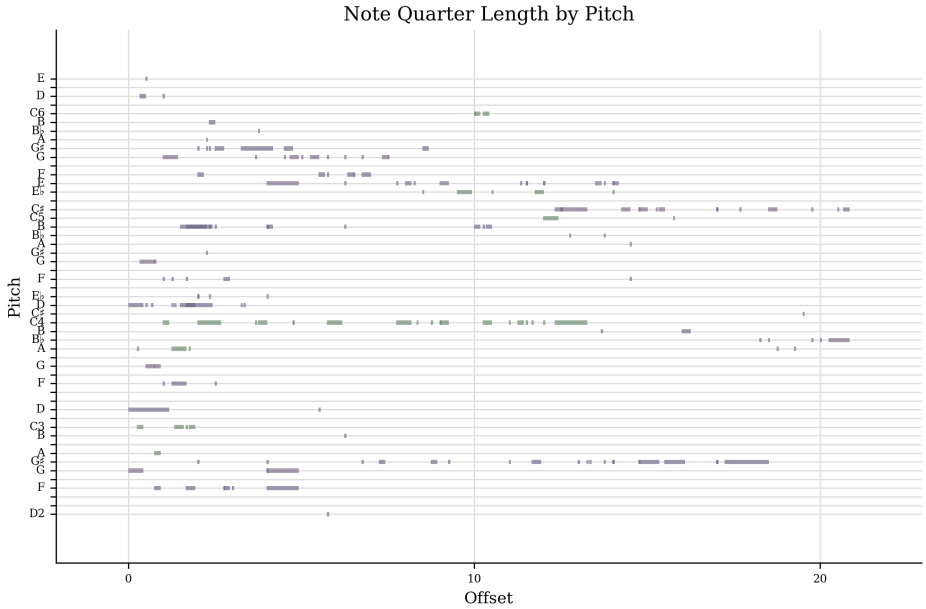
Další limitací byla výpočetní technika, kterou jsem pro trénování použil. Veškeré výpočty jsem prováděl na grafické kartě Nvidia GeForce GTX 1060. Vnitřní šesti gb paměť karty zdaleka nestačila pro uložení celého modelu, včetně dávek vstupních piano rollů. Tuto limitaci jsem zpočátku řešil omezením parametrů architektury a menší velikostí mini-batch dávek. Síť za těchto podmínek nemohla pracovat s plným potenciálem, protože pro správné fungování potřebovala větší prostor. Z toho důvodu jsem architekturu přenesl na Google Drive a pro trénování využil službu Google Colab. Colab umožňuje využití vzdálené výpočetní techniky poskytnuté společností Google. Pomocí předplacené verze Google Colab lze trénovat až 24 hodin bez rizika odebrání prostředků. Toto řešení velmi urychlilo trénování a usnadnilo experimentování s hyperparametry a velikostí prostoru síti poskytnuté pro co nejlepší výsledky.

Po ověření správné funkcionality kodéru a dekodéru bylo nutné zajistit vzájemnou spolupráci obou sítí pro dosažení optimálního řešení. Právě tato úloha činí trénování variačních autoenkodérů natolik obtížné. Úlohou kodéru je minimalizovat vzdálenost latentního prostoru od $N \sim (0, 1)$. Dekodér upravuje své parametry pro nejlepší porozumění jednotlivých prvků vektoru z. Zároveň posílá kodéru zpětnou vazbu pro aktualizaci latentního prostoru tak, aby rekonstrukce byla maximálně přesná. Tyto dva úkoly jsou v rozporu. Přesunutí latentního prostoru co nejblíže standardnímu normálnímu rozdělení je pro kodér snadná úloha, vyřešitelná v jednotkách epoch. Dekodér oproti tomu posílá signály o nastavení latentního prostoru tak, jak mu nejlépe vyhovuje. Výsledkem tohoto rozporu je, v případě nevyváženosti hodnot obou loss funkcí, velmi rychlé převzetí kontroly nad celou architekturou dominantních sítí. Pokud latentní loss výrazně převyšuje rekonstrukční, konverguje velmi rychle k nule a nenaučí se žádné smysluplné reprezentace informací pro dekodér. Pokud však rekonstrukční loss převyšuje latentní, je kodér donucen díky velmi vysokým příchozím gradientům nastavit latentní prostor pro co nejlepší rekonstrukci. Pro adresaci toho problému se využívá hyperparametr β, kterým je funkční hodnota latentní chybové funkce násobena. Je-li $\beta = 1$, je architektura ekvivalentní klasickému variačnímu autoenkodéru. Změna hodnoty β slouží k vyvážení hodnot loss funkcí. Ideálně tak, aby se vzájemně nevyrušily. Hledání rovnováhy mezi oběma funkcemi je jedním z nejdůležitějších prvků trénování. Ve většině existujících projektů lze nalézt hodnotu β hyperparametru pohyblivou. Z počátku je pro zajištění relevance informací v latentním prostoru spuštěno několik epoch s $\beta = 0$. Následně se její hodnota zvyšuje k cílové hodnotě velmi často větší než 1. Ke konci trénování zůstane β po několik epoch konstantní pro dokončení kovergence latentního losu. Konkrétní cílové hodnoty β se liší dle implementace.

Po ukončení trénování ukládám parametry dekodéru do .pth souboru. Tento formát je standardem pro persistenci parametrů neuronových sítí reprezentovaných pytorch objektem. Ve fázi generování inicializuji síť bez kodéru. Dekodér je inici-

alizován optimalizovanými parametry. Jeho vstupem je latentní vektor z naplněn vzorky ze standardního normálního rozdělení a počet piano rollů konečné, vygenerované sekvence. Výstup je po převodu do binárních hodnot uložen jako midi soubor.

Mé pokusy o vyrovnání obou chybových funkcí pomocí hyperparametru β skončily nezdarem. Kodér i dekodér svůj úkol samostatně plní velmi dobře. Společné učení obou sítí je velmi nevyvážené. Model vždy dokonverguje do hraničního bodu, ve kterém se bez ohledu na hyperparametry snižuje latentní loss na úkor rekonstrukce. Dekodér, naučený převážně na prvních 20 epochách začne ztrácet kontrolu nad latentním prostorem a nedokáže s ním udržet krok. Z toho důvodu je konečná kvalita vygenerovaných skladeb velmi nízká. Prvních 50 časových kroků je téměř vždy zaplněno velkým množstvím náhodných tónů, neboť skutečný latentní prostor je stále vzdálen $N \sim (0, 1)$. Sekvence pokračuje neúčinnými pokusy o nápravu, jejíž jediným výsledkem je redukce hustoty náhodných tónů. Mimo tuto redukci jsou ve výstupu slyšitelné pokusy o vedení melodie a občasný akord. Bohužel zašumění náhodnými tóny je příliš znatelné. Ve výsledku jsou vygenerované skladby neposlouchatelné. Je pravděpodobné, že pro správnou funkci a trénování sítě by postačilo správně nastavit dimenzionalitu kodéru a dekodéru, upravit learning rate a experimentovat s *beta* parametrem. Po mnoha pokusech o správnou konfiguraci modelu mi nezbyl čas a bylo třeba práci ukončit. Příklad vygenerované skladby je ilustrován na obrázku 5.1. Další příklady výstupu sítě lze nalézt na přiloženém DVD nebo v souborech elektronické verze práce v systému is/stag.



