ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Štíbal** Jméno: **Pavel** Osobní číslo: **412008**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Syntéza zvukových signálů dle příkladů pomocí neuronových sítí

Název diplomové práce anglicky:

Example-driven Audio Signal Synthesis using Neural Networks

Pokyny pro vypracování:

Seznamte se s existujícími architekturami neuronových sítí, které je možné využít pro reprezentaci a reprodukci zvukového signálu dle příkladu, zejména s architekturou WaveNet. Zaměřte se na vybranou množinu typů zvuků, běžně používaných v moderní taneční hudbě (různé typy bicích, různé typy melodických nástrojů, různé typy harmonických výplní, apod.) Prozkoumejte možnosti parametrizace výstupu, zejména jak může uživatel deterministicky ovlivňovat výsledek tohoto procesu. Prozkoumejte, jak je možné využít WaveNet pro ?interpolaci? dvou instancí stejného typu zvuku. Zhodnoťte funkci pro různé typy zvuků pomocí poslechových testů s uživateli. Zhodnoťte možnosti a omezení tohoto přístupu ke generování zvuku

Seznam doporučené literatury:

Oord, A. V. D., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., ... & Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. arXiv preprint arXiv:1609.03499.
OLAH, Christopher. Understanding LSTM Networks. 2015. URL http://colah. github. io/posts/2015-08-Understanding-LSTMs/img/LSTM3-chain. png, 2015.
KARPATHY, A. The unreasonable effectiveness of recurrent neural networks, 2015. Dispon?vel em http://karpathy. github. io/2015/05/21/rnn-effectiveness, 2016.
NIELSEN, Michael. Neural Networks and Deep Learning.(2016). Saatavilla http://neuralnetworksanddeeplearning. com/index. html, viitattu, 2016, 18.

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Adam Sporka, Ph.D., Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **02.02.2018** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **30.09.2019**

doc. Ing. Adam Sporka, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

© ČVUT v Praze, Design: ČVUT v Praze, VIC

III. PŘEVZETÍ ZADÁNÍ

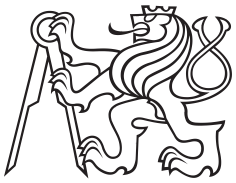
Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Diplomová práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Syntéza zvukových signálů dle příkladů pomocí neuronových sítí

Bc. Pavel Štíbal

Vedoucí: doc. Ing. Adam Sporka, Ph.D.
Obor: Otevřená Informatika
Studijní program: Softwarové inženýrství
Leden 2019

Poděkování

Rád bych poděkoval všem, kteří mě při psaní této práce podporovali. Především bych rád poděkoval doc. Ing. Adamovi Sporkovi, Ph.D., vedoucímu diplomové práce, za odborné vedení a za pomoc při zpracování všech materiálů. Také bych rád poděkoval Ing. Romanovi Janovskému za konzultaci neuronových sítí. Děkuji Mgr. Janě Sukové a Bc. Johaně Emmě Křečkové za jazykovou korekci. Dále chci poděkovat účastníkům závěrečného testování.

Prague, 3. ledna 2019

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškerou použitou literaturu.

V Praze, 3. ledna 2019

Abstrakt

Diplomová práce se věnuje syntéze zvukových signálů dle příkladů za pomoci neuronových sítí.

Při řešení této problematiky byla provedena rešerše neuronových sítí a existujících architektur, které je možné využít pro reprezentaci a reprodukci zvukového signálu.

V rámci moderní taneční elektronické hudby byla vybrána vhodná množina běžně užívaných zvukových typů pro zkoumání parametrizace výstupu.

Cílem je zejména možnost uživatele deterministicky ovlivňovat výsledek procesu syntézy dvou instancí stejného typu zvuku s následným zkoumáním možností při využití softwarové architektury WaveNet, kde výsledky celé práce jsou hodnoceny pomocí poslechových testů uživateli.

Přínosem této práce je zejména vytvořená dokumentace výše zmíněné problematiky, její zhodnocení a analýza výsledků poslechových testů.

Klíčová slova: WaveNet, neuronové sítě, zvuková syntéza, syntezátor

Abstract

The diploma thesis covers the synthesis of audio signals according to examples using neural networks.

The research of neural network and existing architectures was performed for the audio signal reproduction and representation.

Within the modern dance electronic music, a suitable set of commonly used audio types was chosen to examine the output parameterization.

The goal is user deterministic influenced output of two same sound type instances used in the synthesis process, and further possibilities exploration using WaveNet software architecture, where the results are evaluated by audio user testing.

The contribution of this thesis is especially created documentation of the explored aspects, its evaluation and audio tests results analysis.

Keywords: WaveNet, neural networks, synthesis, synthesizer

Obsah

1 Úvod **1**

1.1 Cíl práce 1

2 Rešerše typických zvuků v taneční hudbě **3**

2.1 Syntéza zvuku 6

2.2 Principy tvorby zvuku 7

3 Rešerše neuronových sítí **13**

3.1 Umělá inteligence 13

3.1.1 Inteligence 13

3.1.2 Umělá inteligence 14

3.1.3 Rozdíly inteligence člověka a stroje 15

3.1.4 Techniky 15

3.2 Rešerše neuronových sítí 17

3.2.1 Stavební jednotky neuronových sítí 18

3.2.2 Učení neuronové sítě 20

3.2.3 Využití neuronových sítí ve zvuku 22

3.2.4 Typy neuronových sítí 22

3.3 Architektury neuronových sítí . . 27

3.3.1 WaveNet 27

3.3.2 Prostředky pro realizaci a experimenty 30

4 Příprava k experimentům **37**

4.1 Implementace MNIST autoencoderu 37

4.2 Vytvoření datasetů pro tensorflow-wavenet 39

5 Experimenty a uživatelské testování **41**

5.1 Participanti 41

5.2 Pre-test 42

5.3 Post-test 42

5.4 Experimenty se syntézou různých typu zvuku 42

5.4.1 Uživatelské testovaní experimentů 43

5.4.2 Experiment 1 44

5.4.3 Experiment 2 45

5.4.4 Experiment 3 45

5.4.5 Experiment 4 46

5.4.6 Experiment 5 47

5.4.7 Experiment 6 49

5.4.8 Vizualizace vygenerovaných experimentů 49

5.4.9 Výsledky experimentů 51

5.4.10 Závěr experimentu 56

5.5 Experiment se syntézou mezi dvěma instancemi stejného typu . . 57

5.5.1 Uživatelské testovaní experimentu 57

5.5.2 Experiment 7 59

5.6 Zhodnocení Experimentů 64

5.6.1 Zhodnocení celého testovaní . 64

6 Závěr **67**

A Literatura **69**

B Seznam zkratek **75**

C Obsah přiloženého CD **77**

D Lineární prediktivní kódování enkodér-dekodér **79**

Obrázky

2.1 Bass drum (převzato z [1]) 4

2.2 Stupnici C dur od velkého C do tříčárkovaného C (převzato z [2]) . . 4

2.3 Snare drum (převzato z [1]) 4

2.4 Hi-hat (převzato z [1]) 5

2.5 Syntezátor (převzato z [1]) 7

2.6 Sampler (převzato z [1]) 8

2.7 Blokové schéma bass drumu (převzato z [3]) 9

2.8 Amplituda v čase pro zvuk bass drum 9

2.9 Blokové schéma snare drumu (převzato z [3]) 10

2.10 Amplituda v čase pro zvuk snare drum 10

2.11 Blokové schéma hi-hatu (převzato z [3]) 11

2.12 Amplituda v čase pro zvuk hihat 11

3.1 Nedokreslený čtverec 15

3.2 Rozdělení AI (převzato z [4]) . . . 16

3.3 Perceptron 18

3.4 Jednoduchá síť (převzato z [5]) . 20

3.5 Neuronová síť s vrstvy (převzato z [5]) 20

3.6 Možnosti výsledků učení neuronových sítí (převzato z [6]) . . 21

3.7 FFNN (převzato z [7]) 23

3.8 2D konvoluce (převzato z [8]) . . . 24

3.9 Ukázka RNN (převzato z [9]) . . 25

3.10 Celá architektura (převzato z [10]) 28

3.11 Princip WaveNetu (převzato z [10]) 28

3.12 Princip rozšířeného WaveNetu (převzato z [11]) 29

3.13 Autoencoder (převzato z [12]) . 31

3.14 Princip WaveNetu jako autoenkodér (NSynth) 35

5.1 Zjednodušené znázornění učení WaveNetu 49

5.2 Amplituda v čase pro vygenerovaný zvuk z experimentu 1 50

5.3 Amplituda v čase pro vygenerovaný zvuk z experimentu 2 50

5.4 Amplituda v čase pro vygenerovaný zvuk z experimentu 3 50

5.5 Amplituda v čase pro vygenerovaný zvuk z experimentu 4 50

5.6 Amplituda v čase pro vygenerovaný zvuk z experimentu 5 50

5.7 Amplituda v čase pro vygenerovaný zvuk z experimentu 6 50

5.8 Amplituda v čase pro dva původní zvuky a vygenerované zvuky z experimentu 7 61

5.9 Porovnání poslechových testů experimentů 1 až 6 65

D.1 Blokové schéma LPC enkodéru-dekodéru (převzato z [13]) 80

Tabulky

5.1 Tabulka popisující výsledky experimentu 1 52

5.2 Tabulka popisující výsledky experimentu 2 pro dataset bass drums 52

5.3 Tabulka popisující výsledky experimentu 2 dataset snare drums 53

5.4 Tabulka popisující výsledky experimentu 2 dataset hihats 53

5.5 Tabulka popisující výsledky 1. části experimentu 3 53

5.6 Tabulka popisující výsledky 2. části experimentu 3 54

5.7 Tabulka popisující výsledky 1. části experimentu 4 54

5.8 Tabulka popisující výsledky 2. části experimentu 4 54

5.9 Tabulka popisující výsledky 1.části experimentu 5 55

5.10 Tabulka popisující výsledky 2. části experimentu 5 55

5.11 Tabulka popisující výsledky 1. části experimentu 6 55

5.12 Tabulka popisující výsledky 2. části experimentu 6 56

5.13 Tabulka popisující výsledky 1. části experimentu 7 62

5.14 Tabulka popisující výsledky 2. části experimentu 7 63

5.15 Tabulka popisující výsledky 3. části experimentu 7 63

Kapitola 1

Úvod

Problematika syntézy zvukových signálů podle předloh pomocí neuronových sítí navzdory jejich rozmachu není natolik zmapovaná. Neuronové sítě se pro tuto problematiku téměř nepoužívají, proto je tato oblast skvělou příležitostí k výzkumu. Tyto sítě nejčastěji pracují s obrázky, ale pokud se budeme hovořit o nejčastějším využití neuronových sítí v hudbě, tak jednoznačně jedná o generování hudební skladby, například se může jednat o určitý předem daný žánr, nebo o napodobení některého skladatele.

Tato diplomová práce se bude zabývat syntézou zvuku, a to jak s použitím neuronových sítí, tak i bez jejich použití. Danou problematiku bych tedy chtěl prozkoumat od úplného základu, jako je například popis syntézy zvuků, až po využití neuronových sítí pro syntézu zvuků podle předloh.

Neuronové sítě jsou obecně velmi zajímavé téma, protože jejich využití je poměrně rozsáhlé. Jak jsem již psal výše, i přes jejich rozsáhlé využití se téměř vůbec nepoužívají pro syntézy zvukových signálů podle předloh. Vzhledem k potenciálu takového upotřebení a použití jsem tedy k vyřešení problému velmi vysoce motivován. Pokud jsou neuronové sítě používané pouze tímto směrem, tak jsou bohužel využité jen pro napodobení předlohy, což je podle mě sice velmi zajímavé, ale stále mě to vede k otázce, kterou bych chtěl pomocí této diplomové práce zodpovědět. Otázka zní: Je možné parametricky ovlivnit reprodukci zvuku pomocí neuronových sítí a dvou předloh stejného typu?

1.1 Cíl práce

Záměrem mé diplomové práce je splnění několika úkolů, které uvedu níže:

- **1. cíl - architektury neuronových sítí**: Seznámení se s existujícími architekturami neuronových sítí, které je možné využít pro reprezentaci a reprodukci zvukového signálu podle předloh. Zejména je potřeba se zaměřit na architekturu WaveNet. K tomuto cíli bude potřeba udělat rešerši

architektur neuronových sítí, které používají alespoň zčásti architekturu WaveNet (viz sekce 3.3 a 5)

- **2. cíl - typy zvuků**: Zaměření na vybranou množinu typu zvuků, které jsou běžně používané v moderní taneční hudbě. Tedy bude nutné udělat rešerši typu zvuků, které jsou používané v taneční hudbě (viz sekce 2). Také je zapotřebí se zaměřit na určitou množinu zvuků, se kterou se bude pracovat v praktické části (viz sekce 4.2 a 5).
- **3. cíl - deterministické ovládání**: Prozkoumání možnosti parametrizace výstupu, zaměřeno zejména na možnost uživatele deterministicky ovlivňovat výsledek tohoto procesu. Tento cíl se nachází v kapitole 5.
- **4. cíl - využiti WaveNetu pro syntézu zvuku**: Zjištění možností využití WaveNetu pro "interpolaci"dvou instancí stejného typu zvuku. "Interpolací"je myšlená jako syntéza dvou a více zvuků pomocí architektury WaveNet. Důležité sekce pro tento cíl jsou 2.1, 4 a 5.
- **5. cíl - poslechové testy**: Zhodnocení funkcí pro různé typy zvuku pomocí poslechových testů s uživateli. Tento cíl se nachází v kapitole 5.
- **6. cíl - vygenerovaní zvuku**: Zhodnocení možností a omezení přístupu, který je založený na generování zvuku podle předlohy. Daný cíl je uveden v sekcích 3.2, 3.3 a 5

Kapitola 2

Rešerše typických zvuků v taneční hudbě

V této kapitole je uvedená rešerše tzn.: od popsaných typických zvuků, které se nejčastěji používají v taneční hudbě, až po přiblížení syntézy těchto zvuků.

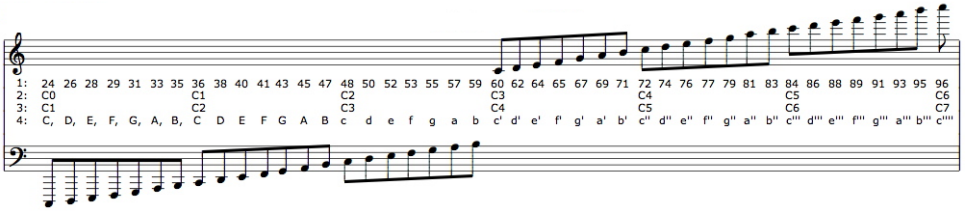
Než se dostaneme k typům zvuků, bylo by dobré zmínit, co se skrývá pod pojmem elektronická hudba. Elektronická hudba je taková, která je vytvořena pomocí hudebních přístrojů, či hudebních softwarů. Pod těmito termíny si můžete konkrétněji představit syntezátory a samplery. Za zmínku určitě stojí i některé hudební žánry (styly), které patří do elektronické hudby, jako např.: Dubstep, House, Drum'n'Bass, atd. [14]

Typické elektronické zvuky se snaží napodobit jeden nebo více hudebních nástrojů a také se snaží vyjít ze zvuku hudebních nástrojů. Tím myslím, že základní zvuk zněl jako buben, ale výsledný zvuk je velmi modifikovaný na to, abychom mohli říci, že se jednalo o buben. Níže uvedu typické zvuky a též popíšu, který hudební nástroj má daný zvuk reprezentovat. Jedná se o tyto zvuky: [3]

- **Bass drum**: také se označuje jako Bass drum. Tento zvuk napodobuje reálný nástroj (viz obrázek 2.1), jenž nese totožné označení jako tento zvuk.

Obrázek 2.1: Bass drum (převzato z [1])

Tento zvuk je obecně velmi hluboký a měl by ležet ve velké oktávě, konkrétněji někde mezí C a G (viz obrázek 2.2), ale sluchově vnímaný je ještě nižší. Typicky má frekvenci mezi 80Hz a 150Hz. Daný zvuk trvá obvykle 3 - 4 sekundy. [3]



Obrázek 2.2: Stupnici C dur od velkého C do tříčárkovaného C (převzato z [2])

- **Snare drum**: tento zvuk má napodobovat hudební nástroj, který má v angličtině stejný název, ale v překladu je označován jako bubínek (viz obrázek 2.3).

Obrázek 2.3: Snare drum (převzato z [1])

Jedná se o zvuk, který je kratší než bass drum, obvykle trvá 0,5 až 2,5 sekundy, ovšem v závislosti na napětí strun. To, co je pak slyšet, je však pouze vibrace strun, která má obvykle podobu jediného tónu. Také tento zvuk jako bass drum leží ve velké oktávě, ale pohybuje se v rozmezí mezi C a D (viz obrázek 2.2). Obvykle má frekvenci mezí 120Hz a 250Hz. [3]

- **Hihat**: také napodobuje reálný hudební nástroj, jenž patří jako snare drum a bass drum do bicí soupravy (do picích hudebních nástrojů). Jedná se o činel nebo téže Hi-hat (viz obrázek 2.4)

Obrázek 2.4: Hi-hat (převzato z [1])

Typické hihaty mají obvykle frekvenci mezi 300Hz a 3000Hz. Jedná se o jeden z nejvyšších zvuků, který je možné v bicí soupravě nalézt. Je stejně trvající jako snare drum. [3]

- **Bass**: nejprve bych se měl zmínit, že pod pojmem bass se vyskytuje mnoho hudebních nástrojů a tónů, proto napíšu jejich společnou charakteristiku. Hlavní znakem tohoto zvuku je, že máš nízkou frekvenci (rozsah je 16-256 Hz, neboli leží na stupnici mezi subkontra C (C0) a e) a výšku. [3]

- **Lead**: tento zvuk může reprezentovat také jako bass mnoho hudebních nástrojů, proto zde uvedu také pouze společné charakteristiky, které platí pro tento zvuk. Jak název napovídá, jedná se o zvuk, který je velmi výrazný. Bohužel nejde jednoznačně říci, jaký rozsah frekvence by měl daný zvuk mít, proto uvádím spíše příklady hudebních nástrojů, a to: Saxofon, housle, atd. [3]

- **String / pad**: podobně jako dva předešlé zvuky (bass a lead) může tento napodobovat více nástrojů. Uvedu také společnou charakteristiku. Tento zvuk může ležet na stupnici v rozmezí od G do čtyřčárkovaného a (A7), což odpovídá frekvenci 95-3150Hz. Zjednodušeně se může hovořit o tom, že tento zvuk se snaží napodobit nějaký strunný nástroj. [3]

2.1 Syntéza zvuku

V této podkapitole popíšu, co je to syntéza zvuku a k čemu slouží.

Syntéza zvuku je technika, která slouží ke generování nebo vytváření zvuku. Tedy jedná se o libovolný proces vedoucí k vytvoření předvídatelného nebo i nepředvídatelného zvuku. Z toho vyplývá, že existuje nepřeberné množství způsobů, jak vytvořit zvuk, či generovat signál, který se dá interpretovat jako zvuk. Rád bych podotkl, že se k syntéze zvuku využívá syntezátor, který může být hardwarový nebo softwarový

Syntéza zvuku se využívá nejčastěji ke dvěma účelům:

1. k vytváření zvuků, které se snaží co nejvíce přiblížit k již existujícímu hudebnímu nástroji. Jinými slovy: tato metoda se snaží namodelovat co nejpřesněji konkrétní hudební nástroj.
2. k tvorbě nových, zatím neexistujících a dosud neslyšených zvuků.

O něco těžší je první varianta, protože v tomto případě máme možnost srovnávat daný výsledek s konkrétním hudebním nástrojem a tak můžeme objektivně hodnotit, zda je zvuk správně vygenerovaný, či nikoliv. [15]

Zajímavé je, že v dnešní době máme celou řadu metod elektronické syntézy, ale jednotlivé syntézy nejsou vzájemně zcela zastupitelné a nepokrývají úplný prostor všech možných zvuků. Tzn.že jednou konkrétní metodou se nemusím nutně dobrat kýženého výsledku. Předem je nutné říci, že existuje více kritérií a rovin, díky kterým se jednotlivé metody zvukové syntézy dají členit. Pro zjednodušení jsou metody syntézy následovně rozčleněny takto:[16]

- **Aditivní syntéza:** také se jí říká součtová syntéza. Tato metoda vychází ze součtu signálů v časové i frekvenční oblasti. Jestliže jsou funkce $F(t)$ a $G(t)$, které reprezentují dva zvukové signály, tak $F(t) + G(t)$ je novým signálem kvantitativně i kvalitativně odlišným. Jinými slovy: je založena na prostém součtu jednoduchých signálů.
- **Subtraktivní syntéza:** také se nazývá Rozdílová syntéza. Tato metoda vychází z rozdílu frekvenčních charakteristik komplexního signálu a filtru. Jedná se v podstatě o analytický postup, ke kterému se používá filtrace, zdůrazňování nebo potlačování frekvenčních složek daného signálu
- **Modulační syntéza:** tato metoda vychází z ovlivňování daného parametru zvukového signálu jiným signálem. Máme více typů modulací signálu - amplitudová, frekvenční a pulzně šířková. Např.: Pokud máme dvě funkce $F(t)$ a $G(t)$ výsledný signál bude odpovídat funkci $F[G(t)]$.
- **Tvarová syntéza:** tato metoda vychází buď z přímého působení na signál v jeho časové oblasti, nebo z realizace požadovaného tvaru jeho časového průběhu.