Obrázek 5.1: Výsledek trénování - piano roll

6 Porovnání s volně dostupným řešením

Jedna z dosud nevyřešených výzev návrhu kreativních neuronových sítí je jednoznačná kvantifikace výsledku pro měření kvality. Mezi tyto sítě patří generátory hudby, style transfer modely, nebo generátory uměleckých obrázků. V hudebním zaměření je za standard považováno subjektivní hodnocení vzorkem nezávislých pozorovatelů, hodnotící původ nebo kvalitu hudby. Velmi populárním řešením v okruhu hudebních generátorů je hudební Turingův test. Obdobně jako v obyčejném Turingově testu je pozorovatelům kladena otázka, zda obsah jim prezentovaný vytvořil lidský autor, nebo počítačový systém. Při hudebním Turingově testu je navíc brán zřetel na výběr prezentovaných nahrávek, prostředí jejich poslechu a znalosti jednotlivých posluchačů v oboru hudební teorie. Tyto úrovně znalostí jsou často seskupeny do několika kategorií, jejichž počet a rozdělení určuje míru komplexnosti testu, obdobně jako velikost vzorku posluchačů. Dle úlohy modelu neuronové sítě může být otázka testu upravena. Často používanou alternativou je dotazování na míru estetičnosti jednotlivých nahrávek. I přes frekventované používání Turingova hudebního testu jsou zde nevýhody pramenící z jeho použití. Výsledek testu není jednoznačným určením kvality. Pokud pomineme statistický původ a subjektivní hodnocení posluchačů, projekty používají různá kritéria vyhodnocení testu. Posluchačům jsou kladeny rozdílné otázky. Přístup k rozdělení znalostí hudebního oboru posluchačů je často podceněn, nebo je použita malá velikost vzorku. Pravděpodobně vzhledem k nákladnosti testu. Z těchto důvodů je možno považovat výsledky testu pouze jako prvotní pokusy o evaluaci.

I přes výraznou dominanci kvalitativních technik měření kvality modelů bylo implementováno velké množství pokusů o kvantitativní postup. Některé metody jsou zaměřeny více na pravděpodobnostní analýzu, jako např. měření negativní logaritmické věrohodnosti vygenerovaných dat vůči použitému datasetu. Jiné techniky cílí spíše na analýzu pravidel hudební teorie, podle kterých lze formulovat obecné závěry o schopnostech modelu. Hlavním nedostatkem těchto metod je neúplnost jejich hodnocení. Model pozitivně hodnocený podle jednoho kritéria nemusí splňovat kritérium jiné.

Vzhledem k ne příliš přesvědčivé kvalitě výstupů sítě jsem nebyl nakloněn myšlence kvalitativního porovnání. Na každém jednotlivém výstupu lze velmi rychle poznat systémový původ převážně z důvodu velkého šumu náhodných tónů. Ze stejného důvodu se ze skladeb vytrácí estetika a dotazy směřující k jejímu ohodnocení ztrácí smysl. Oproti tomu vygenerované skladby mají určitou strukturu a pokusy o melodii, kterou si snaží udržet po celou dobu hraní. Síť se také dobře naučila hrát jednotlivé akordy, které jsou v doprovodu melodie často slyšet.

Použil jsem některé z metrik prezentovaných v [35], jejíž cílem je snaha o vytvoření několika kvantitativních metrik pro ohodnocení a porovnání modelů generující hudbu. Spočteny jsou základní metriky navržené pro získání relevantních informací o dodržení pravidel hudební teorie. Na základě výsledků je získána absolutní a relativní míra porovnání mezi výstupními daty dvou modelů. Všechny postupy navrženy v [35] jsou podle autorů aplikovatelné pouze pro monofonní skladby. Důvodem je problémová aplikace metrik, které cílí na vlastnosti po sobě jdoucích tónů, jako jsou např. intervaly. Polyfonní skladba dovoluje použití několika tónů ve stejný čas. Tím se může vytvořit k hlavní melodii doprovod. Výsledná skladba proto obsahuje individuální sekvence tónů a je nutno využít určitý způsob clustrování. V případě použití metrik zkoumající chování po sobě jdoucích tónů tedy vyhledávám další zahraný tón pouze v rozsahu dvou oktáv. Tím vzniká okno pozorování následujícího tónu, které se posouvá s dalším zahraným tónem. Tato technika není ideální, neboť pokud je doprovod situovaný velmi blízko k melodii, dojde k nežádoucímu efektu prolnutí obou sekvencí.

6.1 Aplikovaná měření

Provedená měření jsou navržena pro získání přehledu rozdělení tónů a rytmických vlastností vygenerovaných skladeb. Měření je provedeno pro každou skladbu zvlášť. K metrikám zkoumající tónové vlastnosti patří počet tónů, histogram not a matice přechodu not. Mezi metriky zkoumající rytmické vlastnosti modelu patří průměrný čas mezi zahráním dvou po sobě jdoucích not, histogram délek not a matice přechodu délek not.

V porovnání s [35] jsou některé metriky upraveny, jiné nejsou použity. Mezi vyloučené patří:

- Rozdíl minimálního a maximálního tónu.
- Průměrný interval mezi tóny.
- Počet unikátních not.

V případě histogramů jsou přidány sloupce pozorující stejnou vlastnost u trénovacího datasetu modelu pro určení míry úspěšného naučení vlastností vstupních dat. Z důvodu normalizace jsou hodnoty histogramů procentuální, nikoli absolutní. Všechna provedená měření se dají vysvětlit několika způsoby. Mnohdy není jednoduché určit, jaké hodnoty jsou pozitivní, což činí porovnání složitějším. Tato skutečnost je jedním z důvodů pro přidání vlastností původního datasetu do grafu histogramu.

Z jednotlivých měření je následně dopočtena absolutní míra porovnání výsledků obou modelů. [35] navrhuje průměr a standardní odchylku každého měření. Standardní odchylku z matice přechodu not a matice přechodu délek not jsem vyřadil, neboť neobsahovala žádné intuitivní informace. Důležitější část [35] je mířena k relativním metrikám, které jsou navrženy pro porovnávání modelů natrénovaných na rozdílných datasetech. Všechny zmíněné metriky jsou velmi silně ovlivněny vůči použitému datasetu. Zatímco absolutní metrika je jednoduché porovnání získaných

vlastností, v relativních metrikách jsou odhadnuty pravděpodobnostní rozdělení výsledků měření a je získána plocha průniku těchto rozdělení. Tento systém porovnání jsem se rozhodl neimplementovat z důvodu nemožného zajištění validity výsledků měření předpokládající monofonní vstup. Abych minimalizoval problém rozdílných datasetů, rozhodl jsem se vybrané modely natrénovat na svém datasetu.

6.2 Výsledky porovnání

Svůj natrénovaný model jsem porovnával s dvěma projekty volně dostupnými na veřejných github repozitářích. Prvním je projekt [34] od Dustina Wright, který má podobnou architekturu. Generátor používá variační autoenkodér s rekurentním dekodérem implementovaným pomocí GRU sítí. Rozdílem vůči mému řešení je konvoluční kodér. Projekt používá piano-roll reprezentaci dat bez předzpracování. Vzhledem k mé nejistotě použití konvolučních sítí, jsem chtěl porovnat konvoluční kodér s rekurentním a zároveň získat srovnání obou autoenkodérů. Trénování této sítě probíhá nad jedním midi souborem. Síť tedy nedokáže vygenerovat naprosto jedinečný výstup. Navzdory tomu každá vygenerovaná skladba má odlišnou strukturu a i přes podobnost s trénovací písní dokáže zajistit individuálnost výstupu. Model jsem trénoval po 1000 epoch. Použité parametry jsou stejné jako autorovy. Výstup sítě byl překvapující. Nedošlo k přeučení sítě, ovšem kvalita jednotlivých skladeb byla horší než jsem čekal. Mým předpokladem bylo, že síť potřebuje pro správné trénování dostatečně dlouhý vstup s velkými variacemi. Tento předpoklad byl mylný, neboť po předání 12 minut dlouhého vstupu pro trénování s častými odchylkami od původní melodie nedokázal model vyprodukovat žádné výsledky, krom několika náhodných not. Ostatní výstupy obsahují akordy doprovodu a místy i zajímavou melodii neokopírovanou z trénovacího vstupu. Ovšem po většinu času není nic přehráváno. Píseň po chvíli přestane hrát a po několikasekundové pauze opět začne velmi nejistými tóny, které znějí místy až falešně. To je pravděpodobně důsledek silné regularizace. Síť je nucena získat ze vstupu pouze inspiraci pro svoji inovaci. Zkouší zahrát některé tóny, ale nedokáže podle nich vytvořit globální kompozici. I přes to jsou vygenerované výsledky kvalitnější než výstup mé sítě. V porovnání je tento projekt odkazován pod jménem cnn-vrnn.

Druhý projekt [18] od Yuankui Lee jsem zvolil z důvodu velmi dobrých výsledků generování. Model je inspirován úspěšnou sítí vyvinutou v rámci Magenta projektu 2.1.2. Model je tvořen pouze rekurentními sítěmi typu LSTM. Jako vstup používá midi soubory transformované do sekvence midi událostí typu Note on - začátek tónu, Note off - konec tónu, Time-shift - postup v času a Velocity - síla tónu. V rámci předzpracování jsou posunuty konstanty ovlivňující čas a tempo písní. Skladby jsou transformovány do rychlejších i pomalejších verzí. Navzdory použití pouze LSTM sítí dosahuje model vysoce kvalitních výsledků, které by byly vhodné i pro hudební Turingův test. Původní dataset modelu obsahoval 445 midi souborů klasické hudby, které jsem vyměnil za stejný počet souborů mého datasetu. Po předzpracování jsem spustil trénovací proceduru po dobu 2800 epoch. Parametry trénování jsou ponechány stejné jako autorovy. Performance-RNN umožňuje dva typy generování.

Počet tónů	Průměr	Standardní odchylka
Variační rekurentní autoenkodér	38.65	6.41
cnn-vrnn	32.3	24.52
Performance-RNN	23.1	11.82

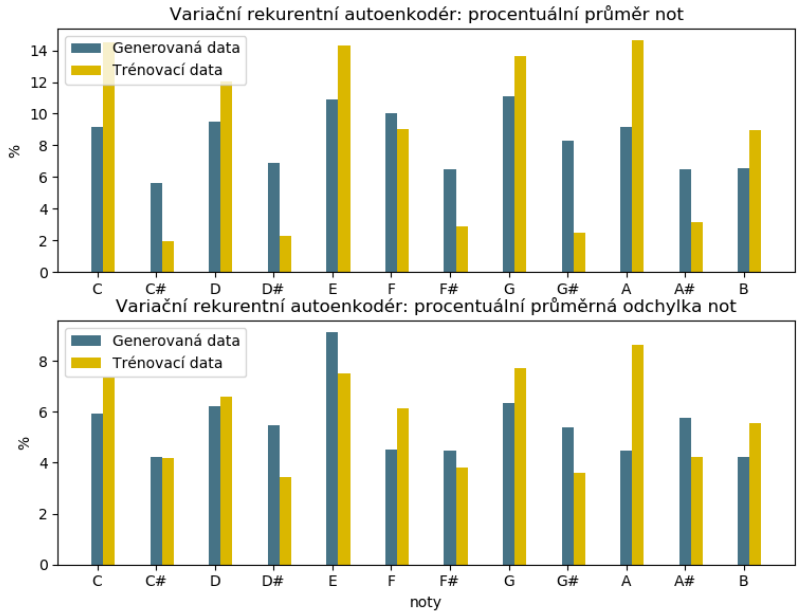
Tabulka 6.1: Počet unikátních tónů v porovnávacích datasetech

V prvním je model inicializován náhodnými čísly a na uživateli je zadání velikosti výstupní sekvence. V druhém případě je síti předán předzpracovaný midi soubor pro inicializaci modelu a délka výstupní sekvence je stejná jako délka vstupu. Výstupy obou generování nejsou zašuměny náhodnými tóny a nevyskytuje se v nich téměř žádný falešný tón. V případě náhodné inicializace ovšem skladba velmi rychle ztrácí kreativitu. Po několika sekundách velmi kvalitního výstupu se pravděpodobně rekurentní síť modelu zacyklí a zbytek skladby je pouze opakováním několika not nebo akordů až do konce. V případě inicializace inspirativním vstupem je výsledek různý. Ve dvou případech došlo ke zmiňovanému zacyklení hned od začátku písně. Ve většině případů se ovšem síti podařilo vygenerovat výstup v podobné tónině jako vstup, ovšem jinak velmi odlišný a silně strukturovaný. V těchto výstupech se vyskytují komplexní melodie a přesné akordy doprovodu. Pro inicializaci generování jsem použil soubory mimo trénovací dataset. Obecně výstup této sítě převyšuje můj model ve všech ohledech. V porovnání je tento projekt odkazován pod jménem Performance-RNN.

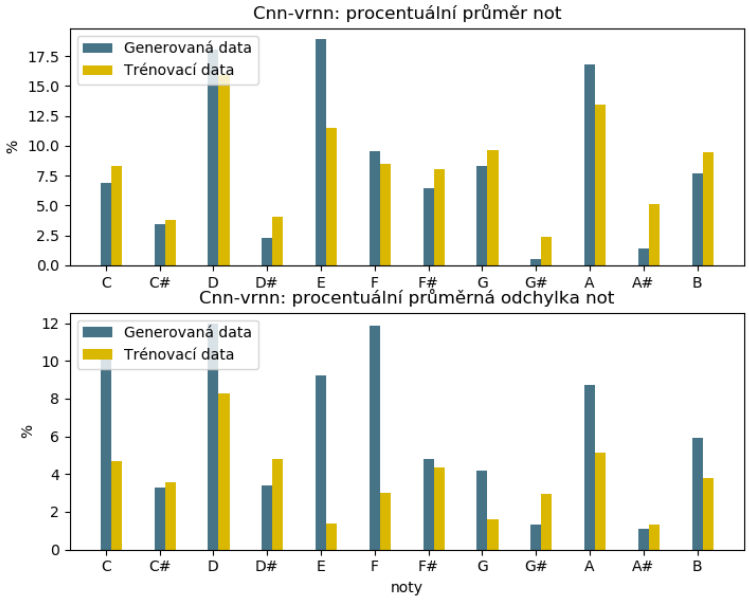
Pro samotné kvantitativní porovnání jsem vygeneroval 20 souborů ze všech tří architektur. V případě [34] je síť trénována celkem pětkrát, pokaždé na jiném vstupním souboru. Po každé tréninkové iteraci jsou vygenerovány 4 soubory pro porovnání. Při generování ze sítě [18] je použita desetkrát náhodná inicializace a desetkrát inicializace vstupním souborem. Výstupní soubory mé architektury jsou vybrány z několika úspěšnějších trénovacích procedur. Na těchto skladbách je šum náhodných not nejméně patrný. V porovnání je můj projekt odkazován pod jménem Variační rekurentní autoenkodér.