- **Slučovací syntéza:** tato metoda vychází ze skládání malých zvukových jednotek dlouhých pár milisekund do větších, zvukově zajímavých celků. Máme více typů slučovacích syntéz, které se nazývají segmentační, granulační a formantová.

2.2 Principy tvorby zvuku

V elektronické hudbě se používají tři hlavní nástroje (principy) pro tvorbu elektronických zvuků, mezi tyto nástroje patří:

- **Syntezátor**: princip neboli syntézu jsem popsal v kapitole 2.1, proto pouze přiblížím pouze tento hudební nástroj, který muže být softwarový nebo hardwarový jako sampler. Slouží pro vytváření téměř jakýchkoli druhů zvuku za pomocí filtrů a oscilátorů, tedy je schopný napodobit reálné nástroje, nebo vytvořit zcela nové zvuky. [3]

Obrázek 2.5: Syntezátor (převzato z [1])

- **Vokodér**: je hudební nástroj, který muže být softwarový nebo hardwarový. Analyzuje a syntetizuje lidský hlas, který se například používá pro hlasovou transformaci. Základem vokodéru jsou dva hlavní bloky: enkodér a dekodér. V enkodéru se nachází vícepásmový filtr, který rozdělí vstupní signál do mnoha frekvenčních pásmem. Následně každý pás prochází syntetickou úpravou jako např.: přes obálku. Dekodér rozkóduje přijímaný signál a dále pomocí syntézy zvuku přiblíží signál co nejvíce originálu. [17]
- **Sampler**: než budu popisovat sampler, měl bych alespoň zjednodušeně přiblížit princip, který se jmenuje sampling nebo vzorkování, které se používá pro práci se zvukovými vzorky, tzv. samply. Jedná se o převzetí zvuku nebo části zvukové nahrávky, a tu znovu použít pro vytvoření nové skladby.

Sampler je hudební nástroj, který může být hardwarový nebo softwarový. Může se použít i se syntezátorem. Slouží k vytváření nových zvuků za pomocí zvukových záznamů, které mohou být např.: zvuky reálných nástrojů, kusy z nahraných skladeb, nebo jiné zvuky (např.: klaksonů, sirény, oceánské vlny, atd.). Tyto zvuky jsou pak přehrávány pomocí sampleru a za pomoci nějakého spouštěcího zařízení např.: elektronických bicí apod. vytváří hudbu. Často obsahují i efekty, filtry a modulace. [1]

Obrázek 2.6: Sampler (převzato z [1])

Jelikož prozkoumání všech elektronických zvuků by bylo velmi náročné, proto jsem se zaměřil v rámci této diplomové práce na konkrétní tři zvuky, u kterých uvedu jejich syntézu (neboli jak se tvoří pomocí syntezátoru). Zaměřil jsem se na bubny a konkrétně jsou to:

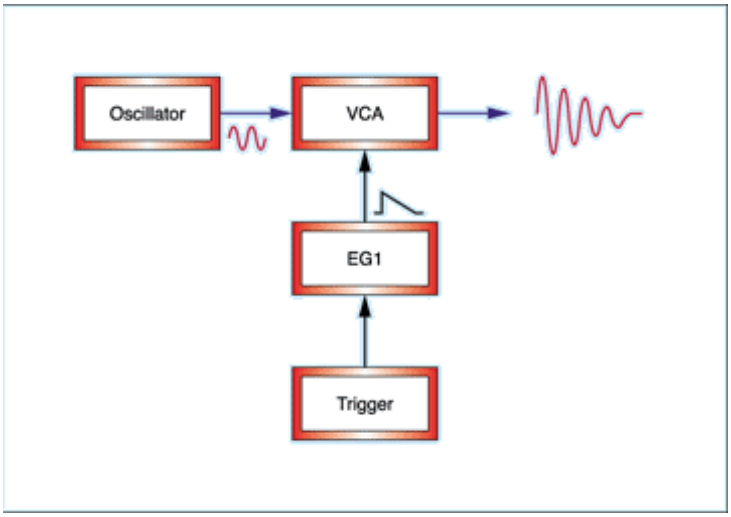
- **Bass drum**: pomocí jednoduchého blokového schématu (viz obrázek 2.7) popíšu syntézu tohoto zvuku. Na blokovém schématu jsou 4 důležité bloky: oscilátor, VCA, EG1 a trigger. [3]

  Oscilátor generuje různý typ signálů, pro tento zvuk se může konkrétně použít čtvercový, trojúhelníkový, ale nejčastěji sinusový signál. Tedy pomocí oscilátoru vygenerujeme nějaký signál, který se pomocí dalších částí snažíme upravit tak, aby zněl jako požadovaný. [3]

  VCA se používá v hardwarové konstrukci. Jedná se napěťově řízený zesilovač, který jde ve softwarové konstrukci upravit jakýmkoliv zesilovačem. Tedy tento blok zesiluje signál na požadovanou hodnotu. [3]

  Je patrné, že z výše popsaných bloků bychom nemohli vytvořit tento požadovaný zvuk, proto je potřeba přiblížit i blok, který nese název EG1, obálka. Tento blok dá signálu požadovaný amplitudový tvar. [3]

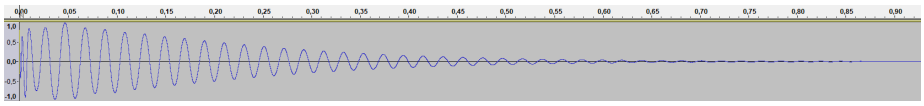
  Trigger slouží pouze k spuštění obálky, a tím vytvoření požadovaného momentu chtěného zvuku.



Obrázek 2.7: Blokové schéma bass drumu (převzato z [3])

Zvuk (viz obrázek 2.8) potřebuje ke svému vytvoření určité parametry. Obecně jsou to tyto:

- **Sample rate** (vzorkovací frekvence): definuje počet vzorků za jednotku času (obvykle za 1 sekundu)
- **Typ signálů**: označuje, jaký tvar signálů bude generovat oscilátor
- **Obálka**: určuje tvar výsledného zvuku
- **Délka**: rozmezí, jak dlouhý má být výsledný zvuk, musí být definované, ale někdy se to dělá pouze pomocí obálky.
- **Frekvence**: tím je myšleno, že frekvence nemusí být po celý čas konstantní a tímto parametrem by se mělo určit, kdy se má zpomalit či zrychlit, aby výsledný signál zněl požadovaným způsobem



Obrázek 2.8: Amplituda v čase pro zvuk bass drum

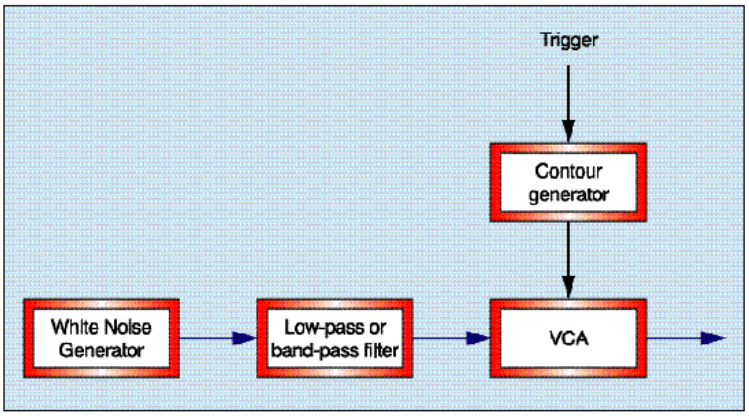
- **Snare drum**: Tento zvuk také popíši pomocí blokového schématu (viz obrázek 2.9). Jelikož se některé bloky budou opakovat z výše uvedeného blokového schématu (viz obrázek 2.7), popíšu jen ty bloky, které již nebyly popsané výše. [3]

  Noise generátor je v digitálním pojmu zjednodušeně jakýkoliv generátor, který generuje hodnoty mezi -1 a 1. [3]

  Filtr, který zde může být buď s dolní propustností, nebo s pásmovou propustností. Filtr s dolní propustností je filtr, který propouští signál s nižší frekvencí než je určitá mezní frekvence. Filtr s pásmovou propustností

(pásmový filtr) je zařízení, kterým prochází signál s frekvencí v určitém daném rozsahu. Tedy může mít dolní i horní omezení. [3]

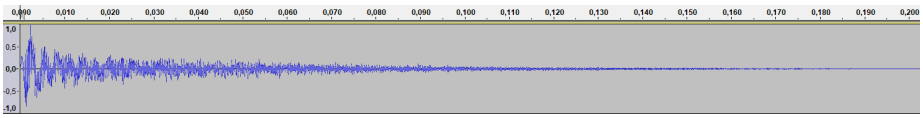
Contour generátor (generátor obrysů) zjednodušeně se jedná o generátor, který je podobný obálce, Tedy generuje, jaký tvar má mít výsledný zvuk. [3]



Obrázek 2.9: Blokové schéma snare drumu (převzato z [3])

Uvedu zde také obecné parametry, které jsou potřeba k vytvoření tohoto zvuku (viz obrázek 2.10), a rovněž popíšu jen parametry, které již nebyly výše uvedené. Mezi tyto parametry patří:

- **Sample rate**
- **Obálka (obrys)**
- **Délka**
- **Frekvence**
- **Hodnota mezní frekvence**
- **Generátor hluku**

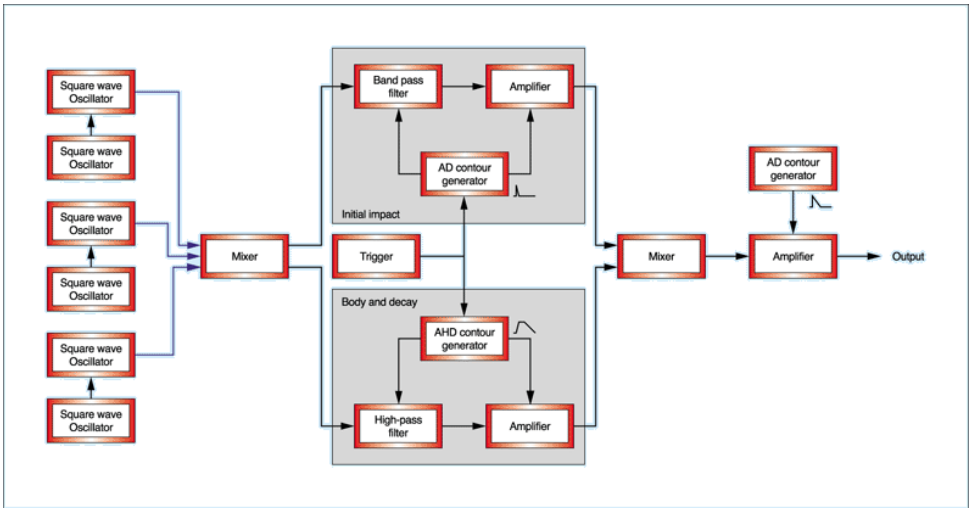


Obrázek 2.10: Amplituda v čase pro zvuk snare drum

- **Hihat**: Tento zvuk také popíšu pomocí blokového schématu (viz obrázek 2.11), který je o něco složitější, ale toto blokové schéma se může ještě zjednodušit. Je možno nahradit bloky oscilátory a 1. mixer vyměnit za generátor hluku. Je ale jednoznačně lepší použít oscilátory, které generují čtvercový signál. V mixeru se vygenerované signály smíchají dohromady. Dále je možno nahradit všechny bloky, které jsou napravo od 1. mixeru, třemi bloky, které jsou za sebou - pásmovým filtrem, obálkou a filtrem s horní propustností. Neuvedl jsem zatím pouze blok filtr s

horní propustností, ostatní bloky jsou popsané v bass drumu a snare drumu. [3]

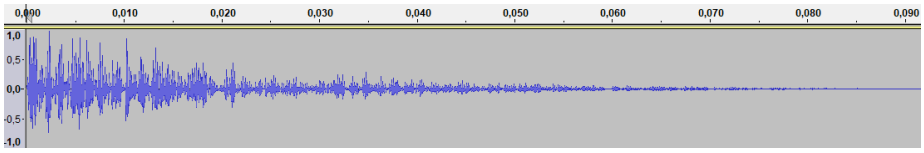
Filtr s horní propustností je filtr, který propouští signál s vyšší frekvencí než je určitá mezní frekvence. [3]



Obrázek 2.11: Blokové schéma hi-hatu (převzato z [3])

Také zde uvedu obecné parametry, které jsou potřeba k vytvoření tohoto zvuku (viz obrázek 2.12), a rovněž popíšu jen parametry, které již nebyly výše uvedené. Mezi tyto parametry patří:

- **Sample rate**
- **Obálka (obrys)**
- **Délka**
- **Frekvence**
- **Hodnota mezní frekvence pro pásmový filtr**
- **Hodnota mezní frekvence pro filtr s horní propustností**
- **Generátor hluku nebo typ signálů pro oscilátory**



Obrázek 2.12: Amplituda v čase pro zvuk hihat

Kapitola 3

Rešerše neuronových sítí

V této kapitole je uvedená celá rešerše tzn.: od uvedení pojmu umělá inteligence až po popis implementací, které používají architekturu WaveNet.

3.1 Umělá inteligence

Abych mohl srozumitelně vysvětlit, co se skrývá pod pojmem neuronové sítě, musím nejprve přiblížit abstraktnější souvislosti, a to pojmy inteligence a umělá inteligence. Tedy tato kapitola pojednává obecně o problematice umělé inteligence. Zde se především dozvíte, jaké techniky se obecně používají pro vytvoření umělé inteligence.

3.1.1 Inteligence

Nejdříve bychom měli popsat inteligenci člověka, protože pokud chceme pracovat s umělou inteligencí. Je dobré vědět, co pojem inteligence znamená. To se budu snažit popsat v této kapitole.

Je několik druhů inteligence, ale mě zajímá především hudební inteligence. Hudební inteligence je schopnost vytvářet a rozumět významu vytvoření zvuku. Jinými slovy - jak správně složit/komponovat hudbu. Například AI by měla znát pojem rytmus a umět ho využít jako hudebníci či zpěváci.[18]

Z čeho se skládá inteligence?

- **Uvažování:** je soubor procesů, které člověku umožňuje poskytovat základ pro posouzení, rozhodování a předpovídání. Jsou dva typy uvažovaní. [18]

  První typ je induktivní uvažování. Můžeme jej popsat jako konkrétní vyjádření, které obsahuje celou řadu obecných prohlášení. Tato prohlášená mohou být často i mylná. Např.: Polo je rychlé, protože všechna auta jsou rychlá.

Druhý typ přestavuje deduktivní uvažování, které můžeme popsat jako uvažovaní začínající obecným prohlášením a dále zkoumající možnosti, jak dosáhnout určitého logického závěru. Najdeme mnoho příkladů, protože člověk toto uvažování používá běžně. Např.: knihy, filmy a seriály o Sherlocku Holmesovi, kde je na konci příběhu deduktivně vysvětlené, jak daný případ vyřešil. Nebo v karetní hře Bang, musíte v roli šerifa zabít bandity. Tedy musíte pomocí dedukce odhalit, kdo je bandita.

- **Učení se:** je získávání znalostí nebo dovedností. Učení také zvyšuje povědomí o předmětech, které jsem získali studiem. [18]

  Učení můžeme rozdělit do několika kategorií. Zde popíši alespoň tři z nich:

  - Sluchové - schopnost učit se poslechem. Například: studenti si přehrají nahrané zvuky
  - Pohybové - schopnost učit se pohybem. Například: při psaní- jaké svaly se musí použít k tomuto úkonu.
  - Prostorové - schopnost učit se pomocí vizuálních podnětů, jako jsou obrázky, atd.
  - A další...

- **Řešení problému:** Jedná se o proces, ve kterém se člověk snaží dospět k požadovanému řešení odlišného od současného stavu. Např.: máme zadaný příklad, řešení problému je výpočet daného příkladu. Řešení problému je také rozhodování (rozhodování je proces výběru nejvhodnější varianty ze všech možných možností). [18]

- **Vnímání:** je proces získávání, přeformulovaní a vybírání informací ze smyslů. Smyslem je myšleno například: sluch. [18]

- **Hudební inteligence:** zde může být uvedená i jiná např.: matematická, atd. Jinými slovy: inteligence je pojmenovaná podle toho, na jakou oblast lidské činnosti se zaměřuje. Např.: Hudební inteligence se zaměřuje na veškerou práci s hudbou. [18]

3.1.2 Umělá inteligence

Umělá inteligence se snaží, aby software i hardware myslel podobným způsobem, jako myslí lidé. Řekneme, že se jedná o AI (česky UI), která se učí podobně jako lidský mozek. Díky tomu AI může vyřešit daný problém, jenž mu byl přidělen k vykonání. [18]

Umělá inteligence je technologie, která je založená na výsledcích jiných oborů - počítačové vědy, biologie, psychologie, matematika, a další. [18]

Hlavním cílem AI je vývoj počítačových funkcí spojených s lidskou inteligencí, jako je uvažování, učení a řešení problémů.

Hlavní filozofická otázka je tato: *„Může stroj myslet a chovat se jako to dělají lidé?“*. Znamená to tedy, že vývoj AI začal za účelem vytvořit podobnou inteligenci ve strojích, kterou mají nebo používají lidé. [18]

Kde najdeme umělou inteligenci?

- Počítačové hry
- Zpracování přirozeného jazyka
- Kamerové systémy
- Inteligentní roboti
- Generování hudby
- Mnoho dalších aplikaci

3.1.3 Rozdíly inteligence člověka a stroje

Když jsem se již věnoval pojmům inteligence a umělá inteligence, je nutné se podívat i na rozdíly mezi nimi. To se pokusím vysvětlit v této sekci.

Lidé vnímají podle předem daných vzorců, naopak stroje vnímají pomocí sad pravidel a dat, které jim byly přidělené.

Lidé uvažují především deduktivně, stroje obecně zase induktivně. Také je rozdíl v ukládání a vyvolávaní dat, protože stroje musí použít vždy vyhledávací algoritmus, ale člověk si nemůže pamatovat tolik věcí jako stroj a zároveň si může zapamatovat data jednodušeji než on. Např.: posloupnost čísel 1, 3, 5, 7 tak, že si pamatuje první a poslední číslo z posloupnosti a zapamatuje si, o kolik se další číslo zvětšuje, Tohle ale bohužel stroj nemůže využít, protože jeho software musí být co nejobecnější. [18]

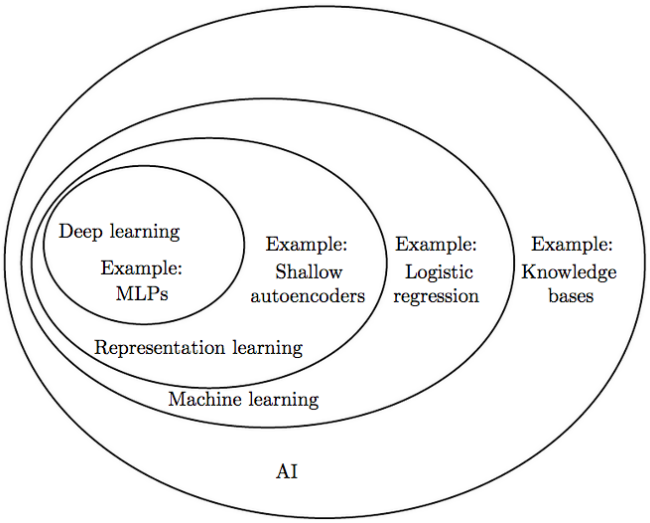
Člověk si může domyslet celý objekt, i když některá část chybí nebo je objekt zkreslený, např. čtverec (viz Obrázek 3.1). Stroj si danou věc ale nemůže domyslet, proto bude pracovat se špatným objektem. [18]

Obrázek 3.1: Nedokreslený čtverec

3.1.4 Techniky

V této podkapitole přiblížím možnosti využití umělé inteligence, které označuji za techniky.

Techniky je možné rozdělit na dvě větší oblasti, a to strojové učení (machine learning) a ostatní (viz Obrázek 3.2)



Obrázek 3.2: Rozdělení AI (převzato z [4])

Z obrázku lze vyčíst, že opravdu velikou část AI zabírá strojové učení. Je to způsobeno tím, že strojové učení se neustále rozvíjí a také se o něm více píše, což má za následek zintenzivnění výzkumu v této oblasti.

Strojové učení

Strojové učení lze rozdělit podle několika algoritmů učení:

- **Učení s učitelem (supervised learning)**: počítači je předán příklad vstupů s jejich požadovanými výstupy. Cílem je naučit se obecná pravidla pro mapování vstupů a výstupů. Příklady konkrétních učení: Náhodné lesy (random forests), Bayesovské sítě (Bayesian networks), Algoritmus nejbližšího souseda (nearest neighbor algorithm), umělé neuronové sítě (artificial neural networks), atd. [19]
- **Učení bez učitele (unsupervised learning)**: učebnímu algoritmu se nedávají žádné popisky, proto algoritmus musí najít sám strukturu svého vstupu. Tedy pro toto učení muže být cílem nalezení skrytých vzorů v datech. Příklady konkrétních využití: Generativní topografická mapa (Generative topographic map), umělé neuronové sítě (artificial neural networks), shlukovací/klasteringová analýza (cluster analysis), atd. [19]
- **Kombinace učení s učitelem a bez učitele (semi-supervised learning)**: stroj dostane neúplná tréninkové data, což znamená, že dostane pouze vstupní i výstupní data. Ale výstupní data neobdrží všechna.