V tabulce 6.1 jsou vidět výsledky porovnání počtu unikátních tónů. Tato metrika nám dává intuitivní náhled do hustoty použitých frekvencí v průměru na skladbu. S rostoucím průměrem model generuje více tónově různorodé skladby. Velký průměr ovšem může také značit přítomnost náhodných tónů. Obecně i v případě nízkého průměru může mít skladba silnou strukturu a skvělou kompozici. Dobrým příkladem jsou jednodušší klasické písně, které nepotřebují velkou zásobu tónů pro unikátní skladbu. Díky rozsáhlé problematice sítí adresovat náhodný šum not lze asociovat horší výsledky spíše pro velké průměry. Velká odchylka v tomto případě značí, že ne všechny písně modelem generované musí tímto problémem trpět.

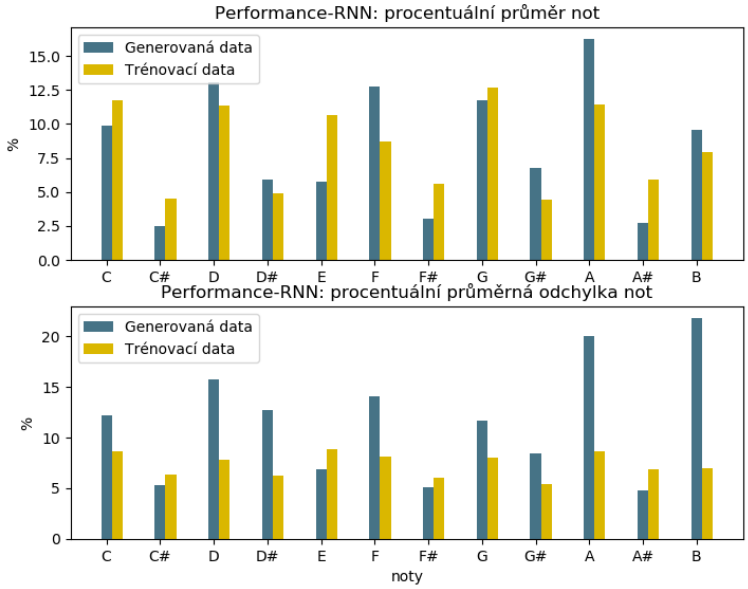
Variační rekurentní autoenkodér má v porovnání s ostatními modely maximální průměr a minimální odchylku. Tato skutečnost přímo implikuje šum náhodných tónů a nedostatečné naučení rozdělení tónů v trénovacím datasetu. Tyto vlastnosti jsou pozorovatelné rovněž přímým poslechem. Ostatní modely si v porovnání vedou lépe. Nejlepší výsledek lze sledovat u Performance-RNN.



Obrázek 6.1: Histogramy not Variančního rekurentního autoenkodéru



Obrázek 6.2: Histogramy not modelu cnn-vrnn



Obrázek 6.3: Histogramy not modelu Performance-RNN

Na obrázcích 6.1, 6.2, 6.3 je vyobrazeno přímé porovnání rozdělení not vygenerovaných souborů a trénovacích datasetů pro všechny tři modely. Pro každý model je porovnáván procentuální průměr a procentuální průměrná odchylka výskytu not. Histogram nám ukazuje, jak dobře se model naučil notové rozdělení trénovacího datasetu. Zda model neupřednostňuje nebo neutlačuje některé noty a jakou stupnici se snaží dodržet.

Mimo noty nepatřící do C-dur stupnice varianční rekurentní autoenkodér dodržuje rozdělení trénovacího datasetu. Vysoké průměry těchto not ve výstupech značí nedokonalé trénování na vstupním datasetu. Na histogramu je patrné, že transpozice datasetu do C-dur stupnice opravdu výrazně snížila výskyt not mimo C-dur, ovšem model díky svému náhodnému chování tuto skutečnost přehlíží. Na druhou stranu procentuální průměrné odchylky jsou téměř identické s trénovacím datasetem. Model se tedy alespoň snaží o přibližné napodobení notového rozdělení. Performance-RNN trpí velkými odchylkami not A, B. Lze usuzovat, že právě tyto noty se z velké části podílí na cyklech rekurentní sítě modelu, které popisuji v 6.2. Mimo tyto noty je Performance-RNN model natrénován velmi dobře. V případě cnn-vrnn modelu jsou výstupní soubory po většinu času prázdné a hrají v nich jen některé tóny, které nedávají ostatním prostor. Dle vnitřního stavu sítě jsou tyto noty vybrány a tedy jejich výskyt se liší od písně k písni. Tato vlastnost se opět odráží na odchylkách not modelu. Lze si všimnout vysokého využití not C-dur stupnice. Tuto vlastnost má model společnou s Performance-RNN z důvodu trénování na datech transponovaných právě do této stupnice.

Tabulky 6.2, 6.3 a 6.4 obsahují mapování frekvence přechodu jedné noty na následující. Pomocí těchto matic můžeme pozorovat přístup jednotlivých modelů k intervalům. Na diagonále matic se nacházejí oktávy a primy, mezi kterými měření ne-

	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
C	**7.45**	0.9	1.65	2.05	2.	1.25	1.2	**2.55**	1.3	**2.15**	1.55	1.
C#	0.95	**3.05**	0.95	1.05	0.85	1.4	0.6	0.7	1.05	0.9	0.75	0.5
D	1.7	0.65	**6.7**	1.1	1.75	**2.35**	1.	**3.05**	1.05	1.9	1.1	1.1
D#	1.45	0.8	1.	**3.55**	1.7	1.1	1.	1.3	1.9	1.1	2.	1.05
E	1.55	0.6	1.5	1.55	**8.1**	2.	1.85	1.9	1.45	1.95	1.1	**2.05**
F	1.2	0.85	2.2	1.1	**2.05**	**6.3**	1.65	**2.9**	1.25	**2.4**	1.85	1.25
F#	1.1	0.85	1.05	0.8	1.65	1.8	**2.45**	1.6	1.4	1.15	0.75	1.1
G	**2.65**	0.8	**2.6**	1.1	1.7	3.3	1.6	**8.85**	1.45	**2.35**	1.4	1.55
G#	**2.25**	1.15	1.15	1.7	1.1	1.3	1.25	1.2	**5.25**	1.55	1.	0.95
A	**2.05**	1.3	**2.1**	1.2	1.95	1.95	1.1	1.9	1.	**5.9**	1.1	1.25
A#	1.25	0.95	1.4	1.75	0.95	1.35	0.8	1.85	1.3	0.8	**4.35**	0.35
B	1.05	0.7	1.2	0.75	1.9	1.2	0.85	1.9	1.	1.	0.55	**3.8**

Tabulka 6.2: Variační rekurentní autoenkodér: průměrná matice přechodu tónů

	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
C	**9.05**	1.45	3.65	2.1	4.65	3.65	1.95	3.45	1.85	4.35	2.05	1.95
C#	1.45	2.05	3.05	1.65	3.85	1.65	1.45	1.25	1.05	1.75	0.9	0.7
D	4.	1.95	**25.55**	2.1	**6.6**	**6.5**	**6.95**	4.	1.2	**10.1**	1.8	**5.6**
D#	0.95	1.1	2.55	2.45	2.45	1.95	1.35	1.35	1.35	0.9	1.5	0.6
E	4.25	3.2	**5.6**	1.85	**21.6**	3.45	**6.7**	**8.3**	0.9	**12.3**	1.25	**5.7**
F	2.15	1.7	**5.25**	1.2	4.05	**15.15**	2.4	4.55	2.35	4.7	3.45	1.4
F#	2.3	1.55	**7.25**	1.1	3.25	1.6	**7.95**	**5.5**	1.55	**5.6**	1.6	3.45
G	3.1	1.4	4.35	0.8	**7.95**	3.1	2.95	**7.8**	1.65	**6.6**	2.7	4.65
G#	2.	1.	1.5	0.95	1.1	1.8	1.05	0.65	3.2	1.55	2.65	0.85
A	**5.95**	2.85	**8.3**	1.2	**12.55**	2.9	4.55	**5.**	1.3	**23.3**	3.1	**7.05**
A#	2.75	1.5	1.85	1.4	1.25	4.2	1.05	1.5	2.35	1.85	4.7	1.3
B	3.25	1.	**6.55**	1.35	**6.45**	1.8	3.8	3.7	0.4	**5.85**	0.65	**8.5**

Tabulka 6.3: Cnn-vrnn: průměrná matice přechodu tónů

	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
C	**4.55**	0.35	0.65	2.45	2.95	**5.5**	0.25	1.35	0.75	**5.65**	0.9	0.35
C#	0.4	2.85	0.15	0.55	2.3	2.55	2.95	0.2	0.7	1.15	0.4	0.35
D	0.8	0.45	**11.1**	0.75	0.75	**3.65**	2.05	5.4	1.	2.3	0.45	**3.45**
D#	0.55	0.95	0.45	**38.1**	0.	0.55	0.9	2.65	**3.9**	0.05	0.95	0.1
E	1.05	0.3	0.8	0.	**7.9**	1.8	0.65	**3.45**	0.5	**6.1**	0.	1.1
F	**5.15**	1.3	0.85	1.35	0.6	**28.8**	0.4	1.25	**4.65**	**3.2**	**4.1**	1.35
F#	0.5	2.3	2.05	1.4	0.7	0.65	**5.3**	0.2	0.45	1.8	0.75	0.85
G	2.9	0.1	**3.15**	1.55	1.8	0.85	0.2	**29.1**	0.35	2.75	1.55	**3.45**
G#	2.25	1.2	**3.15**	**3.15**	0.15	2.05	0.4	0.15	**5.1**	0.3	0.75	1.25
A	**6.65**	**2.75**	3.1	0.05	**3.5**	2.75	2.15	1.3	0.45	**46.6**	0.1	1.65
A#	0.7	0.95	1.2	1.75	0.15	**3.25**	0.65	0.25	1.1	0.15	6.3	0.05
B	0.5	0.65	4.95	0.1	2.4	0.2	0.15	2.7	1.	1.1	0.	**99.9**

Tabulka 6.4: Performance-RNN: průměrná matice přechodu tónů

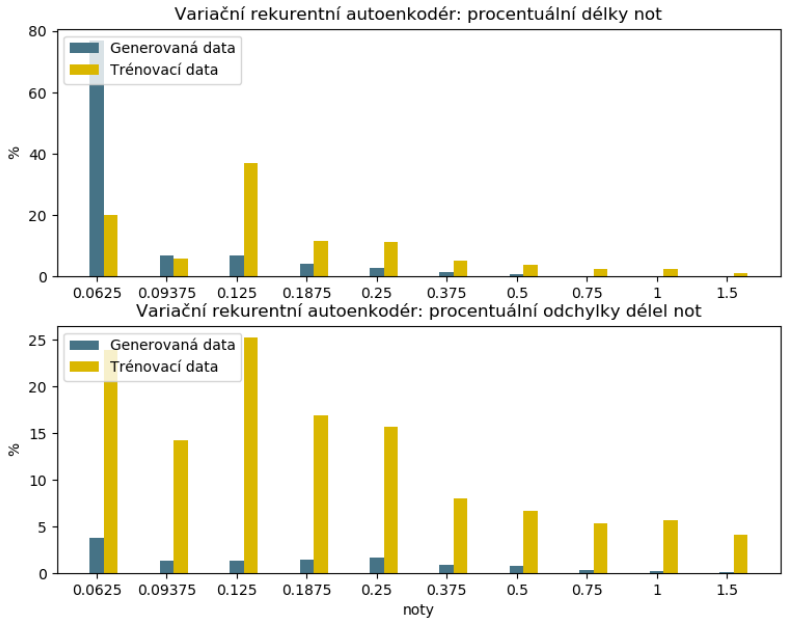
rozlišuje. V každé tabulce jsou pro orientaci zvýrazněny nadstandardní hodnoty. Pokud model využívá stejné intervaly u každé noty, je možno říci, že rozumí důležitosti konkrétního intervalu pro harmonii a pokouší se ho maximálně aplikovat bez ohledu na notu. V případě rovnoměrně distribuovaných hodnot lze očekávat velké množství náhodných not ve výstupu. Pomocí matic lze vypozorovat i jednotlivé akordy sledováním postupného přechodu několika not.

Variační rekurentní autoenkodér mezi intervaly příliš nerozlišuje. Téměř rovnoměrné hodnoty intervalů not implikují velkost míru náhodnosti. Jedinou výjimkou je oktáva/tercie, jejíž hodnoty jsou vychýlené od ostatních z důvodu častého opakování stejných not. Tato skutečnost je pozorovatelná i na obrázku 5.1. Často využívaná jsou rovněž A-D, C-G kvarteta, což jsou ale jediné zajímavé hodnoty. Cnn-vrnn model se oproti tomu snaží dávat vzdálenosti not více do souvislostí. Tato skutečnost může bý zapříčiněna konvolučním kodérem. Mimo diagonální hodnoty lze pozorovat tercie (např. D-F), kvarteta (např. E-B, D-A) a sexty(např. F-D). Performance-RNN model důležitost intervalů adresuje ze všech sítí nejvíce. V tabulce 6.4 nacházíme největší množství nadstandardních hodnot, jejíchž hodnota vysoce převyšuje zbytek matice. Model používá všechny typy základních intervalů a díky tomu dokáže dobře vyjádřit harmonii. Zajímavým extrémem jsou noty A, B v diagonální pozici.