Jinými slovy dostane všechna vstupní data, ale výstupní data mohou obsahovat jen jedu sekvenci dat. Konkrétní příklady: Aktivní učení (Active learning), (generativní modely) Generative models, atd. [19]

- **Zpětnovazebné učení (reinforcement learning)**: tréninková data jsou dána pouze jako zpětná vazba k činnostem programu v dynamickém prostředí např.: hra proti soupeři. Konkrétní příklady: Q-učení (Q-learning), learningové automaty (learning Automaty), atd. [19]
- **Další metody**: Snížení dimenze (Dimensionality reduction), Soustředění (ensemble learning), Meta učení (Meta learning), Hluboké učení (Deep learning). Příklady: Analýza faktorů (Factor analysis), Metadata, Hluboké konvoluční neuronové sítě (Deep Convolutional neural networks), Evoluční algoritmus (Evolutionary algorithm), atd. [19]

Ostatní

Do ostatních učení patří vše, co nezahrnuje strojové učení. Např.: Behaviorální stromy. Tento strom je matematický model, který se převážně používá ve videohrách. Tento strom reprezentuje mozek NPC (Non-playable character) jednotky, které řídí vykonávání jednotlivých akcí i reakci na vnější impulzy. Listy tohoto stromu reprezentují dílčí akce a vnitřní uzly pak rozhodovací mechanismus. [20].

3.2 Rešerše neuronových sítí

V této kapitole budou rozebrány neuronové sítě, které jsou podmnožinou strojového učení (viz podkapitola 3.1.4). Také zde budou popsané typy neuronových sítí.

Neuronové sítě jsou nedeklarativní systémy umělé inteligence. Nedeklarativní znamená, že nemusí mít předem definovaná pravidla, kterými se neuronová síť řídí. Pravidla řešení se stanovují až během učení. Neuronové sítě umožňují ideální řešení sítí, neboť jimi realizované sítě jsou nezávislé na topologii a také propojovacích prvcích. Nevýhodou těchto sítí jsou především velké nároky na uživatele jak z podhledu dat, tak i z pohledu trénování. Jelikož neznáme pravidla na vstupu, vytváří se během testování. Může se říci, že vytváříme nějaké nejistoty v řešení, což se nazývá fuzzy logika. Matematicky vyjadřuje zachycené nejistoty spojené s poznávacím procesem. Poskytuje interferenční mechanizmus pro poznávání nejistoty. Výsledky řešení jsou závislé na vstupní, skryté a výstupní vrstvě. Rychlost řešení je závislá na topologii sítě.[5]

Neuronová síť je orientovaný graf s ohodnocenými hranami, kde rozeznáváme uzly vstupní, výstupní a skryté a kde hrany reprezentují tok signálu.

Hrany jsou ohodnoceny parametrem zpracování signálu, který je nazýván vahou.[21]

Jinými slovy: Propojení mezi dvěma uzly má vždy orientační směr, což určuje, z jakého do kterého uzlu vedou informace nebo data. Propojení mohou být obousměrná. Jedna z nejdůležitějších vlastností propojení je váha, která určuje intenzitu přenášení informace. Váha může mít kladnou i zápornou hodnotu.

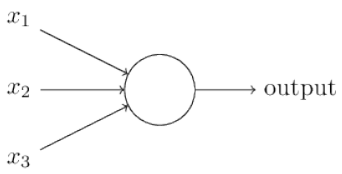
Zde uvádím stavební jednotky pro neuronové sítě.

- Umělý neuron
- Synapse
- Váha
- Vstup a vstupní vrstva
- Práh a přenosové funkce
- Výstup a výstupní vrstva
- Skrytá vrstva

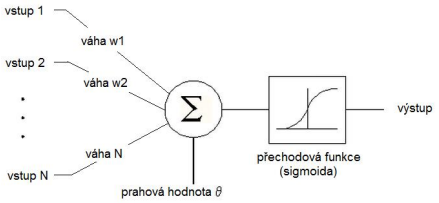
3.2.1 Stavební jednotky neuronových sítí

Jeden z typů umělých neuronů se nazývá perceptron, ale v současné době se častěji používají jiné modely umělých neuronů jako například sigmoidní neurony.

Perceptron má pouze jeden výstup, ale naopak může mít libovolný počet vstupů. Vstupy i výstup jsou binární číslice. Tedy můžou nabývat dvou hodnot, které jsou 0 a 1(viz Obrázek 3.3).[5]



(a) : Jednoduché znázornění perceptronu (převzato z [5])



(b) : Podrobnější znázornění perceptronu (převzato z [22])

Obrázek 3.3: Perceptron

Váhy jsou reálná čísla, která vyjadřují význam příslušných vstupů do výstupu. Výstup neuronu 0 nebo 1 je určen podle toho, zda vážená součet:

$$f(x) = \sum_i w_i x_i, \text{ kde x reprezentuje vstupy a w reprezentuje váhy} \quad (1)$$

je menší nebo větší než nějaká prahová hodnota. Také můžeme říci, že váha je násobitelem nesené informace.[5]

Prahová hodnota je reálné číslo stejně jako váhy. Pomocí prahové hodnoty se aktivuje výstup neuronu a přenosová funkce je pak funkcí, která upravuje výstup na hodnoty, které se dále šíří v neuronové síti. Příklady přenosových funkcí: Skoková přenosová funkce (0 a 1 kde práh definuje zlom mezi 0 a 1), sigmoidální přenosová funkce (sigmoida), atd.[5]

Výstup umělého neuronu může vypadat například takto:

$$\text{výstup} = \begin{cases} 0, & \text{pokud } \sum_i w_i x_i \leq \text{prahová hodnota} \\ 1, & \text{pokud } \sum_i w_i x_i > \text{prahová hodnota} \end{cases} \tag{2}$$

Můžeme změnit nerovnice tak, že nahradíme $\sum_i w_i x_i$ pomocí skalárního součinu dvou vektorů: $\mathbf{w} \cdot \mathbf{x} = \sum_i w_i x_i$, kde w a x jsou vektory a přesunutím prahové hodnoty na druhou stranu nerovnosti a nahradit ji zkreslením perceptronu, $b \equiv -\text{práh}$. Pravidlo perceptronu lze tedy přepsat:

$$\text{výstup} = \begin{cases} 0, & \text{pokud} w * x + b \leq 0 \\ 1, & \text{pokud} w * x + b > 0 \end{cases} \tag{3}$$

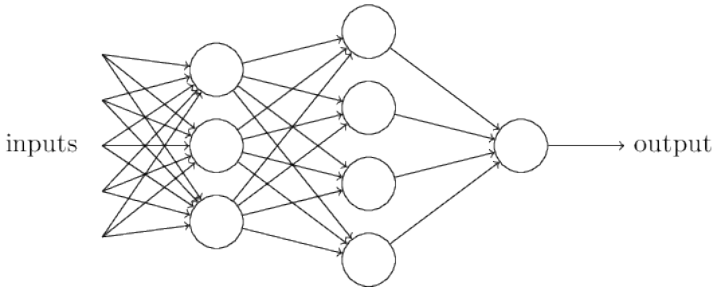
Zkreslení perceptronu může způsobit, že vyjde jiný výstup než ten, který bychom chtěli. Pokud je zkreslení velmi negativní, mluvíme o tzv. zaujatosti.[5]

Perceptrony mohou být reprezentované také pomocí těchto logických funkcí: AND, OR nebo NAND.

Je zřejmé, že perceptron není úplný model lidského rozhodování. Ale můžeme říci, že perceptron může zvážit různé druhy důkazů, pomocí kterých se rozhoduje.

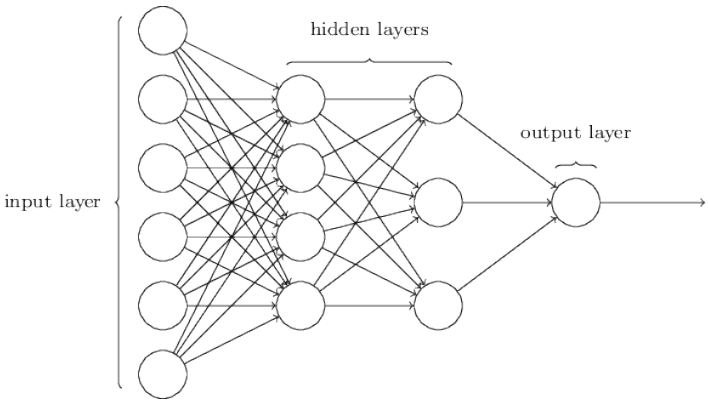
Sigmoidní neurony jsou podobné perceptronům, ale modifikované tak, že malé změny v jejich váhách a zkresleních způsobují pouze malou změnu v jejich výkonu. To je zásadní fakt, který umožní, aby se síť z sigmoidních neuronů učila.[5]

Synapse jsou hrany (propojení) mezi jednotlivými umělými neurony. Počet synapsí určuje velikost nebo mohutnost neuronové sítě, tzn. že větší počet synapsí v neuronové síti má možnost uchovat větší množství dat.[21]



Obrázek 3.4: Jednoduchá síť (převzato z [5])

V síti (viz Obrázek 3.4) první sloupec perceptronů, který nazveme první vrstva. Provádí obecně jednoduché operace s váhami a se vstupy. Perceptrony ve druhé vrstvě rozhodují tak, že zváží výsledky první vrstvy. Díky tomu perceptron ve druhé vrstvě může rozhodovat na složitější a abstraktnější úrovni než perceptrony v první vrstvě. Nečekaně perceptron v třetí vrstvě může udělat složitější rozhodnutí než perceptrony ve druhé vrstvě.[5]



Obrázek 3.5: Neuronová síť s vrstvy (převzato z [5])

Z Obrázku 3.5 je vidět, že vrstva vlevo byla dříve pojmenována jako první vrstva, ale správně se nazývá vstupní vrstva a umělé neurony uvnitř této vrstvy se nazývají vstupní neurony. Vrstva, která je zcela vpravo se nazývá výstupní vrstva, která obsahuje výstupní neurony. Dříve tato vrstva byla pojmenována jako 3. vrstva (viz Obrázek 3.4). Všechny vrstvy, které se nacházejí mezi vstupní a výstupní vrstvou, se nazývají skrytá vrstva. V této vrstvě se dělají téměř veškerá rozhodnutí nebo operace. Význam slova „skrytý“ znamená, že se nejedná o vstup ani o výstup. [5]

3.2.2 Učení neuronové sítě

O učení jsem se zmínil již v kapitole 3.1.4, ale v této kapitole zmíním pouze, jaké typy učení se používají v neuronové síti. Následně přiblížím nejpoužívanější algoritmus, který nese název Backpropagation.

Pro učení neuronové sítě se používají tyto dvě metodiky: učení s učitelem a učení bez učitele. Obě metodiky jsou obecně popsané v kapitole 3.1.4, ale konkrétněji,pro neuronové sítě, je popíšu zde .

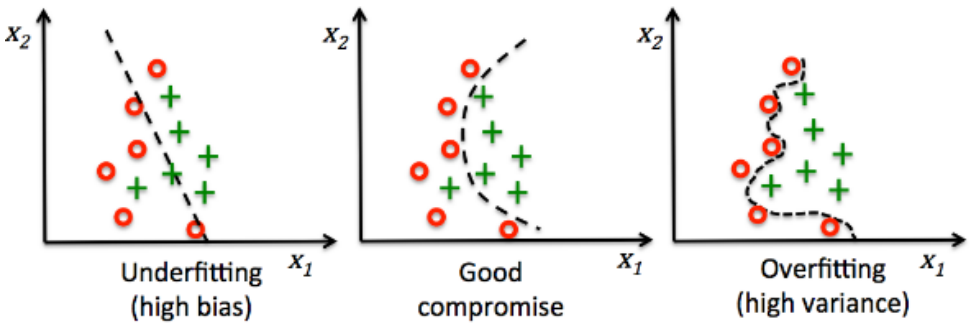
Při učení s učitelem má algoritmus k dispozici konečnou množinu dvojic, které představují vstupy a odpovídající výstupy dané úlohy. Tedy tyto množiny představují správné chování sítě, jehož bychom chtěli dosáhnout. Často bývá rozdělena na trénovací a testovací část. Při trénování neuronové sítě se předává vstupní množina trénovacích dat na vstupy a po zjištění výstupu sítě se postupně upravují hodnoty vah, které jsou mezi neurony. Je to kvůli tomu, aby odchylka výstupu sítě byla co nejmenší oproti korektnímu výstupu ze zadané množiny dat. [23]

Při učení bez učitele nemá neuronová síť k dispozici žádnou tréninkovou množinu dat. Hledání správných výsledků probíhá na principu shlukování, kdy se mezi vstupními daty hledají navzájem podobné elementy. [23]

Snažíme se neuronové sítě naučit vše tak, aby byly úspěšné na všech datech. Či spíše aby byly 100% úspěšné, přestože toho neuronové sítě nemohou dosáhnout. Ve skutečnosti je možné dosáhnout úspěšnosti 95% až 99% na datech, která jsou určená k trénování neuronové sítě. Tomu se říká bias. Naopak na testovacích datech je možné dosáhnout 85% až 95% úspěšnosti, což je úspěšnost vysoká.

Mohou nastat dva chybné výsledky učení neuronových sítí (viz obrázek 3.6):

- Underfitting zjednodušeně znamená, že se neuronová sít nic nenaučila.
- Overfitting zjednodušeně znamená, že se neuronová sít přeučila na tréninkové sadě a špatně generalizuje.



Obrázek 3.6: Možnosti výsledků učení neuronových sítí (převzato z [6])

Backpropagation

Tento algoritmus se používá u učení s učitelem a je považován za velmi rychlý algoritmus.

Algoritmus pracuje tak, že na začátku učení náhodně nastaví synaptické váhy neuronů, a to v nějakém intervalu např.: od -0.5 do 0.5. Poté se postupně, od vstupní po výstupní vrstvu neuronové sítě, spočítá výstup každého neuronu v jednotlivých vrstvách. Výstup jednotlivých neuronů se počítá například pomocí prahové funkce. Jakmile se spočítá výstup (výstupní vektor) výstupní vrstvy neuronové sítě, tak se porovná s požadovaným výstupem (výstupním vektorem), a tím se zjistí chyba neuronové sítě. Na základě toho se zpětně vypočítá, jak se mají změnit synaptické váhy neuronů. Celý výše popsaný proces (postup) se opakuje pro celou tréninkovou množinu, a to tak dlouho, dokud se chyba neuronové sítě nezmenší pod požadovanou hranici, nebo nepřekročí časový limit. [5]

Prozkoumáním algoritmu můžeme zjistit, proč se říká mu backpropagation. Je to způsobeno tím, že se vrací zpět ke vstupní vrstvě pomocí výstupních (chybových) vektorů, určité přenosové funkce a zkreslení a postupně mění hodnoty vah.

3.2.3 Využití neuronových sítí ve zvuku

Neuronové sítě se používají na všech úrovní syntézy zvuku. Pod pojmem úroveň syntézy zvuku se skrývá např. zvukový signál, souzvuky, melodie, skladby, atd. Jinými slovy: s neuronovými sítěmi je možné se setkat u vytvoření zvukového singlu, nebo tvorby nových melodií atd. Ale nejčastěji se používají neuronové sítě pro generování skladeb, avšak ne vždy to může znít podle našich představ.

Je evidentní, že se neuronové sítě požívají u mnohých činností týkajících se zvuku, tudíž jsem se zaměřil v rámci této diplomové práce na využití neuronových sítí na práci se zvukovým signálem. Jinými slovy: jsem se zaměřil na nejnižší úrovně ve zvukové syntéze, což obnáší vytvoření parametrického syntezátoru, kde se budu snažit reprodukovat zvukový signál za pomocí neuronových sítí, vstupních dat a parametrů.

Tato sekce dále pokračuje sekcemi 3.2.4, 3.3 a 5

3.2.4 Typy neuronových sítí

V této kapitole popíši typy neuronových sítí, které se používají nejčastěji pro práci se zvukem.

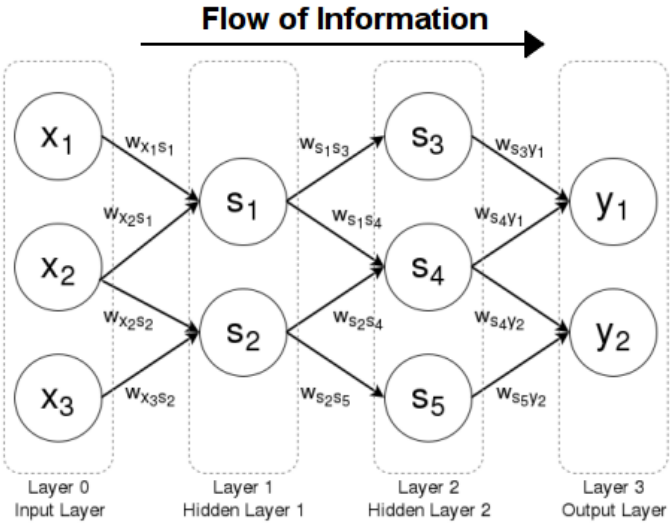
Typy (architektury) neuronových sítí se určují pomocí těchto dvou aspektů: uspořádání jednotlivých neuronů a propojení mezi nimi. Dále se neuronové sítě mohou rozdělovat dle algoritmu učení. I přesto existuje mnoho různých druhů neuronových sítí, ale lze je rozdělit na dvě základní skupiny: dopředné neuronové sítě (Feedforward Neural Networks) a rekurentní neuronové sítě (Recurrent Neural Networks). [24]

Našel jsem nejvhodnější druhy neuronových sítí pro generování nebo vytváření hudby. Jsou to řízené rekurentní jednotky (Gated recurrent units), které se označují GRU, konvoluční neuronové sítě (Convolutional Neural Networks), které nesou označení CNN, rekurentní neuronové sítě, které mají zkratku RNN a dlouhá krátkodobá paměť (Long Short-Term Memory) s označením LSTM. Tyto konkrétní příklady blíž popíši níže. [25]

Dopředné neuronové sítě (FFNN)

Nejprve přiblížím dopředné neuronové sítě (dále jen FFNN), protože jejich princip je nesmírně důležitý k vysvětlení dalších neuronových sítí.

FFNN mohou být propojeny v libovolné struktuře, ale o FFNN se jedná tehdy, jestliže se mezi propojenými neurony v grafu sítě nevyskytuje cyklus. Nazývají se FFNN , protože informace se pohybuje pouze vpřed v síti (bez smyčky), nejprve přes vstupní vrstvu, pak přes skrytou vrstvu a nakonec přes výstupní vrstvu. FFNN vyhodnocují každý vstup zvlášť, bez ohledu na výstupech ostatních uzlů ve stejné vrstvě. Tedy perceptrony jsou nezávislé. [7]



Obrázek 3.7: FFNN (převzato z [7])

Použití v hudbě. Klasická FFNN se k různým aplikacím již moc nepoužívá, tudíž se nepoužívá ani v hudbě, protože klasické FFNN nahradili v dnešní době CNN. Ale i tak můžeme najít některé aplikace, jež slouží klasifikaci akordů. [26]

Konvoluční neuronové sítě (CNN)

Zde popíši konvoluční neuronové sítě (dále jen CNN).

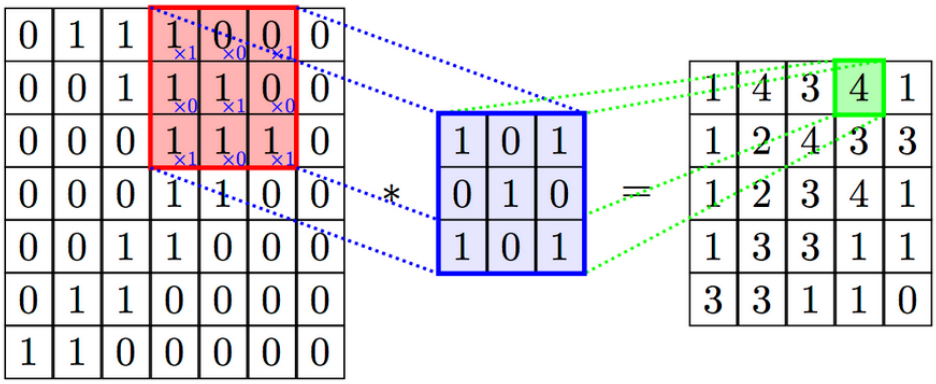
Zajímavostí CNN je propojení neuronů, které je inspirováno organizací zrakové kůry u zvířat. [27]

CNN patří do kategorie FFNN. Byly navrženy zejména pro rozpoznávání dvourozměrných obrazových dat. Díky konvoluci není síť tolik náchylná k chybám, jelikož konvoluce do jisté míry ignoruje posuny nebo jiné deformace vstupních dat. V těchto sítích se používá algoritmus zpětného šíření chyby (backpropagation). CNN je tvořena více konvolučními vrstvami. Mezi těmito vrstvami často bývá vrstva, která je zodpovědná za tzv. pooling, shlukování okolních pixelů do jednoho a tím postupné zmenšování velikosti vstupního obrázku. Po těchto vrstvách většinou následuje plně propojená vrstva, která propojuje všechny výstupy předchozí vrstvy se všemi svými vstupy. Mezi další používané vrstvy se řadí například Dropout, provádějící náhodné vynechávky dat pro lepší generalizaci, nebo Softmax, která na konci sítě určuje pravděpodobnosti jednotlivých určených tříd. [27]

Konvoluční vrstva obsahuje sadu filtrů zvaných kernel (jádro), které jsou zodpovědné za vytváření příznakových map. Tyto mapy jsou výsledkem konvoluce daného filtru přes celou šířku a výšku vstupního obrázku. Tyto filtry slouží k tomu, aby byla umožněna správná detekce požadovaných vlastností nezávisle na posunutí obrázku. Výsledkem je skalární součin dat obrázku a filtru. [27]

Pooling vrstva, má na starosti sdružování okolních pixelů do jednoho, čímž zmenšuje rozměry výstupních dat a tím i počet aktivací neuronů potřebných pro jejich zpracování. [27]

Na obrázku 3.8 je vyobrazena tzv. 2D konvoluce neboli obě výše zmíněné vrstvy. Tedy je ukázáno, jak vrstvy v CNN pracují. [8]



Obrázek 3.8: 2D konvoluce (převzato z [8])

Použití v hudbě. CNN, jak již jsem výše uvedl, slouží primárně pro práci s obrázky. Z toho plyne, že může velmi dobře pracovat se spektrogramem. Spektrogram je vizuální znázornění spektra z kmitočtů ze zvuku nebo jiného signálu. Zde uvedu dvě použití, které pracují se spektrogramem, jenž využívají jako vstupní data:

- Klasifikace zvuku [28]
- Klasifikace zpěvavé melodické extrakce [29]

Když se budeme zjišťovat, jestli je CNN schopná zpracovávat digitalizovanou hudbu, či nikoliv, musím konstatovat, že CNN je v tomto velmi úspěšná. Za zmínku stojí, že obecné použití CNN v hudbě je:

- Vytvoření podobného zvukové stopy podle předlohy. Jako jeden příklad muže být uvedený autoencoder [30]
- Jakákoliv klasifikace např.: melodie, akordů, atd. [29]
- Jakákoliv extrakce např.: melodie, akordů, atd. [31]
- Jakékoliv rozpoznávání např.: melodie, akordů, atd. [32]

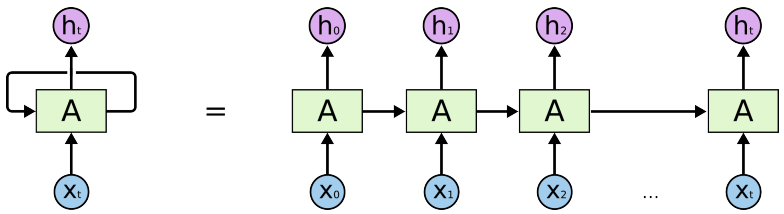
Rekurentní neuronové sítě (RNN)

Rekurentní neuronové sítě (dále jen RNN) vylíčím v této podkapitole.