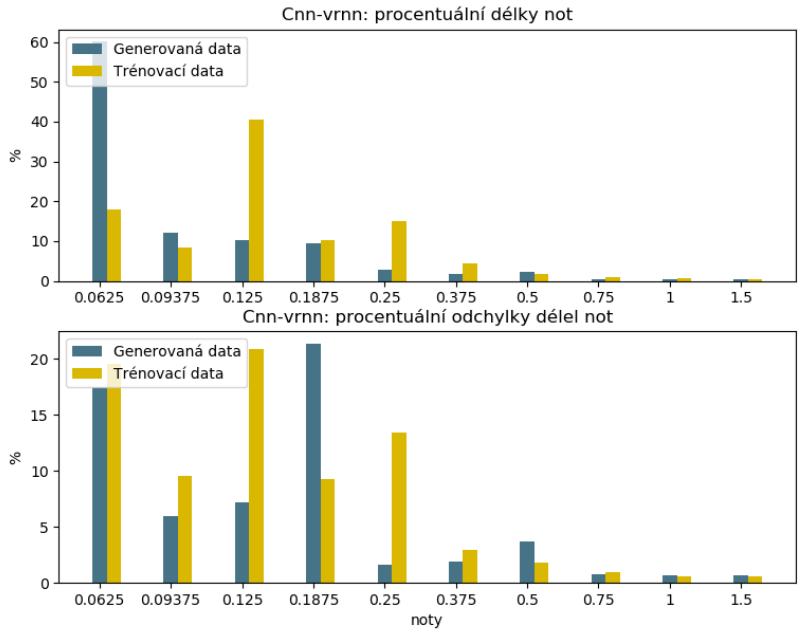
Čas mezi tóny	Průměr	Standardní odchylka
Variační rekurentní autoenkodér	0.092	0.030
cnn-vrnn	0.247	0.161
Performance-RNN	0.345	0.143

Tabulka 6.5: Čas mezi tóny

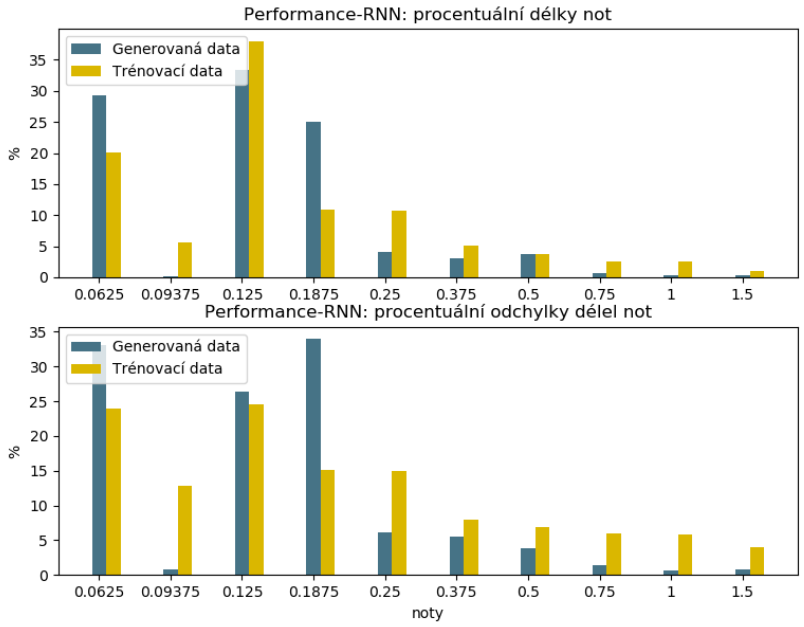
V tabulce 6.5 je možno pozorovat průměrný čas čekání před zahráním následující noty po okamžitém zahrání noty předcházející, doplněný o standardní odchylku. Tyto hodnoty jsou nic neříkající z hlediska harmonie, ale mohou být nápomocné pro pochopení generovaného rytmu. Důležitým kontrastem jsou hodnoty Variačního rekurentního autoenkodéru a Performance-RNN modelu. I přes trénování na stejném datasetu se zde průměry rozcházejí. Nízké průměry s velkou odchylkou se mohou nebezpečně přiblížit k nulovému času mezi notami. Téměř nulový čas implikuje náhodný šum not, neboť v takto vysokém rytmu se skladby nevyskytují. Vysoké průměry z vysokou odchylkou značí schopnost modelu generovat rozličně rychlé písně bez rizika náhodných not. Performance-RNN model tuto vlastnost patrně má vzhledem k maximálnímu průměru a poměrně vysoké odchylce. Ostatní modely problematiku rytmu v hudbě adresují hůře. Nízký průměr a odchylka Variační rekurentního autoenkodéru potvrzuje pozorovaný šum náhodných not, neboť malá hodnota odchylky nemůže vychýlit průměr z extrémně minimálních hodnot. Cnn-vrnn model má poměrně vysoký průměr, ovšem velká hodnota odchylky může některé písně srazit k hodnotám velmi blízko nule.



Obrázek 6.4: Histogramy délek not Variačního rekureního autoenkodéru



Obrázek 6.5: Histogramy délek not cnn-vrnn modelu



Obrázek 6.6: Histogramy délek not Performance-RNN modelu

Na obrázcích 6.4, 6.5 a 6.6 jsou zobrazeny histogramy průměrných procentuálních zastoupení délek not pro každý model a jeho trénovací dataset. Zvoleny byly všechny

symbolické délky not včetně jejich prodloužených verzí. Překlad symbolických délek na absolutní je proveden přímým dělením dané délky:

- 1/16 - 0.0625
- 1.5/16 - 0.09375
- 1/8 - 0.125
- 1.5/8 - 0.1875
- 1/4 - 0.25
- 1.5/4 - 0.375
- 1/2 - 0.5
- 1.5/2 - 0.75
- 1/1 - 1
- 1.5/1 - 1.5

Obsah histogramu nám opět může mnohé prozradit. Pokud se průměry modelů a datasetů podobají, můžeme říci, že model adresuje rytmus podobně jako dataset. Průměrné odchylky informaci následně doplňují o konzistenci tohoto chování. Přirozeností neuronové sítě je vytvářet náhodný šum, pokud není správně natrénovaná. V případě binárních hodnot výsledného piano roll to značí, že nedostatečně natrénovaná síť bude vytvářet velmi krátké noty ve velkém počtu. Pouze pokud je model kvalitní, bude používat i delší noty. Tato skutečnost je posílena malým výskytem dlouhých notových délek v datasetu. Variační rekurentní autoenkodér nedodržuje rozdělení původního datasetu. Je vidět velké množství šestnáckových not téměř bez výskytu vyšších délek. Tato skutečnost opět potvrzuje náhodný šum výstupních skladeb. Absence not vyšších délek pouze dále upevňuje skutečnost nedokonale natrénovaného modelu. Cnn-vrnn model je na tom podobně. Oproti variačnímu rekurentnímu autoenkodéru zde lze najít vyšší výskyt not větších délek. Ani v tomto případě není rozdělení délek podobné datasetu. Zvýšený výskyt delších délek může být důsledkem větších průměrů trénovacího datasetu. Extrémní odchylky několika málo délek lze chápat jako snahu o adresaci problému, ovšem ve výsledku je tato snaha příliš malá. Oproti tomu Performance-RNN je jedinou sítí o které lze říci, že se snaží adresovat rytmus hudby. Rozdělení modelu není perfektní vůči datasetu, ale je zde vidět alespoň snaha o napodobení, která by mohla být vylepšena úpravou parametrů, nebo delším trénovacím časem. Model navíc konzistentně využívá noty střední délky. Nejvyšší délky jsou opět téměř nevyužity. Další vadou je ignorování prodloužené šestnáckové délky.