FFNN a CNN jsou příliš omezené, protože akceptují vektory s pevnou velikostí jako vstup (např. obrázek) a produkují ve vektoru s pevnou velikostí výstup, a také provádějí mapování pomocí pevného množství výpočetních kroků.

Rozdíl oproti FFNN je, že RNN obsahují v grafu smyčky, které jim umožňují uchovávat informace. Hlavním důvodem, proč jsou RNN zajímavější, je to, že nám umožňují pracovat se sekvencemi vektorů. Sekvence na vstupu, výstupu nebo v nejběžnějším případě obojí. RNN kombinují vstupní vektor se svým stavovým vektorem s pevnou (ale naučenou) funkcí pro vytvoření nového vektoru stavu. To může být v programovacím smyslu interpretováno jako běh programu s určitými vstupy a interními proměnnými. Tímto způsobem jsou RNN v podstatě popsány programy. Ve skutečnosti je známo, že RNN jsou Turing-Complete v tom smyslu, že mohou simulovat libovolné programy. [33]

V jádru mají RNN klamně jednoduché rozhraní. Přijímají vstupní vektor a poskytují výstupní vektor. Obsah výstupního vektoru je ovšem ovlivněn nejen vstupem, který jste právě vložili, ale také celou historií vstupů, které byly v minulosti vloženy. [33]



Obrázek 3.9: Ukázka RNN (převzato z [9])

Použití v hudbě. RNN mají v hudbě více využití oproti CNN. ale rád bych tuto informaci uvedl na pravou míru. Znamená to, že RNN může být použita na stejné účely jako CNN a ještě na něco více. Níže uvede jaké jsou možnosti použití RNN v hudbě:

- vytvoření podobného zvukové stopy podle předlohy. Jako jeden z příkladů muže být uvedený autoencoder [34]
- Jakákoliv klasifikace např.: melodie, akordů, atd. [35]
- Jakákoliv extrakce např.: melodie, akordů, atd. [35]
- Jakékoliv rozpoznávání např.: melodie, akordů, atd. [36]
- Skládání nebo komponování hudby. Do této kategorie paří jakékoliv skládání hudby, tedy tu může být nejen harmonické i neharmonické skládaní, ale také vocoder, syntezátor, atd. [37]
- Napodobení hudebního nástroje.

Dlouhá krátkodobá paměť (LSTM)

Dlouhá krátkodobá paměť zde popíši.

Jak již vyplývá z českého překladu, dlouhá krátkodobá paměť, z anglického Long short-term memory (dále jen LSTM), patřící do kategorie RNN, je schopná realizovat krátkodobou paměť i dlouhodobou paměť. RNN jsou schopny rozeznat vzorce ve vstupních datech pouze v malém časovém odstupu kvůli mizejícímu nebo explodujícímu gradientu. LSTM je využívá speciální uzávěrový mechanismus, který ovládá přístup k vnitřnímu stavu buňky. Uzávěry můžou bránit zbytku sítě, aby upravoval obsah vnitřního stavu buňky, při propagování chyby zpět do sítě gradient, proto tak lehce nemizí oproti obyčejným RNN. [9]

Vnitřní stav LSTM určuje v každém kroku mnoho faktorů, které buď k aktuálnímu stavu přidávají informace, nebo ubírají pomocí zmiňovaných uzávěrů. Výsledek uzávěry určuje, jestli danou informaci propustí nebo nepropustí. [9]

Použití v hudbě. Mohl bych zde vyjmenovat jednotlivé použití LSTM v hudbě, ale jsou totožná s těmi, které jsem uvedl v kapitole 3.2.4. LSTM se v hudbě nejčastěji používají ke komponování určitých stylů hudby, např. jazz [38], blues [39] atd.

Gated recurrent units (GRU)

Gated recurrent units (dále jen GRU) přiblížím v této kapitole.

GRU může být také považována za variantu LSTM, protože obě jsou navrženy podobně a v některých případech mají stejně vynikající výsledky. GRU jsou vylepšená verze RNN. stejně jako LSTM byly vytvořeny proto, aby vyřešily problém mizejícího nebo explodujícího gradientu v RNN.

GRU má dvě brány, které se nazývají resetovací a aktualizační. Resetovací brána určuje, jak spojit nový vstup s předchozím stavem v paměti. Aktualizační brána určuje, kolik předchozích stavů v paměti bude držet kolem. Pokud nastavíme resetovací bránu pro všechny na 1 a aktualizační bránu pro všechny na 0, opět dostaneme obyčejnou RNN. Jinými slovy: GRU má stejný princip jako LSTM, ale počtem bran, kde GRU má 2 a LSTM má 3. [40]

Použití v hudbě. Opět bych v této sekci mohl vypsat jednotlivá použití GRU v hudbě, ale jsou stejná s těmi, jenž jsem popsal v kapitole 2.4.4. Bohužel GRU ještě nemají své nejčastější použití v hudbě, ale jak jsem výše zmínil, jsou považovány za variantu LSTM. Tedy je pravděpodobné, že se časem také budou často používat pro komponování určitých hudebních stylů.

3.3 Architektury neuronových sítí

Tato kapitola popisuje existující architektury neuronových sítí, které je možné využít pro reprezentaci a reprodukci zvukového signálů podle předloh.

Existují tři více známé architektury, a to: WaveNet [10], Magenta (konkrétněji implementace NSynth) [41] a Neural Parametric Singing Synthesizer [42]. Pokud se podíváme pořádně na všechny aplikace (architektury), tak zjistíme, že všechny tyto architektury používají nějakou část WaveNetu, která může být od modelu po celou architekturu. Díky tomu jsem se rozhodl, že se v této kapitole zaměřím zájmena na architekturu WaveNet.

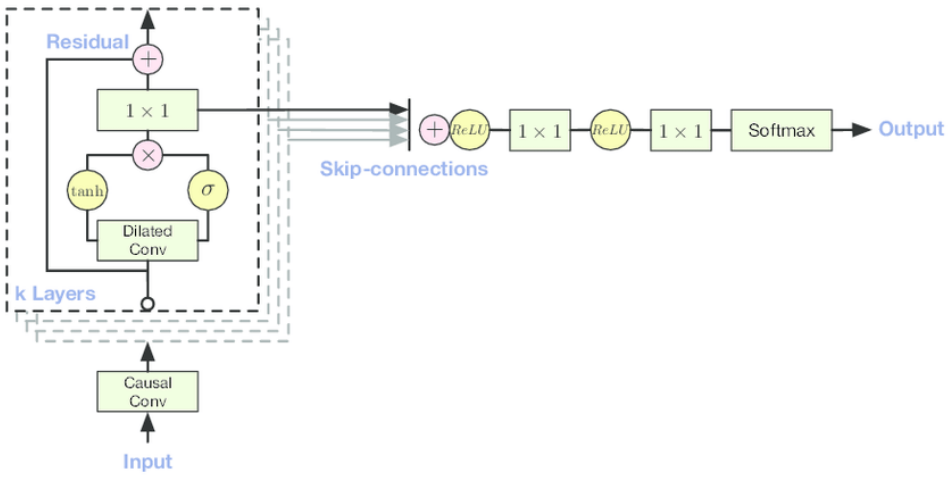
3.3.1 WaveNet

Wavenet je založen na CNN, která v posledních letech funguje velmi dobře v oblasti klasifikace a generování obrazu. Snaží se je posílit aplikace typu text-to-speech prostřednictvím generování přirozenějšího toku v hlasovém zvuku. Nicméně jejich metoda může být použita také na hudbu, jak vstup, tak výstup se sestává ze surového (raw) zvuku. WaveNet může využívat pro práci např.: framework TensorFlow (viz sekce 3.3.2). [10]

Výhodou WaveNetu je to, že pracuje s raw zvuky jako vstupy, a to má za následek, že může generovat jakýkoli druh nástroje, a dokonce i jakýkoli zvuk. Podle mě může být výhodou, že některé zvuky zněly velmi vtipně, i když to nebylo požadované.

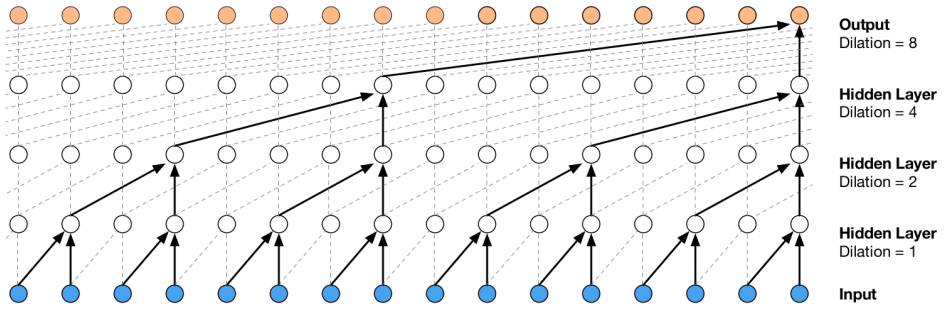
Nevýhodou WaveNetu je, že algoritmus je výpočetně nákladný. K možnosti vygenerovaní nějakého rozumného výsledku, tzn. že výsledek není pouze

šum, je potřeba, aby tréninkový algoritmus běžel minimálně 80 000 krát. Ale to může vést k tomu, že tréninkový algoritmus může běžet i několik dnů v závislosti na hardware, tzn. čím bude horší procesor nebo grafika, tím algoritmus poběží déle.



Obrázek 3.10: Celá architektura (převzato z [10])

Základní koncept architektury WaveNetu, jenž je znázorněný na obrázku 3.10. Podrobněji se architekturou bude tato práce zabývat v sekci 3.3.2. Jak bylo uvedené výše, WaveNet je strukturovaný jako CNN, ale já osobně bych tuto danou architekturu popsal jako pseudo-RNN, protože vytvořený vzorek závisí na předchozích vzorcích. Konvoluční vrstvy WaveNetu mají různé dilatační faktory, které umožňují, aby jeho vnímané pole rostlo exponenciálně s hloubkou a pokrývalo tisíce časových úseků. (viz obrázek 3.11) [10].



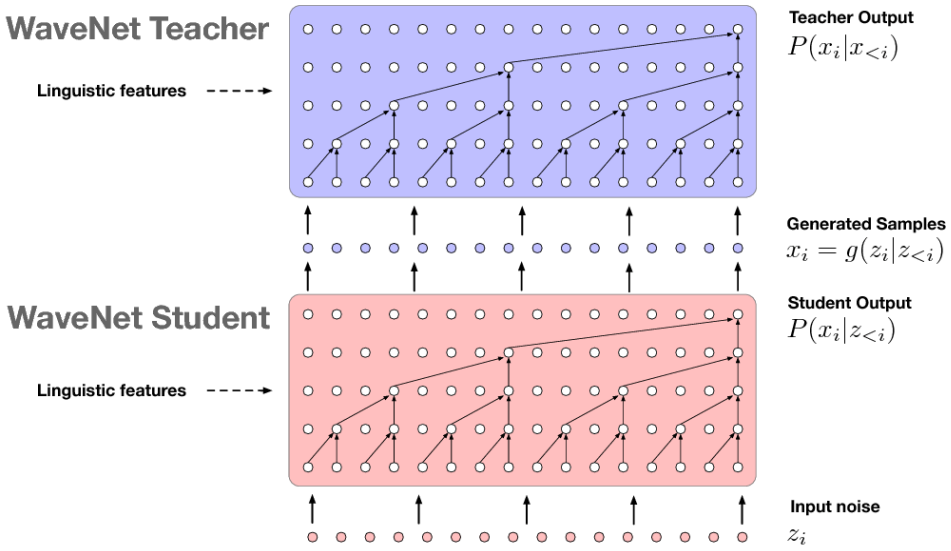
Obrázek 3.11: Princip WaveNetu (převzato z [10])

Jako základní myšlenky pro trénink mohou být použity jakékoli vstupní sekvence, ale trénuje se jako ostatní CNN. Zajímavější část nastává po tréninku, kdy se může vygenerovat vzorek. Pro každý krok vygenerovaní je vyvozena hodnota distribuce pravděpodobnosti. Tato hodnota je potom vrácena zpět do vstupu a je provedena nová předpověď pro další krok. Vytváření vzorků v jednom kroku je poměrně nákladné. Jinými slovy: ve WaveNetu je použito

auto-regresivní spojení, což znamená, že každý nový vzorek je závislý na předchozích vzorcích. [10]

V roce 2017 bylo vytvořeno zajímavé rozšíření pro WaveNet, které se týká toho, pomalejšího generování vzorků. K rozšíření se použil plně vyškolený model WaveNetu, který je určen pro výuku druhé „studentské“ sítě. Tato síť je podobná původnímu WaveNetu, ale liší se tím, že je menší a paralelní. Hlavní aspekt rozšíření je, že generování každého vzorku nezávisí na žádném z dříve vytvořených vzorků, což znamená, že můžeme generovat všechny vzorky současně. Např.: Kdybychom měli vstupní parametry vzorků, které obsahují řeč jednoho řečníka, algoritmus může generovat první slovo, poslední slovo i všechna slova mezi nimi současně. [11]

WaveNet (studentská síť) začíná během tréninku v náhodném stavu. Vstupem studentské sítě je náhodný bílý šum a má za úkol produkovat zvukovou vlnu jako výstup. Vygenerovaná vlna je pak připojená k vyškolenému modelu WaveNetu, který vyhodnocuje každý vzorek a dává studentovi signál, jež mu sděluje, jak je daleko vygenerovaný signál od požadovaného výstupu. To vede k tomu, že studentská síť může být upravená pomocí backpropagation. Jinými slovy: učitelská i studenská síť vytvoří pravděpodobnostní rozdělení pro každou hodnotu zvukového vzorku a cílem je minimalizovat relativní entropii mezi pravděpodobnostními rozděleními obou sítí (viz obrázek 3.12). [11]



Obrázek 3.12: Princip rozšířeného WaveNetu (převzato z [11])

Za zmínku stojí, že studentská síť hraje roli generátoru a učitelská síť je jako diskriminátor. Obě sítě spolupracují tak, aby se studenská sít přizpůsobila výkonu učitelské sítě. [11]

3.3.2 Prostředky pro realizaci a experimenty

Tato sekce přibližuje všechny prostředky (implementace/architektury), které jsem teoreticky prozkoumal před jejich implementací, či před vytvoření experimentů (viz kapitola 5).

Tensorflow

V této sekci je stručně uvedený pojem TensorFlow.

TensorFlow je open source knihovna, která slouží pro numerické výpočty s použitím grafů toku dat. Uzly grafu reprezentují matematické operace, zatímco hrany grafu reprezentují multidimenzionální datová pole (tenzory), které mezi nimi proudí. Tato flexibilní architektura umožňuje nasadit výpočet na jeden nebo více CPU nebo GPU na stolním, serverovém nebo mobilním zařízení bez přepisového kódu. TensorFlow také obsahuje TensorBoard, nástroj pro vizualizaci dat. Jednodušeji by se dal popsat TensorFlow jako open source framework, který se používá pro strojové učení. [43]

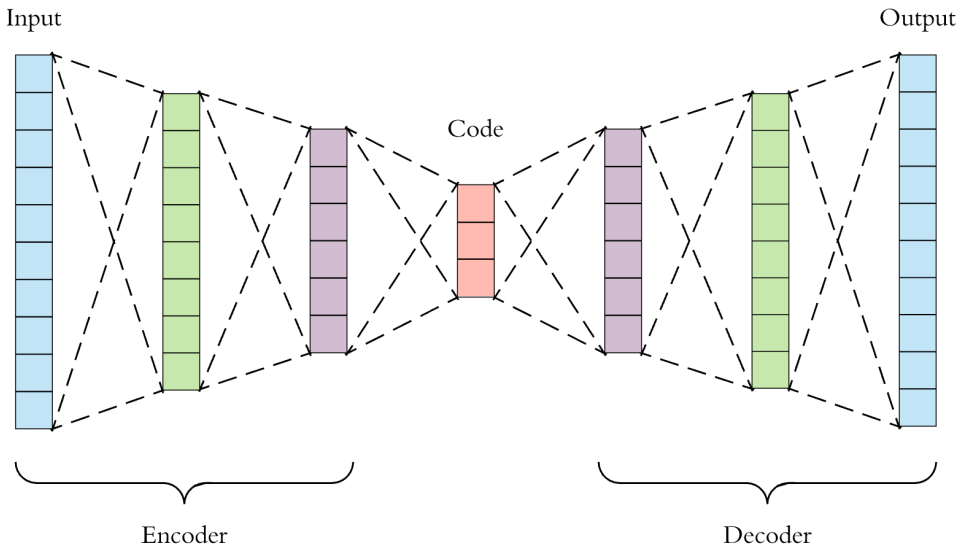
TensorFlow byl vyvinut výzkumnými pracovníky pracujícími v týmu Google Brain v rámci organizace Google Machine Intelligence Research, kteří se zabývali strojovým učením. Systém je dostatečně obecný, aby mohl být použitelný i v celé řadě dalších oblastí. [43]

TensorFlow podporuje mnoho programovacích jazyků, ale jednoznačně nejlepší programovací jazyk pro práci s TensorFlow je Python, protože pro Python má TensorFlow stabilní Python API. Např.: TensorFlow má API bez zpětné kompatibility pro jazyk Java, atd. [43]

MNIST autoencoder

V této sekci je přiblížený pojem autoencoder a také jako konkrétní implementace MNIST autoencoder.

Autoencoder je obecně FFNN (viz sekce 3.2.4), tedy je možné vytvořit architekturu/implementaci autoencoderu jako CNN (viz sekce 3.2.4), a tato NN používá učení bez učitele, ke které používá backpropagation (viz sekce 3.2.2). Ale zajímavé na autoencoderu je to, že vstupní a výstupní vrstva má stejný počet umělých neuronů (viz obrázek 3.13). Skládá se ze 2 částí: kodéru a dekodéru (viz obrázek 3.13). Používají se pro kompresi vstupů, neboli snaží se snížit rozměry vstupů např. pro odfiltrování hluku. Kompresi provádí tak, že komprimuje vstup pomocí kodéru do nízko dimenzionálního kódu a pak dekodér rekonstruuje výstup z této reprezentace. Jinými slovy: komprimuje vstupní data do krátkého kódu, který následně rekonstruuje do něčeho velmi podobného původním datům, to jest před kompresí. [44]



Obrázek 3.13: Autoencoder (převzato z [12])

MNIST autoencoder je zařízení, které pracuje s MNIST daty. MNIST data vycházejí z MNIST databáze, což je rozsáhlá databáze ručně psaných číslic. Číslice byly normalizovány podle velikosti a centrovány v obrázku s pevnou velikostí, která činí 28x28 pixelů (obrázky jsou černobílé). MNIST data se běžně používají pro školení a testování v oblasti strojního učení. Je jednoduché s nimi pracovat, jelikož jsou vytvořena datasety a nemusíme je vytvářet sami. [45]

MNIST autoencoder má důležité parametry pro trénink všech neuronových sítí a jejich implementací/architektur:

- learning rate (míra učení) - je číselná hodnota, která je často označovaná jako hyperparametr, který řídí, o kolik se upraví váhy v NN s ohledem na gradient ztráty.
- num steps (počet tréninkových kroků) - je celočíselná hodnota, která určuje, kolikrát poběží cyklus pro trénování NN
- batch size - je taktéž celočíselná hodnota, jenž definuje počet vzorků, které budou šířeny prostřednictvím sítě. Čím vyšší je toto číslo, tím více paměťového místa bude potřeba.

Tensorflow-wavenet

V této sekci je popis implementace tensorflow-wavenet, kterou jsem používal na experimenty.

Jedná se implementaci, která je open source s licencí (*MIT License (MIT) Copyright (c) 2016 Igor Babuschkin*). Jak bylo uvedeno, v licenci je to projekt

na githubu, který vytvořil Igor Babuschkin, ale na daném projektu pracuje více uživatelů. Tato konkrétní implementace využívá framework TensorFlow (viz sekce 3.3.2).V tensorflow-wavenet je implementovaný nejnovější rozšíření WaveNetu, které bylo uvedeno v kapitole 3.3.1.

Parametry. Ve WaveNetu existuje mnoho vstupních parametrů pro trénování sítě i pro vytváření výsledného audia v generování, proto zde nebudou uvedené všechny. Budou v této sekci uvedené parametry, které jsou pro WaveNet důležité, ale nebudou zde popsané parametry, které již byly zmíněné v sekci 3.3.2.

Společný parametr pro tréninkovou fázi a fázi generování je načtení JSONu, který může obsahovat mnoho parametrů, ale jako defaultní počet parametrů je 10. Parametry vně JSONu jsou parametry především pro tvorbu neuronové sítě. Budou zde uvedené atributy, které byly nějakým způsobem upravené v některém z experimentů:

- Sample rate (vzorkovací frekvence) - jedná se o celočíselnou hodnotu, která udává počet vzorků za jednotku času (obvykle za 1s). Výchozí hodnota je 16000.
- Dilations (dilatace) - jedná se o list celočíselných hodnot, které určují dilatační faktor pro každou vrstvu. Dilatace 8 je znázorněná na obrázku 3.11. Na zmíněném obrázku je dobře vidět, s kolika prvky pracuje určitá vrstva s jistou dilatací.
- Residual channels (reziduální kanály) - jedná se o celočíselnou hodnotu, jenž znázorňuje počet kanálů/filtrů reziduálního výstupu, což je počet výstupních kanálů levého horního konce 1x1 a počátečního kauzálního konvoluce (viz obrázek 3.10).
- Dilation channels (dilatační kanály) - jedná se o celočíselnou hodnotu, která udává počet výstupních kanálů/filtrů dilatační konvoluce. Neboli kolik kanálů/filtrů se dozví o rozšíření. Na obrázku 3.10 je zmíněná věc znázorněna blokem Dilated conv.
- Quantization channels (kvantizační kanály) - jedná se o celočíselnou hodnotu, jenž reprezentuje počet amplitudových hodnot, které jsou použité pro kvantizaci zvuku, což odpovídá jednorázovému kódování. Výchozí hodnota je 256 (8bitová kvantizace). Na obrázku 3.10 je tato věc reprezentovaná blokem Causal conv.
- Skip channels (vynechávané kanály) - jedná se o celočíselnou hodnotu, která znázorňuje počet výstupních kanálů/filtrů první 1x1 konvoluce vpravo (viz obrázek 3.10). Neboli tyto filtry/kanály přispívají ke kvantovanému výstupu softmaxu.
- Use biases (Použití odchylky) - jedná se o binární hodnotu, která určuje, zda má každá konvoluce přidat vrstvu zaujatosti/odchylky. Doporučená hodnota je False.

Když uživatel spouští tréninkovou fázi programu, která spouští takto: `python train.py`, může k ní přidat parametry, které mohou pomoci k lepšímu výsledku. Parametry se mohou přidat např.: `python train.py -data_dir=cesta/k/datasetu`. Je možnost nastavit až 17 atributů, ale popsané bude 6 z nich (důvod je uvedený výše):

- Data dir (Adresář s daty) - jedná se o řetězovou hodnotu, která reprezentuje adresář, ve kterém jsou vstupní data pro trénování NN. Pro tuto práci jsou datasety více rozebrané v sekci 4.2.
- Logdir (Adresář pro logy) - jedná se o řetězovou hodnotu, která slouží k určení adresáře, do kterého chceme uložit informace o protokolování pro TensorBoard. V případě, že ve složce již existuje model, načte se a obnoví. Poté se bude pokračovat dále v tréninku. Pro generování slouží adresář ke stejnému účelu, ale neobnovuje model. Tedy pokaždé začne generování od začátku.
- Checkpoint every (frekvence vytváření checkpointu) - jedná se o celočíselnou hodnotu, která určuje po kolika krocích se vytvoří checkpoint pro trénovaný model. Defaultní hodnota je 50 kroků.
- Sample size (Velikost vzorku) - je to celočíselná hodnota, jenž reprezentuje, po jaké délce se budou řezat vzorky. S menšími vzorky NN se rychleji učí. Výchozí hodnota je 100000.
- Silence threshold (Prah ticha) - jedná se o číselnou hodnotu, která reprezentuje prahovou hodnotu hlasitosti. Tento atribut určuje, co je považováno za ticho, neboli, pokud je nějaká vstupní hodnota pod touto hodnotou je považovaná za ticho (tedy 0). Defaultně má tento parametr nastavenou hodnotu na 0,3.
- Max checkpoints (Maximální počet checkpointu) - jedná se o celočíselnou hodnotu, jenž označuje maximální počet checkpointu, které nebudou smazané. Výchozí hodnota je 5.

Také při spouštění generovací fáze, která spouští `python generate.py`, se mohou přidat atributy, jež mohou vylepšit výsledek. Počet možných parametrů je 12, ale budou přiblíženy pouze tyto 4 z nich (důvod je uvedený výše):

- Checkpoint - je to řetězová hodnota, která určuje, se kterým naučeným modelem se bude pracovat, neboli z jakého se bude generovat výsledek. Tento atribut je povinný a nemá defaultní hodnotu. Jako jediný parametr se vypluje např.: `python generate.py logdir/train/model.ckpt-80000`, tedy nedává se před hodnotu žádné pojmenování atributu.
- Samples (počet vzorků) - je to celočíselná hodnota, jenž reprezentuje počet vzorků k vytvoření výsledku. Výchozí hodnota je 16000.

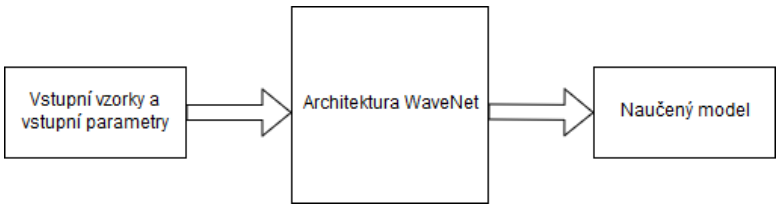
- Wav out path (Cesta k výstupu souboru) - je to řetězová hodnota, která určuje název a cestu k výstupnímu souboru. Výstupní soubor je typu WAV.
- Wav seed (Počáteční vzorek) - je to řetězová hodnota, která reprezentuje cestu k WAV souboru. Tento soubor se používá jako pomocný soubor ke generování výstupu, neboli jedná se o počátek, na který naváže generování NN. Tedy NN díky tomu může vygenerovat delší zvuk než bez tohoto parametru.

NSynth

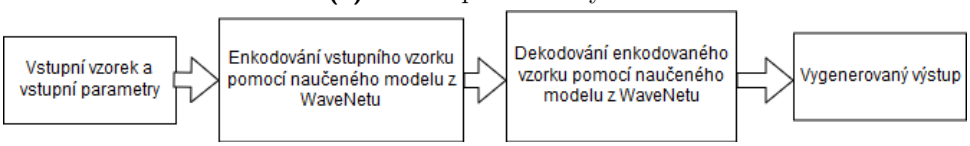
V této sekci je možné najít popis implementace NSynth a její použití k experimentům.

Jedná se o implementaci, která využívá WaveNet jako autoencoder. Tato architektura se jmenuje Magenta (konkrétně NSynth) [41]. NSynth je implementace, která je open source s licencí (*Copyright 2016 The Magenta Team*). Jak bylo uvedeno, v licenci je to projekt na githubu, který vytvořil tým Magenta, pro který pracují zaměstnanci společnosti Google. Tato konkretní implementace využívá framework TensorFlow (viz sekce 3.3.2). V NSynth je implementováno nejnovější rozšíření WaveNetu, které bylo uvedeno v kapitole 3.3.1.

NSynth pracuje tak, že se nejprve naučí model podobně jako u tensorflow-wavenet (viz sekce 3.3.2), ale generování funguje jako MNIST autoencoderu (viz sekce 3.3.2). Aby to bylo uvedené na pravou míru, k trénování NN je potřeba vytvořit dataset, který se skládá ze souborů typu WAV a souboru typu JSON, v němž jsou popsané všechny WAV soubory pomocí parametrů. Tyto parametry mohou být např.: instrument (nástroj), sample rate (vzorkovací frekvence), atd. Díky JSON souboru je možné, aby mohl být vytvořený dataset, který nebude obsahovat pouze jeden typ zvuků. Pomocí tohoto datasetu a vstupních parametrů je možné naučit WaveNet. Po dokončení učení WaveNetu je možné přistoupit ke generování, kde je potřeba nastavit vstupní atributy a vstupní vzorek. Generování enkóduje vstupní vzorek pomocí naučeného modelu WaveNetu a následovně dekóduje enkódovaný vzorek. Díky tomu se vygeneruje výsledek podobný vstupnímu vzorku (viz obrázek 3.14).



(a) : Princip učení NSynth



(b) : Princip generování NSynth

Obrázek 3.14: Princip WaveNetu jako autoenkodér (NSynth)

Za zmínku stojí, že autoři této implementace upozorňují na to, že vytvoření vlastního datasetu pro NSynth a naučení dané architektury je velmi složité. Proto se autoři rozhodli zpřístupnit dataset a naučený model. Dataset se rozděluje na 3 datasety, které jsou: trénovací, validační a testovací, a obsahuje celkem 305 979 vzorků. Nejzajímavější dataset je trénovací, který obsahuje 289 205 vzorků a 10 různých nástrojů, mezi které patří např.: kytara, flétna, atd. Model je naučený pomocí trénovacího datasetu a tento model se učil na 200 000 trénovacích kroků. Autoři také uvádí, abychom využili jejich model bez dalšího trénování nebo jako tzv. pre-trained model [46], což znamená, že model byl již na nějakém datasetu natrénovaný. Pokud se model použije jako pre-trained, měl by se doučit na vlastních vstupních datech.

Kapitola 4

Příprava k experimentům

Tato kapitola přiblíží všechny implementace, které jsem provedl za účelem praktického pochopení WaveNetu (viz sekce 3.3.1) a také proto, aby mohly sloužit jako reference k experimentům. Zde také bude uvedené, jak jsem vytvořil datasety pro tensorflow-wavenet (viz sekce 3.3.2).

Pro všechny implementace jsem použil programovací jazyk Python, protože výhodou Pythonu je možnost snadného použití knihoven z jiných programovacích jazyků.

Cíle experimentů byly:

- Prozkoumání možností využití WaveNetu pro syntézu instancí stejného typu zvuku a ověření pomocí poslechových testů korelace vygenerovaného zvuku k původním zvukům.
- Prozkoumání možností využití WaveNetu pro syntézu dvou instancí stejného typu zvuku. Prozkoumání možností parametrizace výstupu, zejména možnost deterministického ovlivňování výsledků uživatelem. Ověřením pomocí poslechových testů zjistit, zdali participanti rozpoznají obě původní instance ve vygenerovaném zvuku.

Výsledky cílů budou znázorněné pomocí procent v tabulkách (viz kapitola 5) např.: kolik procent participantů slyšelo správný hudební nástroj.

4.1 Implementace MNIST autoencoderu

V této sekci je uvedená moje implementace MNIST autoencoderu (viz sekce 3.3.2).

Chtěl jsem vytvořit nejjednodušší implementaci s NN. Jako nejlepší příklad mi přišel MNIST autoencoder, protože k této implementaci existuje mnoho tutoriálů. Proto jsem postupoval podle tutoriálu a snažil jsem se co nejvíce

komentovat kód pro lepší pochopení autoencoderu. K implementaci jsem použil TensorFlow (viz sekce 3.3.2).

Protože je tato implementace velmi rozřazená má TensorFlow v sobě zabudované stahovaní MNIST dat, které stahuje pomocí importů (`from tensorflow.examples.tutorials.mnist import input_data`). Po spuštění implementace uloží MNIST data do adresáře */tmp/data/*. Pokud existují MNIST data v adresáři */tmp/data/*, /, jsou načtená do programu. Pokud neexistují MNIST data v adresáři */tmp/data/*, je vyhozená chyba a aplikace se ukončí.

V sekci (viz sekce 3.3.2) byly uvedené parametry pro trénink, ale zde popíšu parametry, které jsem použil pro tvorbu NN v této implementaci: Velikost vstupní/výstupní vrstvy, velikosti pro skryté vrstvy, váhy, bias. Tyto parametry byly popsány v kapitole 3.2. Z těchto parametrů je možné vytvořit vrstvy, které využívají sigmoidální přenosovou funkci, implementace obsahuje dvě pro enkodér i dekodér. Příklad 1. vrstvy enkodéru: `tf.nn.sigmoid(tf.add(tf.matmul(x, weights['w1']), biases['b1']))`

Následně definuji dvě nejdůležitější proměnné pro NN, což jsou: loss, optimizer (viz fragment kódu 4.1). Loss reprezentuje ztrátovou funkci, která funguje tak, že od vstupní vrstvy se odečítá výstupní vrstva. Té se také říká predikce. Poté se umocní všechny elementy na druhou a následně se z nich provede průměr. Optimalizér definuje, jak se bude provádět optimalizace v tréninku sítě. Zde je vytvořený jako minimalizování ztrátové funkce.

```
loss = tf.reduce_mean(tf. pow(x - layer_decoder2, 2))
optimizer = tf.train.RMSPropOptimizer(learning_rate)
                        .minimize(loss)
```

Fragment kódu 4.1: ztrátová funkce a optimalizace pro MNIST

V tréninkové fázi je důležité vědět, jak se optimalizace ztrátové funkce provádí, což je vidět v fragmentu kódu 4.2. `feed_dict` slouží k nastavení placeholderu `x` na hodnotu proměnné `batch_x`. Nasledně se bude minimalizovat ztrátová funkce, která bude vypadat takto `tf.reduce_mean(tf.pow(batch_x - predicate_batch_x, 2))`

```
_, l = sess.run([optimizer, loss], feed_dict={X: batch_x})
```

Fragment kódu 4.2: Optimalizace ztrátové funkce pro MNIST

Po tréninkové fázi, která skončí tehdy, pokud je hodnota atributu `num steps` rovna celočíselné hodnotě a jež byla zadána na vstupu programu, se provádí testovací fáze, která komprimuje a dekomprimuje MNIST data z testovacího data setu. Následně sestaví pomocí knihovny matplotlib.pyplot dva výstupní obrázky, které jsou originální obrázek a dekomprimovaný obrázek.

4.2 Vytvoření datasetů pro tensorflow-wavenet

V této podkapitole je přiblíženo Vytvoření datasetů pro tensorflow-wavenet (viz sekce 3.3.2). Tato sekce je nejdůležitější pro experimenty, protože níže popsané datasety jsou využívaná jako jeden ze vstupních parametrů pro experimenty.

Nalezení správného datasetu pro tensorflow-wavenet byl jedním z nejtěžších úkolů pro danou architekturu. Tensorflow-wavenet je vytvořený tak, aby pracoval s delšími vstupními vzorky, které jsou dlouhé minimálně 30s, a právě proto je vhodný pro zpracování řeči a skladeb. U skladeb je to trochu složitější, protože tensorflow-wavenet je vhodné učit pouze jedním hudebním nástrojem např. pianem, než všemi hudebními nástroji, které by se v dané skladbě nacházely.

Když jsem vytvářel datasety, nepovedly se mi vytvořit napoprvé, ale až na několikátý pokus. Jeden z důvodů byl, že jsem neměl dostatečně mnoho zvuků na učení. Vytvořené datasety obsahují vzorky bubnů (viz sekce 2). Tedy vytvořil jsem 3 datasety:

- Bass drums - jedná se o dataset, který je tvořen 135 různými vzorky Bass drumů. Všechny vzorky jsou ve formátu WAV a jsou dlouhé 0,16s až 3,1s.
- Snare drums - tento dataset obsahuje 166 různých vzorků typu snare drum. Všechny vzorky jsou ve formátu WAV a jsou dlouhé 0,165s až 2s.
- HiHats - tento dataset je tvořen 127 různými vzorky typu hihat. Všechny vzorky jsou ve formátu WAV a jsou dlouhé 0,053s až 1,54s.

Důležitá informace je, že jsou všechny vzorky, které jsou použité v datasetech výše, bez autorských práv, neboli jsou volně dostupné a šiřitelné.

Zásadní informace, která mi dlouho dobu unikala, je ta, že pokud chceme vygenerovat vzorek dlouhý 0,5s a velmi podobny vstupním vzorkům, musí být velikost vstupních vzorků minimálně 5s, ale pokud bude delší, bude to jen k prospěchu výsledku. Důležité je také, jak hlasitá jsou vstupní data. Pokud jsou tichá, může se stát, že se bude pracovat jen s tichem (viz sekce 5.6).

Kapitola 5

Experimenty a uživatelské testování

V této kapitole jsou popsané experimenty, které jsem prováděl s WaveNetem. Ke každému experimentu bylo provedeno uživatelské testování (poslechové testy). V následujících podkapitolách bude popsaná skupina participantů, která se účastnila experimentů, a dále jednotlivé části testování a výsledky.

Experimenty byly provedeny na školním serveru (147.32.81.78), kde se nacházejí pouze CPU a žádné GPU. Na tomto serveru také probíhaly dvě třetiny generování experimentu a jedna třetina generování probíhala na mém notebooku DELL Vostro 5000, který má pouze integrovanou grafiku, tzn. že generování probíhalo pouze na CPU.

K ověření dosažení cílů (viz sekce 4) jsme využili poslechových testů, během kterých měli uživatelé za úkol:

- Určit nástroj, který byl syntetizován (viz sekce 5.4.1)
- Určit typ nástroje, který byl syntetizován (viz sekce 5.4.1). Tedy určit, zda se jedná o snare drum, bass drum, nebo hithat.
- Určit, z kolika původních zvuků byl syntetizován výsledný zvuk (viz sekce 5.5.1).
- Určit, z jakých původních vzorků byl syntetizován výsledný zvuk (viz sekce 5.5.1).

5.1 Participanti

Při výběru participantů jsem použil metodu automatického výběru. Zajištění, že participant je vhodným kandidátem pro uživatelské testování, bylo provedeno pomocí pre-testu (viz sekce 4.2). Testovací skupina pro každý experiment byla tvořená z 20 participantů, kteří byli v poměru 60% muži a 40% ženy. Participanti spadají do věkové hranice 15 - 50 let. Žádný z členů netrpí žádnou sluchovou vadou. V testovací skupině se nacházejí participanti,

kteří nejčastěji poslouchají nejen elektronikou hudbu, ale i jiný druh hudby. Konkrétně v poměru 65% participantů nejčastěji poslouchá elektronickou hudbu a 35% poslouchá i jiný druh hudby než elektronickou. 25% participantů hraje na nějaký hudební nastroj a 75% nehraje na žádný hudební nástroj. 70% participantů bylo ochotno provést uživatelské testování na jiném zařízení než na svém a 30% chtělo striktně využívat pouze své zařízení k testování, což vedlo k tomu, že někteří participanti byli testováni vzdáleně pod mým vedením přes určitý komunikační nástroj. Za zmínku stojí, že někteří z těchto participantů se zúčastnili více testů.

5.2 Pre-test

Pre-test se používá k ověření informací participanta nebo k doplnění informací o participantovi. Získané informace z pre-testu pomáhají k lepší analýze participanta. Pre-test se skládá z následujících otázek:

1. Jaký je váš věk?
2. Máte nějakou sluchovou vadu?
3. Jaký druh hudby nejčastěji posloucháte?
4. Hrajete na nějaký hudební nástroj?
5. Jste ochotní poslouchat hudbu na jiném zařízení než na vašem?

5.3 Post-test

Po dokončení všech úkolů (nebo po uplynutí času) byl požádán o vyplnění post-test. Tento dotazník je určen k zjištění toho, s čím účastník byl nebo nebyl spokojen. Dotazník se skládá z následujících otázek.

1. Co se Vám líbilo při testování?
2. Co se Vám nelíbilo při testování?
3. Jak Vám přišli jednotlivé části testování složité a proč?

5.4 Experimenty se syntézou různých typu zvuku

V této kapitole budou popsané experimenty, jenž byly prováděny s architekturou tensorflow-wavenet (viz sekce 3.3.2) a různými datasety (viz sekce 4.2),

které byly použité vždy odděleně. Také zde budou vygenerované zvuky z jednotlivých experimentů a výsledky poslechových testování.

Experimenty vychází z předchozího experimentu. Tím je myšleno, že vzorky v experimentu jsou vygenerovány vždy se všemi změnami, které pomohly vylepšit dané vzorky z předchozích experimentů. Vzorky navíc obsahují další změny pro dané testování.

Uživatelské testování bylo totožné pro všechny experimenty, které jsou uvedené v této podkapitole, proto v jednotlivcích experimentech jsou uvedené jen výsledky testování.

5.4.1 Uživatelské testovaní experimentů

Uživatelské testování experimentů je rozděleno do následujících částí, které jsou níže popsané.