	0.0625	**0.09375**	**0.125**	**0.1875**	**0.25**	**0.375**	**0.5**	**0.75**	**1**	**1.5**
0.0625	146.	13.15	12.8	8.15	5.5	3.2	1.25	0.3	0.3	0.05
0.09375	12.65	1.45	1.65	0.6	0.35	0.3	0.15	0.05	0.05	0.
0.125	12.8	1.35	1.6	0.65	0.75	0.15	0.15	0.05	0.05	0.
0.1875	7.15	1.05	0.85	0.65	0.35	0.15	0.15	0.05	0.	0.
0.25	4.65	0.3	0.55	0.7	0.15	0.1	0.1	0.	0.	0.
0.375	2.5	0.15	0.4	0.15	0.15	0.15	0.	0.	0.	0.
0.5	1.25	0.15	0.1	0.05	0.15	0.05	0.1	0.	0.	0.
0.75	0.25	0.05	0.15	0.	0.	0.	0.	0.	0.	0.
1	0.25	0.	0.05	0.	0.	0.	0.	0.	0.	0.
1.5	0.05	0.	0.	0.	0.	0.	0.	0.	0.	0.

Tabulka 6.6: Variační rekurentní autoenkodér: průměrná procentuální matice přechodu délek not

	0.0625	**0.09375**	**0.125**	**0.1875**	**0.25**	**0.375**	**0.5**	**0.75**	**1**	**1.5**
0.0625	208.65	35.25	24.7	12.55	8.1	7.3	7.	2.	1.75	1.65
0.09375	35.15	6.75	5.35	3.65	1.75	2.8	2.15	0.6	0.3	0.7
0.125	22.8	4.8	8.75	2.75	2.45	1.9	2.1	0.2	0.2	0.25
0.1875	13.	3.25	2.65	2.35	1.6	2.65	1.3	0.2	0.35	0.5
0.25	9.25	2.4	2.35	0.9	2.15	1.75	1.35	0.1	0.35	0.2
0.375	8.4	2.3	2.15	1.6	1.4	2.4	1.3	0.25	0.15	0.25
0.5	5.9	1.85	0.9	2.9	0.95	1.75	16.	1.35	0.3	1.
0.75	1.85	0.5	0.25	0.45	0.3	0.1	1.55	1.45	0.25	0.55
1	1.55	0.65	0.25	0.05	0.15	0.6	0.15	0.15	0.35	0.55
1.5	1.4	0.5	0.25	0.35	0.3	0.4	0.65	0.75	0.35	1.95

Tabulka 6.7: Cnn-vrnn: průměrná procentuální matice přechodu délek not

	0.0625	**0.09375**	**0.125**	**0.1875**	**0.25**	**0.375**	**0.5**	**0.75**	**1**	**1.5**
0.0625	280.	0.05	33.05	2.3	0.65	0.4	0.9	0.1	0.05	0.05
0.09375	0.2	0.05	0.2	0.1	0.	0.	0.05	0.	0.	0.
0.125	35.2	0.3	100.9	2.65	2.2	1.1	2.05	0.15	0.1	0.1
0.1875	0.35	0.05	4.1	26.35	0.75	0.55	0.95	0.	0.	0.1
0.25	0.05	0.05	2.95	0.75	5.4	0.55	0.15	0.25	0.	0.
0.375	0.	0.	1.15	0.5	0.5	2.45	0.45	0.	0.	0.
0.5	0.4	0.1	1.9	0.5	0.3	0.15	2.7	0.15	0.1	0.15
0.75	0.	0.	0.2	0.1	0.25	0.05	0.05	0.4	0.	0.
1	0.	0.	0.25	0.1	0.	0.05	0.	0.	0.3	0.
1.5	0.	0.	0.15	0.	0.1	0.	0.05	0.05	0.05	0.35

Tabulka 6.8: Performance-RNN: průměrná procentuální matice přechodu délek not

Tabulky 6.6, 6.7 a 6.8 představují průměrné procentuální matice přechodu délek tónů pro všechny modely. Z těchto hodnot lze získat zajímavé informace nejen o použitých délkách, ale i o vztazích a závislostech mezi nimi, které si model během trénování vytvořil. Na rozdíl od tabulek 6.2, 6.3, 6.4 zde nejsou zvýrazněny významné hodnoty z důvodu velké řídkosti matice. Všechny sledují stejný trend vysokého

výskytu a opakování nejkratší možné noty šestnáckové. To lze pozorovat v prvním čísle diagonály. Kolem této hodnoty je rozdělena většina použitých přechodů. V případě Variačního rekurentního autoenkodéru jsou veškeré relevantní informace právě v této oblasti. Šestnáctková délka je nejpravděpodobnější následovník jakékoli jiné délky. To implikuje absenci adresace problému rtmu ve výstupní hudbě. Cnn-vrnn model v porovnání s Variačním rekurentním autoenkodérem používá přechody na vyšší délky not. V případě např. prodloužené noty celé je nejpravděpodobnějším následovníkem nota stejné délky a nikoli nota šestnáctková. To vypovídá alespoň o minimální snaze chápat rytmus jako konzistenci podobně dlouhých not. Ovšem v případě modelu cnn-vrnn je toto chování konzistentní pouze v extrémně dlouhých notách, kdy je zachování stejných délek nejvíce evidentní. Performance-rnn model je jediný se signifikantními výsledky u dlouze hraných not. Hraní obecně dlouhého tónu implikuje velkou délku tónu následujícího. Pokud je vygenerovaná skladba v nízkém tempu, je zde mnohem menší pravděpodobnost velkého zrychlení v malém čase. Většina hodnot je ovšem stále koncentrována u tónů nejkratších. Z toho vyplývá, že většina skladeb má vlastnost rychlého rytmu, ovšem model je schopen vytvořit také píseň pomalého rázu.

Jak již bylo zmíněno, kvantitativní měření natolik subjektivního obsahu, jako je hudba, může být interpretováno mnoha způsoby. I přes to se nám na několika jednoduchých metrikách podařilo popsat některé vlastnosti modelů. Ve výsledku Variační rekurentní autoenkodér má podobně jako cnn-vrnn snahu adresovat některá důležitá hudební pravidla, ovšem oba modely selhávají. Cnn-vrnn model je ke správnému výsledku mnohem blíž. Performance-RNN figuruje v měření velmi kontrastně. Díky porovnání s tímto modelem lze získat alespoň přibližnou představu vzdálenosti ke kvalitnímu výsledku.

7 Závěr

Záměrem práce je prozkoumání aktuálních metod a prostředků sloužících ke generování hudebního obsahu. Získání a vhodné předzpracování datasetu pro co nejefektivnější trénování neuronové sítě. Implementace zvolené architektury neuronové sítě a její natrénování na předzpracovaném datasetu. Závěrečnou částí je porovnání implementované sítě s volně dostupnými modely řešící stejnou úlohu.