Ice-breaking (Prolomení ledu)

Před provedením testování byl participant uvítán moderátorem. Moderátor seznámil participanta s postupem testování. Byla uvedená základní instrukce, která je, že se testují pouze zvuky nikoliv účastník.

Plnění úkolů

Hlavní činností účastníků je naplnění připraveného seznamu úkolů. Testování je rozděleno na dvě části, ale druhá část se vůbec vykonávat nemusí. Protože druhá část je závislá na první části.

První část se prováděla v domácím prostředí participanta. Participant byl veden osobně nebo přes komunikační zařízení moderátorem. Úkolem participanta bylo poslechnout si 15 - 40 zvuků a určit, jestli v daných zvucích slyší nějaký hudební nástroj.

Druhá část se nemusela provádět, pokud participant v první části neslyšel žádný buben. Pokud účastník slyšel buben v některém zvuku z první části, měl za úkol poslechnout si daný zvuk a určit k němu vzorky, který znějí podobně jako vygenerovaný, z šesti původních vzorků (2 bass drums, 2 snare drums a 2 hihats). Počet nebyl omezený.

Testovací scénáře

Pro participanty, kteří byli testováni osobně. Participantům pouštěl zvuky moderátor, proto zde není uvedené, jak si dané zvuky mají participanti pouštět.

1. část.

1. Pokud v ukázce zvuků uslyšíte nějaký hudební nástroj, zapište jeho název do tabulky. Tento úkol se aplikuje na všechny zvuky, které uslyšíte.

2. část.

1. Vyberte zvuky, které jsou podobné následujícímu zvuku. Můžete vybrat 0 - 6 zvuků. Tento úkol se může provádět několikrát.

Pro participanty, kteří byli testováni vzdáleně. Participanti si pouštěli zvuky sami, proto zde je uvedeno, jak si dané zvuky mají participanti pouštět. Moderátor je pouze vedl.

1. část.

1. Ujistěte se, že jste v klidném prostředí.
2. Nastavte si prostředí (reproduktory či sluchátka) tak, jak rádi hudbu posloucháte.
3. Poslechněte si po jednom zvuky a do tabulky napište název hudebního nástroje, pokud nějaký uslyšíte. Tento úkol proveďte se všemi zvuky, které jste dostal.

2. část.

1. Opět se ujistěte, že jste v klidném prostředí.
2. Opět se ujistěte, že máte nastavené prostředí (reproduktory či sluchátka) na pro vás vhodnou hlasitost hudby.
3. Nejprve si poslechnete 1. zvuk v 1. složce.
4. Poslechněte si všechny zvuky, které jsou v 1. složce. Určete, jaké zvuky jsou podobné 1. zvuku. Můžete vybrat 0 - 6 zvuků.
5. Pokud máte více složek, udělejte body 3 a 4 na ostatních složkách. Např.: u 2. složky si nejprve přehrajte 1. zvuk, atd.

5.4.2 Experiment 1

Jsem použil datasety (viz sekce 4.2) a nastavení téměř všech parametrů jsem nechal, aby se spouštělo s výchozími hodnotami. Změnil jsem hodnotu parametru sample rate na 44100, protože uvedenou vzorkovací frekvenci mají všechny vzorky, které jsou v datasetech. Při spouštění trénovací fáze tensorflow-wavenet jsem změnil dva atributy: data dir a num steps.

Ukážu na obecném příkladu: při pouštění trénování NN: `python train.py -data_dir=cesta/k/datasetu -num_steps=90000`. Počet trénovací kroků byl pro všechny datasety stejný.

Učení NN jsem pustil třikrát pro každý dataset. Následně jsem pustil generování NN, které se snaží sestavit zvuk z naučených modelů. Generovaní se pouští takto: `python generate.py wav_out=nazev.wav cesta/k/naučenému/modelu`

5.4.3 Experiment 2

Při učení předešlého experimentu se projevila jedna chyba, která by mohla způsobit, že výstupem generování je šum. Domnívám se, že tento výstup je způsoben tím, že mnoho vzorku z datasetů bylo považováno za ticho. Tedy bylo nutné změnit hodnotu parametru silence threshold, protože výchozí hodnota parametru je 0,3, což ve velmi mnoho pro zvuky, které mám v datasetech. Dovedlo mě to k postupnému snižování hodnoty atributu až 0,01. Postupné snižování mělo hodnoty 0,2, 0,15, 0,1, 0,08, 0,05, 0,03 a 0,01.

Pro každou změnu hodnoty atributu byla provedena dvě učení tensorflow-wavenet pro každý dataset. Následně jsem pustil generování NN, které se snaží sestavit zvuk z naučených modelu.

Před výběrem jsem prošel kompletně vygenerované zvuky, tím je myšleno vizuálně i poslechově. Z poslechové stránky se téměř nelišily od experimentu 1, ale z vizuální stránky můžu jednoznačně říci, že hodnoty atributu, které jsou 0,2, 0,15 a 0,1 měly totožné výstupy jako u experimentu 1. Naopak hodnoty parametru, které jsou 0,08, 0,05, 0,03 a 0,01 měly lehké náznaky korelace k předlohám (Viz obrázek 5.3).

5.4.4 Experiment 3

Změny v předešlém experimentu přispěly k vylepšení výsledného zvuku, ale stále ve výsledku byl slyšet šum. Tedy tento experiment se zaměřuje na odstranění ticha na začátku a šumu ve výsledném zvuku, což vedlo k úpravě kódu.

Nejdříve jsem chtěl pochopit, jak vzniká výstup. V open source implementaci je tvořen náhodným výběrem vzorků, které jsou v rozmezí 0-255, vzhledem k pravděpodobnosti predikci (viz fragment kódu 5.1)

```
sample = np.random.choice(np.arange(quantization_channels),
          p=scaled_prediction)
```

Fragment kódu 5.1: Vytváření pomocí náhodného výběru vzorků vzhledem k pravděpodobnosti predikci

Ticho bylo vytvořeno před generováním z modelů. Ticho bylo dlouhé 5200 vzorků (viz fragment kódu5.2). Ticho na začátku nebylo k ničemu přínosné.

```
waveform = [quantization_channels/2] * (net.receptive_field-1)
waveform.append(np.random.randint(quantization_channels))
```

Fragment kódu 5.2: Generování ticha

Začal jsem upravovat část kódu (viz fragment kódu 5.1), která se nachází v souboru `generate.py`, který slouží ke generování výstupu. Každou změnu kódu jsem otestoval na doposud nejlépe naučeném modelu, který je z předešlého experimentu. Konkrétně model, který byl naučený z datasetu bass drums a měl nastavenou hodnotu silence threshold na 0,05.

Odstranit šumu se mi povedlo udělat s použitím `np.argmax` (viz fragment kódu 5.3) místo `np.random.choice`. Argmax vrací indexy maximálních hodnot.

```
sample = np.argmax(prediction)
```

Fragment kódu 5.3: Vytváření pomocí argmax

Vymazaní ticha ve výsledném zvuku jsem udělal tak, že jsem nahradil jsem tento fragment 5.2 za jeden řádek kódu `waveform = [.5]`.

Výhoda tohoto experimentu je, že nemusel být prováděno žádné učení. Byly použité ke generování NN modely z předchozího experimentu. Tedy byly použitý modely, který byly naučený s nastavenou hodnotu silence threshold na 0,05. Modely byli ve stejném počtu pro všechny datasety (viz 4.2)

5.4.5 Experiment 4

V tomto experimentu jsem měnil mnoho aspektů pro trénování NN, ale většina vedla ke zhoršení výsledku z generování NN. Např. se objevil ve výsledku šum, ale dvě úpravy vedly ke zlepšení výsledného zvuku:

- Úprava hodnoty atributu quantization channels - úprava spočívala v tom, že jsem upravil hodnotu parametru z 256 na 512.
- Rozšíření datasetů - rozšíření jsem provedl tak, že jsem datasety rozkopíroval 8x. Jinými slovy: Pokud bylo v bass drums datasetu (viz sekce 4.2) 135 vzorku, po provedení rozšíření bylo 1080 vzorků v daném datasetu.

V součtu jsem provedl 10 změn. Na každé změně byla trénovaná tensorflow-wavenet dvakrát pro dataset bass drums. Následně byla puštěné generování tensorflow-wavenet, kde jsem si ověřil, které změny vylepšily výsledný zvuk. Poté jsem výše uvedené změny spojil a naučil jsem na těchto změnách NN pro každý dataset, Následně jsem pustil generování NN, které se snaží sestavit zvuk z naučených modelu.

5.4.6 Experiment 5

V experimentu jsem chtěl docílit co nejmenší hodnotu ztrátové funkce, protože pokud by se hodnota ztrátové funkce rovnala téměř nule, mohla by teoreticky architektura tensorflow-wavenet vygenerovat zvuk, který by byl téměř shodný s originálem. Dosáhne se toho tak, že bude měnit hodnotu atributu learning rate.

Z předchozích experimentů jsem zjistil, že hodnota ztrátové funkce při učení NN se nemůže dostat pod určitou hodnotu. Např.: výsledek ztrátové funkce pro dataset bass drums a výchozí hodnotu atributu learning rate, která je 0,001, se nedostal pod hodnotou 0,35. Jinými slovy: pokud se výsledek ztrátové funkce přiblíží minimální možné hodnotě, je v dalším trénovacím kroku výsledek ztrátové funkce delegován na vyšší hodnotu. Tento problém je ukázaný na obrázku 5.1(a).

Hodnotu výše zmíněného atributu jsem upravil 2x:

- Snížení hodnoty atributu learning rate z 0,001 na 0,00001. Tuto změnu jsem testoval na 3 učeních a generovaní s datasetem bass drums. Ukázalo se, že bych učení muselo provádět minimálně dvakrát tak dlouho, než se provádělo s defaultní hodnotou. Jinými slovy: abych měl stejně dobrý výsledek, jako před úpravou muselo by trénování tensorflow-wavenet běžet 160 000 tréninkových kroků, což je časově velmi nákladné.
- Snížení hodnoty atributu learning rate z 0,001 na 0,0001. Tuto úpravu jsem testoval na 3 učeních a generovaní s datasetem bass drums. Zjistil jsem, že po 99 000 tréninkových kroků byla hodnota ztrátové funkce pod hodnotou 0,12, ale trénování tensorflow-wavenet bylo časově složitější.

Proto po těchto úpravách jsem vrátil zpět výchozí hodnotu atributu, ale chtěl jsem danou hodnotu měnit programátorsky/automaticky. Po 10 úpravách v kódu v souboru `train.py` a po 2 učeních a generovaní s datasetem bass drums ke každé změně se výsledek ztrátové funkce dostal pod hodnotu 0,1. Ve fragmentu kódu 5.4 jsou ukázané všechny změny, které jsou rozdělené do tří bloků:

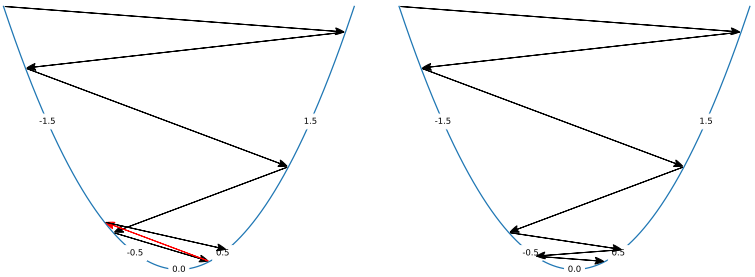
- První blok je od proměnné loss až do proměnné train_op. Tento blok se může označovat i jako inicializační, protože zde se inicializují proměnné pro NN architektury tensorflow-wavenet. V daném bloku jsem přidal `learning_rate_placeholder`, který slouží jako placeholder pro proměnou `learning_rate`. Změnil jsem i optimalizer, protože se mi s tímto optimalizerem lépe pracovalo a měl lepší funkce pro placeholder.
- Druhý blok je pouze jeden řádek, kde je proměnná summary. Tento blok se nachází v hlavní části, která slouží pro naučení tensorflow-wavenet. Daný blok je důležitý kvůli `feed_dict=learning_rate_placeholder: learning_rate`. Feed_dict je mechanizmus, který slouží pro doplnění hodnoty za placeholder.

- Třetí blok se skládá z podmínek. Tento blok se nachází v hlavní části, která slouží pro trénování NN. Tyto podmínky říkají přesně, kdy se proměnná `learning_rate` může změnit. Neboli pokud je hodnota ztrátové funkce mezi 0,5 a 1,5 a hodnota již nebyla měněná, tak se proměnná `learning_rate` vynásobí 0,1, což vede k tomu, že hodnota atributu learning rate se sníží na 0,0001. Hodnota proměnné `learning_rate` se změní podruhé, pokud je hodnota ztrátové funkce pod 0,5 a hodnota nebyla měněná víckrát než jednou, tedy hodnota atributu learning rate se sníží na 0,00001 díky vynásobení číslem 0,1 předešlé hodnoty daného parametru.

```
loss = net.loss(input_batch=audio_batch,
      global_condition_batch=gc_id_batch,
      l2_regularization_strength=args.l2_regularization_strength)

learning_rate_placeholder = tf.placeholder(tf.float32, [])
optimizer = tf.train.RMSPropOptimizer(
            learning_rate=learning_rate_placeholder,
            momentum=args.momentum)

train_op = optimizer.minimize(loss)

.
.
.

summary, loss_value, _ = sess.run([summaries, loss, train_op],
          feed_dict={learning_rate_placeholder: learning_rate})

.
.
.

if 1.5 >= loss_value > 0.5 and update == 0:
   learning_rate = learning_rate * 0.1
   update += 1
elif loss_value <= 0.5 and update == 1:
   learning_rate = learning_rate * 0.1
   update += 1
```

Fragment kódu 5.4: Automatické měnění parametru learning rate

Výše uvedené změny u proměnné `learning_rate` vedou k tomu, že se ztrátová funkce může přiblížit velmi blízko 0 (viz Obrázek 5.1(b)).



(a) : S výchozí hodnotou atributu learning_rate. Šipky znázorňují minimalizaci ztrátové funkce.

(b) : S měnícím se hodnotou atributu learning_rate. Šipky znázorňují minimalizaci ztrátové funkce.

Obrázek 5.1: Zjednodušené znázornění učení WaveNetu

Pro automatickou změnu hodnoty atributu byla provedena dvě učení tensorflow-wavenet pro každý dataset. Následně jsem pustil generování NN, které se snaží sestavit zvuk z naučených modelů.

5.4.7 Experiment 6

Cíl tohoto experimentu byl vytvořit stejnou délku pro všechny vstupních vzorky datasetu, který byl použit pro učení NN.

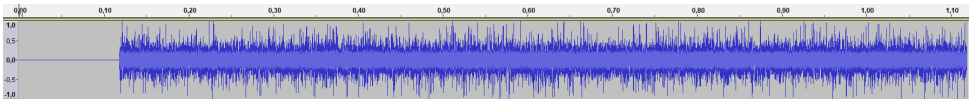
To jsem provedl tak, že jsem si načetl všechny vzorky a následně jsem zjistil, který vzorek je nejdelší. Poté jsem porovnával nejdelší vzorky a vzorky, které byly kratší než nejdelší vzorek, jsem dovyplnil tichem. Neboli jsem kratší vzorky dovyplnil nulami.

Pro tuto úpravu byly provedeny 3 učení tensorflow-wavenet pro každý dataset. Následně jsem pustil generování NN, které se snaží sestavit zvuk z naučených modelu.

5.4.8 Vizualizace vygenerovaných experimentů

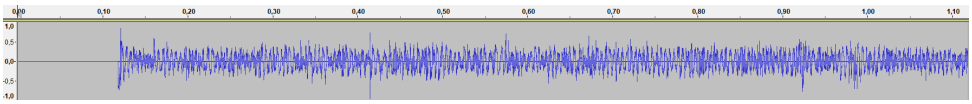
V této sekci bude vizualizace jednoho vygenerovaného zvuku z každého experimentů, které byly popsané výše (viz sekce 5.4).

Z experimentů 1 a 2 jsou zde uvedené výsledné zvuky (Viz obrázek 5.2 a 5.3), které byly vygenerované z datasetu snare drums. Mají společné rysy. Oba zvuky jsou dlouhé 1.119s, z toho 0,119s je pouze ticho na začátku a 1s reprezentují náhodné hodnoty. Jinými slovy jedná se o zvuk, který se skládá z ticha a šumu.



Obrázek 5.2: Amplituda v čase pro vygenerovaný zvuk z experimentu 1

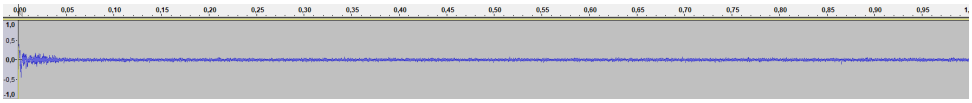
Výsledný zvuk z experimentu 1 (Viz obrázek 5.2) nekoreluje k vstupním zvukům (viz obrázek 2.10) z datasetu snare drums. Tento problém se vyskytl u všech použitých datasetů.



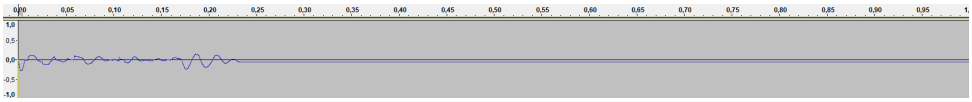
Obrázek 5.3: Amplituda v čase pro vygenerovaný zvuk z experimentu 2

Syntetizovaný zvuk z experimentu 2 (Viz obrázek 5.3) byl vygenerovaný s hodnotou parametru silence threshold nastavenou na 0,05. Díky vizualizaci je patrná menší korelace ke vstupním zvukům (Viz obrázek 2.10) z datasetu snare drums.

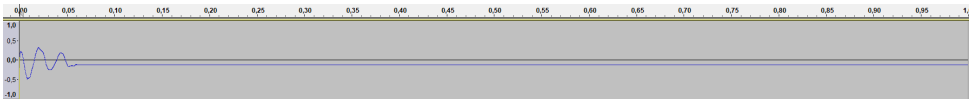
Z experimentů 3, 4, 5 a 6 jsou v této sekci vizualizované výsledné zvuky (Viz obrázek 5.4, 5.5, 5.6 a 5.6), který byly vygenerované z datasetu bass drums. Všechny uvedené zvuky jsou dlouhé 1s. Z obrázků je patrné, že alespoň mírně korelují všechny vygenerované zvuky ke vstupním zvukům (viz obrázku 2.8) z datasetu bass drums.



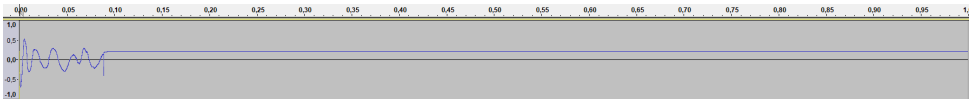
Obrázek 5.4: Amplituda v čase pro vygenerovaný zvuk z experimentu 3



Obrázek 5.5: Amplituda v čase pro vygenerovaný zvuk z experimentu 4



Obrázek 5.6: Amplituda v čase pro vygenerovaný zvuk z experimentu 5



Obrázek 5.7: Amplituda v čase pro vygenerovaný zvuk z experimentu 6

Pouze zvuky vygenerované z datasetu hihats nekorelovaly se vstupními zvuky, protože vygenerované zvuky byly vizualizované jako šum, nebo jako

ticho. Jinými slovy: vizualizace vygenerovaného zvuku z datasetu hihat byla rovná čára, nebo velmi podobná obrázku 2.10.

5.4.9 Výsledky experimentů

V této sekci budou uvedeny výsledky poslechových testů z každého experimentů, které byly popsané výše (viz sekce 5.4). Také zde bude popsán výběr dat pro jednotlivé experimenty.

Výběr testovacích dat

Výběr testovacích dat do uživatelského testovaní bylo u některých experimentů podobné, proto výběr testováních dat není uvedený v sekcích s experimenty.

Pro všechny experimenty platí, že vybrané zvuky do testovaní jsou v poměru 1/3 bass drums, 1/3 snare drums a 1/3 hihats z celkového počtu. Všechny zvuky byly vygenerované z naučených modelů, které jsou v rozmezí 80 000 až 90 000 trénovacích kroků.

Experiment 1. Do uživatelského testování bylo vybráno 15 zvuků, protože jsem chtěl zjistit, jak moc náročné bude testování pro participanty. Od tohoto experimentu se odvíjí úprava testovacích setů pro následující experimenty

Experiment 2. Do uživatelského testování bylo vybráno 36 zvuků. Tyto skupiny se dále dělí 3 zvuky pro hodnoty parametru silence threshold, které jsou 0,08, 0,05, 0,03 a 0,01. Bylo by vhodné, kdybych vybral více testovací dat pro zpřesnění výsledků, ale mělo by to za následek vzrůst náročnosti poslechových testů (viz sekce 5.4.10).

Experiment 3. Do uživatelského testování bylo vybráno 18 zvuků, protože se osvědčilo z předchozích dvou experimentů. 36 zvuků bylo pro participanty příliš náročné. Také jsem chtěl zjistit, jak náročné bude testování pro participanty, když ve zvuku neuslyší šum.

Experiment 4, 5 a 6. Do uživatelského testování bylo vybráno 24 zvuků, protože se ukázalo z experimentu 3, že testování pro participanty nebylo příliš náročné. Z experimentu 4, 5 a 6 se potvrdilo, že tento počet byl optimální.