Obsahem průzkumu aktuálních metod je stručná historie problematiky generování hudby, popis tří úspěšných genrátorů hudebního obsahu, metody kódování hudby do strojově zpracovatelného formátu a popis architektur neuronových sítí, sloužících pro generativní účely. Průzkum probíhal na základě originálních prací zkoumající danou problematiku a volně dostupných výtahů historických i současných řešení. Veškeré zdrojové texty jsou řádně citovány v seznamu literatury.

Vybraným datasetem je 718 midi souborů obsahující arkádové video herní soundtracky. Dataset byl získán z webové stránky vgmusic.com. Autoři stránek uvádějí volnou dostupnost nashromážděných dat. Pro získání hudebně relevantních vlastností datasetu, byla implementována analýza použitých tónů, not, stupnic, taktů a hudebních nástrojů. Na základě výsledků analýzy a originálních prací uvádějící techniky předzpracování hudby byly implementovány algoritmy předzpracování dat. Použité techniky jsou transpozice do C-dur stupnice a omezení tónového rozsahu.

Implementovanou architekturou je Variační rekurentní autoenkodér. Rekurence je implementována několikavrstvými sítěmi typu LSTM. Trénování probíhalo na Google colab cloudovém prostředí. Jako vstup je použit předzpracovaný dataset. Optimalizačním algoritmem je Adam mini-batch gradient descent, variační rekurentní autoenkodér byl rozšířen o metodu Beta-vae pro vyvážení chybových funkcí. Výsledné trénování dosáhlo lokálního minima optimalizační funkce. Vygenerované výstupy jsou uloženy ve formátu midi. Vygenerovaná hudba obsahuje silný šum náhodných not a není příliš strukturovaná. Přesto jsou pozorovatelné krátkodobé melodie a akordy.

Konečnou částí práce je porovnání implementovaného modelu s volně dostupnými projekty generující hudbu. Zvolenými projekty jsou Performance-RNN-PyTorch a cnn-vrnn-polyphonic-music-generation. Obě sítě jsou natrénovány na podmnožině mého datasetu. Velikost podmnožiny je dána velikostí originálního trénovacího datasetu každého modelu. Porovnání je kvantitativní. Mezi použité metriky patří počet unikátních tónů, histogram výskytu not, matice přechodu not, průměrný čas mezi tóny, histogram délek not a matice přechodu délek not. Performance-RNN-PyTorch výrazně převyšuje svou kvalitou ostatní použité projekty. Cnn-vrnn-polyphonic-music-generation je mé síti mnohem blíže, ovšem díky silnějším vazbám k pravidlům

hudby a mnohem menšímu šumu náhodných tónů je vyhodnocen jako kvalitnější.

Práci lze v budoucnu rozšířit vylepšením nebo nahrazením použité architektury pro kvalitnější hudební výstup. Možným rozšířením může být i zahrnutí většího počtu hudebních nástrojů. Lepších výsledků lze pravděpodobně docílit použitím konvolučních sítí, attention rekurentních sítí, nebo pravidel zpětnovazebního učení. Možné je i nahradit samotný variační rekurentní autoenkodér sítí typu GAN, nebo Transformer.

Literatura

[1] Key-finding algorithm. Accessed: 2020-05-11.

[2] Magenta. Accessed: 2020-05-11.

[3] Vgmusic - 31,810 game music midi files. Accessed: 2020-05-11.

[4] Maxime Allard. What is a transformer?, 6 2019.

[5] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.

[6] Michael Scott Cuthbert and Christopher Ariza. Music21: A toolkit for computer-aided musicology and symbolic music data. In J. Stephen Downie and Remco C. Veltkamp, editors, *Proceedings of the 11th International Society for Music Information Retrieval Conference, ISMIR 2010, Utrecht, Netherlands, August 9-13, 2010*, pages 637–642. International Society for Music Information Retrieval, 2010.

[7] S. Dieleman and B. Schrauwen. End-to-end learning for music audio. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6964–6968, 2014.

[8] Hao-Wen Dong and Yi-Hsuan Yang. Convolutional generative adversarial networks with binary neurons for polyphonic music generation, 4 2018.

[9] D. Eck and J. Schmidhuber. Finding temporal structure in music: blues improvisation with lstm recurrent networks. In *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*, pages 747–756, 2002.

[10] Otto Fabius and Joost R. van Amersfoort. Variational recurrent auto-encoders, 2014.

[11] Daniil Gavrilov. pytorch_rvae. https://github.com/kefirski/pytorch_RVAE, 2017.

[12] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Ben;o. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[13] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[14] Jonathan Hui. Gan — wasserstein gan & wgan-gp, 6 2018.

[15] Jean-Pierre Briot. Example-of-symbolic-piano-roll, 2017. [Online; accessed March 27, 2020].

[16] Jeremy Jordan. Introduction to autoencoders., 2018.

[17] Honglak Lee, Peter Pham, Yan Largman, and Andrew Y. Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1096–1104. Curran Associates, Inc., 2009.

[18] Yuankui Lee. Performance-rnn-pytorch. https://github.com/djosix/Performance-RNN-PyTorch, 2018.

[19] Lewis. Creation by refinement: a creativity paradigm for gradient descent learning networks. In *IEEE 1988 International Conference on Neural Networks*, pages 229–233 vol.2, 1988.

[20] Matija Marolt, Alenka Kavcic, and Marko Privosnik. Neural networks for note onset detection in piano music. 2002.

[21] Arunesh Mittal. Variationalrecurrentautoencoder. https://github.com/arunesh-mittal/VariationalRecurrentAutoEncoder, 2017.

[22] Andrew Ng. Cs294a lecture notes - sparse autoencoder, 2011.

[23] Kelly Nick. Transpose midi with python for computational creativity in the domain of music.

[24] Christopher Olah. Understanding lstm networks, 2015.

[25] Jean-Pierre Briot Gaëtan Hadjeres François-David Pachet. Deep learning techniques for music generation – a survey, 2017.

[26] Jordi Pons. Neural networks for music: A journey through its history, 10 2018.

[27] Colin Raffel and Daniel P. W. Ellis. Intuitive analysis, creation and manipulation of midi data with pretty_midi.

[28] Adam Roberts, Jesse Engel, and Douglas Eck, editors. *Hierarchical Variational Autoencoders for Music*, 2017.

[29] Joseph Rocca. Understanding variational autoencoders (vaes), 2019.

[30] P. Todd. A sequential network design for musical applications. 05 2020.

[31] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio, 2016.

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[33] Chi-Feng Wang. The vanishing gradient problem, 2019.

[34] Dustin Wright. cnn-vrnn-polyphonic-music-generation. https://github.com/dwright37/cnn-vrnn-polyphonic-music-generation, 2018.

[35] Li-Chia Yang and Alexander Lerch. On the evaluation of generative models in music. *Neural Computing and Applications*, 11 2018.

[36] Alex Yu. Generating music with artificial intelligence, 3 2019.

Příloha A - obsah přiloženého DVD

- text bakalářské práce: bakalarska_prace_2020_David_Cerny.pdf
- zdrojový kód: network/
- dataset: network/midi_data/game/
- vygenerované výstupy: generated/