Výsledky experimentů

Výsledky poslechových testování pro 1. část testovaní jsou znázorněné v tabulkách (viz tabulky 5.1, 5.2, 5.3, 5.4, 5.5, 5.7, 5.9 a 5.11). Procenta v tabulkách udávají počet participantů, kteří slyšeli správný hudební nástroj

(bubny). Tedy 0% znázorňuje, že nikdo neslyšel správný hudební nástroj a 100% znázorňuje, že všichni participanti slyšeli správný hudební nástroj.

Pokud alespoň jeden participant určil správně hudební nástroj z 1. části testování, absolvoval 2. část poslechových testů. K znázornění výsledků byly vytvořené tabulky (viz tabulky 5.6, 5.8, 5.10 a 5.12), kde procenta určují počet participantů, kteří správně přiřadili vygenerovaný zvuk k některému původnímu vzorku. Tyto výsledky se týkají participantů, kteří v 1. části slyšeli hudební nástroj. Např. pokud hudební nástroj správně slyšeli 2 participanti, 1 participant tzn. 50% správně přiřadilo vygenerovaný zvuk k některému původnímu vzorku.

Experiment 1. Výsledky 1. části poslechových testů pro experiment 1 jsou uvedené v tabulce 5.1. Z této tabulky je patrné, že nikdo nemusel dělat druhou část testu.

Zvuky	Bass drums	Snare drums	Hihat
steps-80000	0%	0%	0%
steps-83000	0%	0%	0%
steps-85000	0%	0%	0%
steps-87500	0%	0%	0%
steps-89999	0%	0%	0%

Tabulka 5.1: Tabulka popisující výsledky experimentu 1

Experiment 2. Výsledky 1. části poslechových testů pro experiment 2 jsou znázorněné v tabulkách 5.2, 5.3 a 5.4.

Zvuky z datasetu bass drums	silence threshold 0,08	silence threshold 0,05	silence threshold 0,03	silence threshold 0,01
steps-80000	0%	5%	0%	0%
steps-85000	0%	25%	10%	5%
steps-89999	0%	10%	5%	5%

Tabulka 5.2: Tabulka popisující výsledky experimentu 2 pro dataset bass drums

Z tabulky 5.2 je patrné, že několik participantů muselo absolvovat druhou část testu. Celkem jich bylo 7. Pouze 2 participanti ze 7 dokázali přiřadit vygenerovaný zvuk pomocí NN, která používala naučený model s dataset bass drums k některému původnímu bass drum. Toto přiřazení se uskutečnilo pouze v případě, kdy hodnota silence threshold byla nastavená na 0,05 a počet trénovacích kroků 85000.

Zvuky z datasetu snare drums	silence threshold 0,08	silence threshold 0,05	silence threshold 0,03	silence threshold 0,01
steps-80000	0%	0%	0%	0%
steps-85000	0%	10%	0%	0%
steps-89999	0%	5%	0%	0%

Tabulka 5.3: Tabulka popisující výsledky experimentu 2 dataset snare drums

Z tabulky 5.3 je patrné, že 2 participanti museli absolvovat druhou část testu. Bohužel žádný participant nedokázal přiřadit vygenerovaný zvuk pomocí NN, která používala naučený model s dataset snare drums k některému původnímu snare drum.

Zvuky z datasetu hihats	silence threshold 0,08	silence threshold 0,05	silence threshold 0,03	silence threshold 0,01
steps-80000	0%	0%	0%	0%
steps-85000	0%	0%	0%	0%
steps-89999	0%	0%	0%	0%

Tabulka 5.4: Tabulka popisující výsledky experimentu 2 dataset hihats

Z tabulky 5.4 je patrné, že nikdo nemusel dělat druhou část testu.

Experiment 3. Výsledky 1. části poslechových testů pro experiment 3 jsou uvedené v tabulce 5.5.

Zvuky	Bass drums	Snare drums	Hihat
steps-80000	5%	0%	0%
steps-83000	30%	5%	0%
steps-85000	30%	10%	0%
steps-86500	10%	0%	0%
steps-87500	10%	0%	0%
steps-89999	10%	5%	0%

Tabulka 5.5: Tabulka popisující výsledky 1. části experimentu 3

Z tabulky 5.5 je patrné, že několik participantů absolvovalo druhou část testu např. pro generování s počtem trénovacích kroků 85000 a s naučeným modelem z datasetu bass drums. Bylo jich 6 z 20. Výsledky 2. části poslechových testů jsou uvedené v tabulce 5.6.

Zvuky	Bass drums	Snare drums
steps-80000	0%	0%
steps-83000	16,7%	0%
steps-85000	33,3%	0%
steps-86500	0%	0%
steps-87500	0%	0%
steps-89999	50%	0%

Tabulka 5.6: Tabulka popisující výsledky 2. části experimentu 3

Experiment 4. Výsledky z 1. části poslechových testů pro experiment 4 jsou uvedené v tabulce 5.7.

Zvuky	Bass drums	Snare drums	Hihat
steps-80000	10%	0%	0%
steps-81000	15%	0%	0%
steps-82500	10%	0%	0%
steps-83000	35%	10%	0%
steps-85000	35%	10%	0%
steps-86500	15%	0%	0%
steps-88500	10%	0%	0%
steps-89999	15%	10%	0%

Tabulka 5.7: Tabulka popisující výsledky 1. části experimentu 4

Z výsledků vyplývá, že několik participantů absolvovalo druhou část testu. Výsledné vyhodnocení z 2. části testu je znázorněné v tabulce 5.8.

Zvuky	Bass drums	Snare drums
steps-80000	0%	0%
steps-81000	33,3%	0%
steps-82500	0%	0%
steps-83000	42,9%	50%
steps-85000	42,9%	50%
steps-86500	33,3%	0%
steps-88500	0%	0%
steps-89999	33,3%	0%

Tabulka 5.8: Tabulka popisující výsledky 2. části experimentu 4

Experiment 5. Výsledky 1. části poslechových testů pro experiment 5 jsou uvedené v tabulce 5.9.

Zvuky	Bass drums	Snare drums	Hihat
steps-80000	0%	0%	0%
steps-81000	20%	0%	0%
steps-82500	10%	0%	0%
steps-83000	40%	15%	0%
steps-85000	35%	15%	5%
steps-86500	5%	0%	0%
steps-88500	15%	0%	0%
steps-89999	30%	10%	5%

Tabulka 5.9: Tabulka popisující výsledky 1.části experimentu 5

Opět je patrné, že několik participantů absolvovalo druhou část testu. Výsledky z 2. části testu jsou znázorněné v tabulce 5.10.

Zvuky	Bass drums	Snare drums	Hihat
steps-80000	0%	0%	0%
steps-81000	25%	0%	0%
steps-82500	0%	0%	0%
steps-83000	50%	66,7%	0%
steps-85000	42,9%	33,3%	0%
steps-86500	0%	0%	0%
steps-88500	33,3%	0%	0%
steps-89999	33,3%	0%	0%

Tabulka 5.10: Tabulka popisující výsledky 2. části experimentu 5

Experiment 6. Výsledky 1. části poslechových testů pro experiment 6 jsou uvedené v tabulce 5.11.

Zvuky	Bass drums	Snare drums	Hihat
steps-80000	5%	0%	0%
steps-81000	0%	0%	0%
steps-82500	10%	5%	0%
steps-83000	55%	20%	0%
steps-85000	45%	15%	5%
steps-86500	20%	0%	0%
steps-88500	10%	0%	5%
steps-89999	30%	10%	5%

Tabulka 5.11: Tabulka popisující výsledky 1. části experimentu 6

Také v experimentu 6 absolvovalo několik participantů 2. část poslechových testu. Vyhodnocení 2. části testování je znázorněné v tabulce 5.12

Zvuky	Bass drums	Snare drums	Hihat
steps-80000	0%	0%	0%
steps-81000	0%	0%	0%
steps-82500	50%	0%	0%
steps-83000	72,7%	50%	0%
steps-85000	66,7,%	66,7%	0%
steps-86500	25%	0%	0%
steps-88500	0%	0%	0%
steps-89999	33,3%	50%	0%

Tabulka 5.12: Tabulka popisující výsledky 2. části experimentu 6

5.4.10 Závěr experimentu

V této sekci bude uveden závěr celého experimentu.

Tensorflow-wavenet (viz sekce 3.3.2) dokázala vygenerovat vzorky, které nejprve žádnému z participantů nepřipadaly jako hudební nástroj, ale postupným vylepšováním architektury WaveNet dokázala větší množina participantů určit z vygenerovaných vzorků, o jaký hudební nastroj se jedná.

Všeobecně také z výsledků vyplývá, že ve dvou třetinách byly dva nejlepší výsledky generování s počtem trénovacích kroků 83000 a 85000 a také k těmto výsledkům dokázala největší množina participantů přiřadit správný typ zvuku z původních vzorků.

Experiment 1. Všichni participanti se shodli, že slyšeli pouze šum. 40% participantů vypadalo po testování a velkém soustředění poměrně vyčerpaně, což se následně potvrdilo z post-testu (viz sekce 5.6.1)

Experiment 2. Jedna čtvrtina participantů se domnívala, že slyšela naznak nějakého bubnu a pak šum, ale ostatní participanti slyšeli pouze šum. 60% participantů vypadalo po testování a velkém soustředění poměrně vyčerpaně a některé dokonce bolela hlava, což se následně potvrdilo z post-testu (viz sekce 5.6.1)

Experiment 3. Participanti se shodli, že ve dvou třetinách testovacích vzorků byl slyšet šum, v ostatních případech slyšet nebyl, a také jedna čtvrtina participantů se shodla, že slyšela naznak nějakého bubnu, ale ostatní participanti nedokázali určit, co slyšeli. Nikdo z participantů nevypadal po testování vyčerpaně, což vedlo k tomu, že se musel zvýšit počet testovacích dat pro 1. část v dalším experimentu.

Experiment 4 a 5. Participanti se domnívali, že v jedné třetině testovacích vzorků byl slyšet šum nebo ticho. Jedna čtvrtina participantů si byla jistá,

jaký hudební nástroj slyší. Další čtvrtina participantů slyšela náznak bubnu, ale nevěděla jakého typu. Dvě čtvrtiny participantů nedokázaly určit žádný hudební nastroj. 20% participantů vypadalo po testování a po poměrně velkém soustředění vyčerpaně, což se následně potvrdilo z post-testu (viz sekce 5.6.1). To značí, že 24 testovacích vzorků pro první část je téměř optimální.

Experiment 6. Všichni participanti měli totožný názor, a to, že v jedné třetině testovacích vzorcích byl slyšet šum nebo ticho. Téměř dvě třetiny participantů slyšely náznak bubnu a z toho téměř polovina participantů dokázala s jistotou říci, o jaký hudební nástroj se jedná. Jedna třetina participantů nedokázala určit žádný hudební nastroj. 25% participantů vypadalo po testování a po poměrně velkém soustředění vyčerpaně, což se následně potvrdilo v post-testu (viz sekce 5.6.1).

5.5 Experiment se syntézou mezi dvěma instancemi stejného typu

Hlavními cíli této diplomové práce bylo prozkoumání možnosti využití WaveNetu pro syntézu dvou instancí stejného typu zvuku a také prozkoumání možností parametrizace výstupu, který by uživatel mohl deterministky ovlivňovat.

Nejprve by bylo dobré zmínit, že tuto problematiku, která je uvedena výše, nikdo veřejně nezkoumá. Protože lidé, kteří pracují s NN a hudbou, se snaží replikovat výstup z NN tak, aby zněl co nejpřesněji vstupním vzorkům. Tedy není možné uživatelsky měnit, ke kterému vstupnímu vzorku se má učení NN, co nejvíce přiblížit. Na této bázi funguje architektura tensorflow-wavenet.

5.5.1 Uživatelské testovaní experimentu

Uživatelské testovaní experimentů vychází z předchozích experimentů (viz sekce 5.4), proto zde nebudou popsané totožné části uživatelského testovaní (viz sekce 5.4.1), jako např.: Ice-breaking (Rozbití ledu). Uživatelské testování je rozděleno do následujících částí, které jsou níže popsané.

Plnění úkolů

Hlavní činností účastníků je naplnění připraveného seznamu úkolů. Testování je rozděleno na tři části.

První část se prováděla v domácím prostředí participanta. Participant byl veden osobně nebo přes komunikační zařízení moderátorem. Úkolem participanta bylo poslechnout si 6 zvuků a seřadit je podle toho, jak se mu líbí.

V druhé části měl participant za úkol poznat, jestli se přehrávaný zvuk skládá ze dvou zvuků nebo pouze z jednoho. Těchto zvuků bylo třicet, byly syntetizované a originální zvuky.

V třetí části participant dostal množinu 10 zvuků, která se skládala z devíti originálních zvuků (3 bass drums, 3 snare drums a 3 hihats) a jednoho vygenerovaného zvuku pomocí NN. Úkolem participanta bylo určit, z jakých vzorků se skládá syntetizovaný zvuk vygenerovaný pomocí NN. Toto bylo prováděné se třemi testovacími datasety.

Testovací scénáře

Pro participanty, kteří byli testováni osobně. Participantům pouštěl zvuky moderátor, proto zde není uvedené, jak si dané zvuky mají participanti pouštět.

1. část.

1. Ohodnoťte zvuky od 1 do 6 podle toho, jak se vám líbí. 6 znamená, že se vám daný zvuk líbí nejvíce a naopak 1 znamená, že se vám daná skladba líbí nejméně. Žádné číslo nesmí být použité víckrát než jednou.

2. část.

1. Ke každému zvuku určete, jestli se skládá zvuk, který se bude přehrát, z jednoho čí dvou zvuků, Neboli do tabulky stačí napsat k danému zvuku 1, pokud si myslíte, že se přehrávaný zvuk skládá z jedno zvuku, anebo 2, pokud si myslíte, že se přehrávaný zvuk skládá ze dvou zvuků.

3. část.

1. Přiřaďte k 1. zvuku, z kolika následujících částí se skládá. Může se tedy skládat z 0 - 9 následujících zvuků. Do tabulky napište pozici zvuku, který slyším v prvním zvuku. Např.: pokud uslyšíte v prvním zvuku 1. a 2. zvuk, napište do tabulky 1 a 2.

Pro participanty, kteří byli testováni vzdáleně. Participanti si pouštěli zvuky sami, proto zde je uvedené, jak si dané zvuky mají participanti pouštět. Moderátor je pouze vedl.

1. část.

1. Ujistěte se, že jste v klidném prostředí.
2. Nastavte si prostředí (reproduktory či sluchátka) tak, jak jste zvyklí poslouchat hudbu.

3. Ohodnoťte zvuky ve složce od 1 do 6 podle toho, jak se vám líbí. 6 znamená, že se vám daný zvuk líbí nejvíce a naopak 1 znamená, že se vám daný zvuk líbí nejméně. Žádné číslo nesmí být použité víckrát než jednou.

2. část.

1. Opět se ujistěte, jste v klidném prostředí.
2. Opět se ujistěte, že máte nastavené prostředí (reproduktory či sluchátka) tak, jak jste zvyklí hudbu poslouchat.
3. Ke každému zvuku ve složce určete, jestli se přehrávaný zvuk skládá z jednoho čí dvou zvuků. Do tabulky stačí napsat k danému zvuku 1, pokud si myslíte, že se přehrávaný zvuk skládá z jedno zvuku, a nebo 2, pokud si myslíte, že se přehrávaný zvuk skládá ze dvou zvuků.

3. část.

1. Opět se ujistěte, že jste v klidném prostředí.
2. Opět se ujistěte, že máte nastavené prostředí (reproduktory či sluchátka) tak, jak rádi posloucháte hudbu.
3. Nejprve si poslechnete 1. zvuk v 1. složce
4. Poslechněte si všechny zvuky, které jsou v 1. složce. Přiřaďte k 1. zvuku z kolika následujících se skládá zvuk. Může se tedy skládat z 0 - 9 následujících zvuků. Do tabulky napište pozici zvuku, který slyšíte v prvním zvuku. Např.: pokud uslyšíte v prvním zvuku 1. a 2. zvuk napište do tabulky 1 a 2.
5. Pokud máte více složek, udělejte body 3 a 4 na ostatních složkách. Např. u 2. složky si nejprve přehrajte 1. zvuk, atd.

5.5.2 Experiment 7

Cílem experimentu je použití architektury WaveNet (viz sekce 3.3.1) tak, aby dokázala udělat parametrizovanou syntézu mezi dvěma instancemi stejného typu.

Upřímně jsem nevěděl, jak experiment vyřešit pomocí architektury tensorflow-wavenet (viz sekce 4.5), proto jsem se rozhodl kontaktovat vývojáře, kteří pracují na vývoji architektury tensorflow-wavenet. Konkrétněji jsem kontaktoval dva vývojáře, kteří tvoří pro google, a to: Igora Babuschkina a Quima Llimona. Nasbíral jsem mnoho informací, ale uvedu opravdu nejzásadnější. Pokud bychom chtěli řešit danou problematiku v architektuře tensorflow-wavenet, je to prakticky velmi špatně proveditelné, protože by se do dané

implementace musela přidat další NN (možná dvě NN). Přidaná NN by sloužila k naučení syntézy, nebo vytvoření syntézy. I když by přidaná NN byla malá, mohla by výrazně zvýšit časovou náročnost pro trénování NN. Od této myšlenky a dalších úprav tensorflow-wavenet jsem musel upustit, protože trénování NN jsem prováděl na školním serveru (viz sekce 4.4). Další informace byla tato: pro to, aby vývojář věděl, co se vygeneruje, a aby tuto implementaci mohl uživatel deterministicky ovlivňovat, hodí se architektura NSynth (viz sekce 2.5.2), ale implementace generátorů musí být předělaná tak, že oba vstupní vzorky se enkodují (viz obrázek 2.26(b)). Poté se enkodované vzorky nějakým způsobem interpelují. Výsledek syntézy bude poslán na dekódování. Výstup dekódování by měl být syntézou obou vzorků. Pokud bych byl přidal parametr, který by určoval, ke kterému vstupnímu vzorku by měl být výsledek podobnější, tato teorie by odpovídala k problematice diplomové práce.

Poslední teorii jsem zrealizoval (viz výše). Tedy použil jsem k experimentu architekturu NSynth (viz sekce 3.3.2), která bude syntetizovat dva vzorky stejného typu, který budou nejdříve enkodovaný. Pro syntézu jsou např.: tyto 2 možnosti:

- Lineární interpolace, častěji se používá označení mix. Lineární interpolace je tvořená po vzorcích určitých dvou zvuků. Díky pythonu se mix může napsat tato problematika takto: (instance1 + instance2)/2.0. Tedy instance1 a instance2 určují dva enkodované vzorky a 2,0 je konstanta, pomocí které se drží výsledný signál mezi hodnotami -1 a 1.

  U mixu je důležité sjednotit délku obou vzorků. Tedy kratší vzorek je potřeba dovyplnit 0 (tichem).

- Crossfade je křížový útlum, což se může popsat jako prolínání 2 vzorků. U prvního vzorku se používá fade out (plynulé potlačování amplitudy až do minima) a u druhého vzorku se provádí fade in (plynulé narůstání amplitudy z minima na příslušnou hodnotu). Důležité je řešit u crossfade: jeho délku, průběh křivek fade in a fade out, kdy se budou křivky protínat. [47].

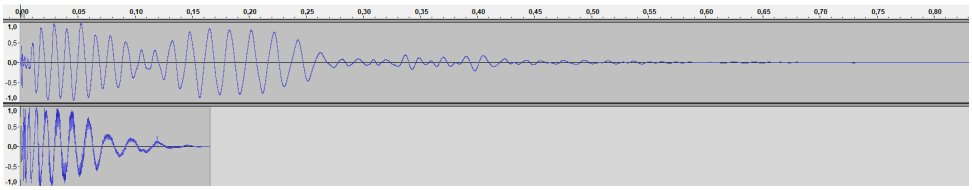
  Došlo mi, že u crossfade je důležité vědět, který ze vzorků je delší, a ten používat jako druhý, neboli se pro něj bude dělat fade in.

Pro ověření teorie jsem provedl 20 generování s modelem, který je k dispozici od autorů. U všech generování byly použité dva vzorky, které jsou rozdílné, ale zároveň jsou stejného typu. Jinými slovy: musely být použít 2 rozdílné bass drums, nebo 2 rozdílné snare drums a nebo 2 rozdílné hihats. Deset generování bylo vytvořeno pomocí mixu jako syntéza dvou enkodovaných instancí a deset generování bylo vytvořeno pomocí crossfade jako syntéza dvou enkodovaných instancí. Díky tomuto se potvrdila teorie a mohl jsem do implementace přidat parametr num_synth, pomocí kterého se určovalo, jestli výsledek generování bude znít podobněji jako 1 či 2 vstupní vzorek.

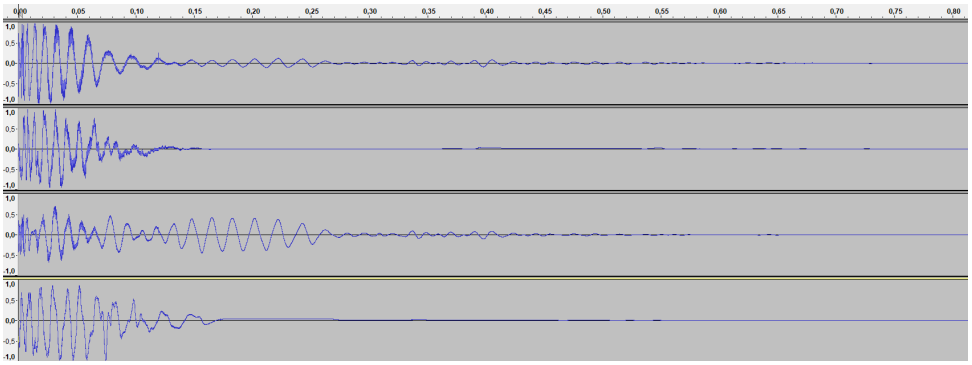
Tento atribut měl 10 hodnot. 0 znamenala, že vygenerovaný výsledek má být co nejvíce podobný 1. vstupnímu vzorku. 10 znamenala, že vygenerovaný výsledek má být co nejvíce odlišný od 2. vstupního vzorku. Výchozí hodnota je 5.

Provedl jsem generování mezi dvěma od sebe rozdílnými vzorky stejného typu a různou hodnotou atributu num_synth.

Na obrázku 5.8 jsou znázorněné dva originální zvuky (viz obrázek 5.8(a)) a vygenerované zvuky (viz obrázek 5.8(b)). Na obrázku 5.8(b) jsou znázorněné 4 zvuky, které byly vytvořené s hodnotou atributu num_synth 5. 1. je vytvořený pomocí crossfade bez použití NN a 2. vygenerovaný pomocí NN s použitou syntézou crossfade. 3. je vytvořený mixem bez použití NN, 4. je vygenerovaný pomocí NN s použitou syntézou mix. Z obrázků je patrné, že 2. zvuk koreluje k 1. zvuku, naopak 4. zvuk téměř nekoreluje k 3. zvuku. Je očividné, že použití mixu k syntéze dvou zvuků stejného typu rozbíjí amplitudu výsledného zvuku.



(a) : Amplituda v čase pro dva původní zvuky



(b) : Amplituda v čase pro vytvořené zvuky ze zvuků výše

Obrázek 5.8: Amplituda v čase pro dva původní zvuky a vygenerované zvuky z experimentu 7

Výběr testovacích dat

Do uživatelského testování bylo vybráno celkem 66 zvuků. Do první části testování bylo vybráno 6 zvuků, které byly pro všechny participanty totožné. Všechny tyto zvuky byly vygenerované s hodnotou atributu num_synth 5. 3 zvuky byly vytvořené bez použití NN a 3 zvuky byly vygenerované pomocí NN. 2 byly vytvořené pomocí crossfade, 1. crossfade byl vytvořen tak, že delší vstupní zvuk měl funkci fade out, 2. crossfade byl vytvořen tak, že další vstupní zvuk měl funkci fade in, a 1. byl vytvořen pomocí mix.

Do druhé části bylo vybráno 30 zvuků, kde bylo 5 zvuků vytvořeno pomocí crossfade nebo mix hodnotou a atributu num_synth 0 nebo 10, 10 zvuků vytvořených pomocí crossfade nebo mix hodnotou a atributu num_synth 1 až 8, zbylých 15 zvuků jsou dříve zmíněné zvuky, které byly vygenerované pomocí NN. Participanti byli rozděleni do 3 skupin, proto jsem vytvořil 3 testovací datasety.

Do třetí části bylo vybráno 10 zvuků, kde byl 1 zvuky vygenerované pomocí NN, crossfade nebo mix hodnotou a atributu num_synth 0 až 10 a 9 zvuků původních. Třetí část se skládá z 6 složek, tedy celkem bylo vybráno 60 zvuků. Participanti byli rozděleni do 5 skupin, proto jsem vytvořil 5 testovacích datasetů.

Výsledky testování

Výsledky celého uživatelského testování jsou znázorněné v tabulkách (viz tabulka 5.13, tabulka 5.14 a tabulka 5.15).

1. část. Procenta udávají ohodnocení participantů – dle toho, jak se jim líbil daný zvuk. Tedy 0% znázorňuje, že se nikomu nelíbil uvedený zvuk, a 100% znázorňuje, že všem participantům se líbil uvedený zvuk.

Zvuky	Hodnocení
Crossfade delší vzorek fade out bez NN	83,3%
Crossfade delší vzorek fade out s NN	49,2%
Crossfade delší vzorek fade in bez NN	74,2%
Crossfade delší vzorek fade in s NN	37,5%
mix bez NN	72,7%
mix s NN	34,2%

Tabulka 5.13: Tabulka popisující výsledky 1. části experimentu 7

2. část. Procenta udávají, kolik participantů určilo správný počet zvuků, ze kterého se skládaly přehrávané zvuky. Tedy 0% znázorňuje, že nikdo z participantů neurčil správný počet, a 100% znázorňuje, že všichni participanti určili správný počet. Syntetizované zvuky znamenají, že jsou 2 zvuky vytvořené pomocí crossfade nebo mixu.

Syntetizované zvuky	Bass drums	Snare drums	Hihats
S hodnotou atributu num_synth 0 nebo 10 bez NN	85%	85%	60%
S hodnotou atributu num_synth 0 nebo 10 s NN	90%	85%	90%
S hodnotou atributu num_synth 1 až 8 bez NN	70%	60%	55%
S hodnotou atributu num_synth 1 až 8 s NN	30%	25%	15%

Tabulka 5.14: Tabulka popisující výsledky 2. části experimentu 7

3. část. Procenta udávají, kolik participantů určilo správný počet zvuků, z kterých se skládají přehrávané zvuky, kolik participantů určilo správný typy zvuku z původních např.: bass drum, kolik participantů správně přiřadilo alespoň jeden zvuk, z kterého se skládají přehrávané zvuky z původních vzorků, a kolik participantů správně přiřadilo oba zvuky, z kterých se skládají přehrávané zvuky z původních vzorků. Tedy 0% znázorňuje, že nikdo z participantů neurčil správný počet, a 100% znázorňuje, že všichni participanti určili správný počet.

	Bass drums	Snare drums	Hihats
Správný počet z kolika se skládají zvuky	30%	30%	15%
Správný počet určených typů	75%	50%	30%
Správně určeného alespoň 1 zvuku, ze kterého se skládají	60%	40%	20%
Správně určených 2 zvuků, ze kterých se skládají	5%	0%	0%

Tabulka 5.15: Tabulka popisující výsledky 3. části experimentu 7

Závěr experimentu

Téměř polovina participantů vypadala po testování a po poměrně velkém soustředění velmi vyčerpaně, což se následně potvrdilo z post-testu (viz sekce 5.6.1).

Z 1. části testování se ukázalo, že zvuky vygenerované pomocí NN měly horší výsledky než zvuky vytvořené bez NN. Překvapivým zjištěním bylo

to, že participanti ohodnotili téměř totožně syntetizaci (mix a crossfade) vytvořenou pomocí NN

Ve 2. části se projevilo, že participanti přesněji určují, z kolika zvuků se skládá vytvořený zvuk bez použití NN. 69% participantů určovalo zvuky vygenerované pomocí NN tak, že se skládají pouze z jednoho zvuku. Konkrétně si to mysleli s hodnotami 0 - 2 a 7 - 10 atributu num_synth. Tedy pokud hodnota atributu num_synth byla nastavená na 4 – 6, tak participanti dokázali určit, že vygenerovaný zvuk pomocí NN skládá ze dvou vzorků.

Z 3. části je patrné, že participanti dokázali velmi dobře přiřadit, z jakého typu se skládá vygenerovaný vzorek pomocí NN. Participanti dokázali poměrně dobře z alespoň jeden konkrétní vzorek ze kterého se skládá vygenerovaný zvuk, ale měli velký problém učit, z kolika vzorků se skládá vygenerovaný zvuk.

5.6 Zhodnocení Experimentů

Experimentů bylo provedeno více, než jich je uvedené v kapitole 5. Experimenty, které nejsou uvedené v této kapitole, neměly žádný dopad na zlepšení výstupního zvuku, nebo nedokázaly vygenerovat žádný zvuk.

Z experimentů 1 až 6 je patrné, že převážná většina vzorků z datasetů snare drums a hihat je příliš tichá nebo krátká (viz sekce 4.2). Tzn. že NN nedokáže vygenerovat podobný zvuk původním. Jedná se primárně o šum nebo ticho. Naopak z datasetu bass drums dokázala vygenerovat NN vygenerovat zvuk, který poměrně velká množina participantů dokázala správně přiřadit (viz sekce 5.4).

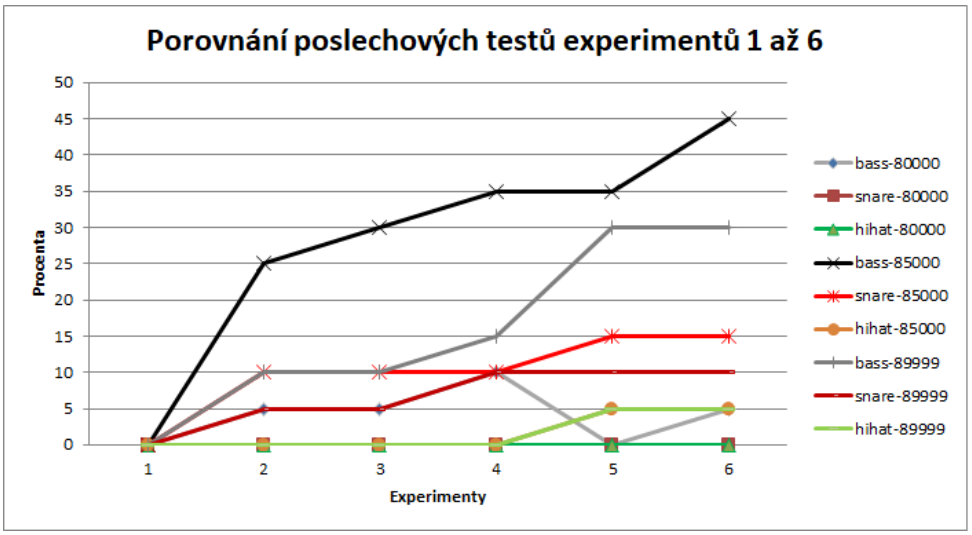
Z experimentů vyplývá, že syntetizování dvou vzorků pomocí architektury WaveNet je možné, ale WaveNet musí být použitý jako autoencoder (viz sekce 5.5). Vygenerovaný zvuk pomocí NSynth velmi dobře koreluje k crossfade.

5.6.1 Zhodnocení celého testovaní

Z experimentů bylo patrné, že určení z kolika instancí se skládá vygenerovaný zvuk pomocí NN je poměrné náročné. Na druhou stranu přiřadit z jakého typu se skládá daný zvuk je poměrně jednoduché. Dokonce není příliš náročné určit minimálně jeden původní zvuk, z kterého se skládá vygenerovaný zvuk.

Experimenty se syntézou různých typu zvuku (viz sekce 5.4) se mohou mezi sebou porovnat tak, že se vybere 1 konkrétní hodnota atributu počtu trénovacích kroků. K tomuto účelu byly vybrané hodnoty 80000, 85000 a 89999, protože se vyskytují od 1. experimentu až do 6. experimentu. Je poměrně velká škoda, že mohla být vybraná hodnota 83000, protože měla jednoznačně nejlepší výsledky. Toto porovnání znázorňuje, kolik participantů

dokázalo určit k vygenerovanému zvuku pomocí NN správný hudební nástroj (viz obrázek 5.9)



Obrázek 5.9: Porovnání poslechových testů experimentů 1 až 6

Z grafu (viz obrázek 5.9) je patrné, že učení NN by stačilo učit s maximálním počtem trénovacích kroků, který je 85000, protože velmi mnoho participantů dokázalo určit, že se jedná o nějaký druh bubnu. Tedy generování zvuků pomocí NN se postupně zlepšovalo. To bylo docíleno pomocí postupným vylepšováním architektury WaveNet především pro datasety sanre drums a bass drums.

Z posledního experimentu (viz sekce 5.5.2) je patrné, že pokud se použije jiný přístup ke generování zvuků, je možné vygenerovat zvuky, které jsou velmi podobné původním vzorkům, ale je velký problém určit z kolika vzorku se skládá vygenerovaný zvuk.

Výsledky post-testu

V této sekci jsou uvedené celkové výsledky post-testů (viz sekce 5.3). Tyto výsledky jsou sjednocené přes všechny experimenty.

Co se Vám líbilo při testování? Při testování se většině participantů líbilo, že jsem jim velmi dobře vysvětlil zadání. Tedy zadání úkolů bylo pro účastníky velmi srozumitelné. Dobrý ohlas také mělo, když se jim povedlo rozpoznat hudební nástroj ve vygenerovaných zvucích.

Co se Vám nelíbilo při testování? Při testování se většině participantům nelíbily některé zvuky, protože zněly nepříjemně, až někdy „trhaly uši“. Především ty, které zněly jako šum.

Jak Vám přišli jednotlivé části testovaní složité a proč? Při testování se většině participantům zdály všechny části poměrně náročné na poslech. Kvůli délkám ukázek, které se musely pouštět několikrát za sebou nebo dokonce ve smyčce, aby se participanti dokázali zaposlouchat do vygenerovaných zvuků. Jednoduché přišlo participantům, že odpovídali podle svých pocitů, tedy neexistuje špatná odpověď.

Kapitola 6

Závěr

Tato diplomová práce se sestávala z řady dílčích úkolů. Prvním úkolem bylo seznámení se s architekturami a zejména s architekturou WaveNet. Díky tomu jsem získal povědomí o teoretickém fungování NN a o úkolech, které jsem musel provést v praktické části (jako například tvorba datatsetů apod.). Dále jsem se seznámil s různými typy tanečních zvuků, z nichž jsem následně vybral skupinu vhodnou pro mé pokusy. Konkrétně jsem si vybral bubny a jejich poddruhy. Jedním z úkolů bylo prozkoumání možnosti parametrizace výstupu. Při použití architektury WaveNet je výsledná syntéza mezi vstupními vzorky randomizovaná. Pokud je WaveNet použitý jako autoenkoder, je možné syntetizovat dva vzorky stejného typu a je možné proces generování deterministicky ovlivňovat změnou přidaného parametru.

Jak bylo výše uvedené, tato diplomová práce se skládá z řady dílčích úkolů (viz podkapitola 1.1) a ty jsou probrané níže:

- **1. úkol - architektury neuronových sítí**: Podle zadání se tato práce se primárně zaměřuje na architekturu WaveNet a její implementace (viz sekce 3.3). Konkrétně dvě tyto implementace jsou v této kapitole podrobně popsané, protože jsou použité v experimentech (viz kapitola 5)

- **2. cíl - typy zvuků**: Práce popisuje typické zvuky (viz sekce 2). Byla vybraná množina bubnů (bass drums, snare drums a hihats) (viz sekce 2 a 4.2) pro práci s NN v experimentech (viz kapitola 5)

- **3. cíl - deterministické ovládání**, **4. cíl - využiti WaveNetu pro syntézu zvuku** a **5. cíl - poslechové testy**: Jsou především prozkoumané v kapitole 5. Parametrizace a syntéza se může vytvořit v architektuře NSynth, která má v sobě architekturu WaveNet. V jiné implementaci s použitím WaveNetu je interpolace náhodná. V této kapitole je provedeno několik experimentů, ke kterým byl provedené poslechové testy (uživatelské testovaní). Také pro tyto cíle jsou důležité sekce 2.1 a 5, kde jsou popsané teoretické znalosti, které se využívají v praktické části.

- **6. cíl - vygenerovaní zvuku**: Daný cíl je popsaný v sekcích 3.2, 3.3 a 5

Příloha A

Literatura

[1] D. Darling, "Instruments." [online], Naposledy navštíveno 23. 3. 2018. Dostupné z http://www.daviddarling.info/encyclopedia_of_music/I/instruments.html.

[2] T. correct, "Midi numbering, the helmholtz pitch notation system, and other octave numbering conventions used in music and music notation." [online], Naposledy navštíveno 22. 3. 2018. Dostupné z http://www.theoreticallycorrect.com/Helmholtz-Pitch-Numbering/.

[3] M. Bloom, "63 in-depth synthesis tutorials by sound on sound." [online], Naposledy navštíveno 21. 3. 2018. Dostupné z http://sonicbloom.net/en/63-in-depth-synthesis-tutorials-by-sound-on-sound/.

[4] T. Singh, "How is artificial intelligence different from machine learning." [online], Naposledy navštíveno 11. 1. 2018. Dostupné z https://www.linkedin.com/pulse/how-artificial-intelligence-different-from-machine-learning-singh.

[5] M. Nielsen, "Neural networks and deep learning." [online], Naposledy navštíveno 14. 1. 2018. Dostupné z http://neuralnetworksanddeeplearning.com/index.html.

[6] K. Hong, "Artificial neural network ann 7 overfitting regularization." [online], Naposledy navštíveno 12. 2. 2018. Dostupné z http://www.bogotobogo.com/python/scikit-learn/Artificial-Neural-Network-ANN-7-Overfitting-Regularization.php.

[7] Brilliant.org, "Feedforward neural networks." [online], Naposledy navštíveno 11. 1. 2018. Dostupné z https://brilliant.org/wiki/feedforward-neural-networks/.

[8] P. Veličković, "2d convolution." [online], Naposledy navštíveno 13. 4. 2018. Dostupné z https://github.com/PetarV-/TikZ/tree/master/2D%20Convolution.

[9] C. Olah, "Understanding lstm networks. 2015," *URL http://colah. github. io/posts/2015-08-Understanding-LSTMs/img/LSTM3-chain. png*, 2015.

[10] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[11] A. v. d. Oord, Y. Li, I. Babuschkin, K. Simonyan, O. Vinyals, K. Kavukcuoglu, G. v. d. Driessche, E. Lockhart, L. C. Cobo, F. Stimberg, *et al.*, "Parallel wavenet: Fast high-fidelity speech synthesis," *arXiv preprint arXiv:1711.10433*, 2017.

[12] A. Dertat, "Applied deep learning - part 3: Autoencoders." [online], Naposledy navštíveno 22. 9. 2018. Dostupné z `https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798`.

[13] N. Phamdo, "Speech compression." [online], Naposledy navštíveno 22. 9. 2018. Dostupné z `http://faculty.uml.edu/jweitzen/16.548/classnotes/Speech%20Compression.htm`.

[14] E. B. Lejaren Hiller, "Electronic music." [online], Naposledy navštíveno 11. 3. 2018. Dostupné z `https://www.britannica.com/art/electronic-music`.

[15] V. Syrový, "Hudební akustika, 1. vyd," *Akademie múzických umění v Praze, Praha*, 2003.

[16] V. Syrový, "Živá hudba, hudební signál a jeho syntéza." [online], Naposledy navštíveno 21. 3. 2018. Dostupné z `http://ziva-hudba.info/article.php?id=215`.

[17] H. DUDI-EY, "The vocoder," 1939.

[18] Tutorialspoint, "Artificial intelligence tutorial." [online], Naposledy navštíveno 11. 1. 2018. Dostupné z `https://www.tutorialspoint.com/artificial_intelligence/index.htm`.

[19] N. J. Nilsson, "Introduction to machine learning. an early draft of a proposed textbook," 1996.

[20] R. Marcotte and H. J. Hamilton, "Behavior trees for modelling artificial intelligence in games: A tutorial," *The Computer Games Journal*, vol. 6, no. 3, pp. 171–184, 2017.

[21] M. Novák, *Umělé neuronové sítě: teorie a aplikace*. Učebnice informatiky, C.H. Beck, 1998.

[22] M. univerzita v Brně, "Neuronové sítě." [online], Naposledy navštíveno 15. 1. 2018. Dostupné z `https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=21471`.

[23] M. Blaha, "Adaptační dynamika neuronu." [online], Naposledy navštíveno 16. 2. 2018. Dostupné z http://portal.matematickabiologie.cz/index.php?pg=analyza-a-hodnoceni-biologickych-dat--umela-inteligence--neuronove-site-jednotlivy-neuron--adaptacni-dynamika-neuronu.

[24] C. Stergiou and D. Siganos, "Neural networks. 1996," 2010.

[25] T. A. Institute, "Analyzing six deep learning tools for music generation." [online], Naposledy navštíveno 10. 1. 2018. Dostupné z http://www.asimovinstitute.org/analyzing-deep-learning-tools-music/.

[26] J. Osmalsky, J.-J. Embrechts, M. Van Droogenbroeck, and S. Pierard, "Neural networks for musical chords recognition," in *Journées d'informatique musicale*, pp. 39–46, 2012.

[27] T. D. Team, "Convolutional neural networks (lenet)." [online], Naposledy navštíveno 12. 1. 2018. Dostupné z http://www.deeplearning.net/tutorial/lenet.html#convolutional-neural-networks-lenet.

[28] K. J. Piczak, "Environmental sound classification with convolutional neural networks," in *Machine Learning for Signal Processing (MLSP), 2015 IEEE 25th International Workshop on*, pp. 1–6, IEEE, 2015.

[29] S. Kum and J. Nam, "Classification-based singing melody extraction using deep convolutional neural networks," 2017.

[30] V. Turchenko, E. Chalmers, and A. Luczak, "A deep convolutional auto-encoder with pooling-unpooling layers in caffe," *arXiv preprint arXiv:1701.04949*, 2017.

[31] T. L. Li, A. B. Chan, and A. Chun, "Automatic musical pattern feature extraction using convolutional neural network," in *Proc. Int. Conf. Data Mining and Applications*, 2010.

[32] E. J. Humphrey and J. P. Bello, "Rethinking automatic chord recognition with convolutional neural networks," in *Machine Learning and Applications (ICMLA), 2012 11th International Conference on*, vol. 2, pp. 357–362, IEEE, 2012.

[33] A. Karpathy, "The unreasonable effectiveness of recurrent neural networks, 2015," *Disponıvel em http://karpathy. github. io/2015/05/21/rnn-effectiveness*, 2016.

[34] Y.-A. Chung, C.-C. Wu, C.-H. Shen, H.-Y. Lee, and L.-S. Lee, "Audio word2vec: Unsupervised learning of audio segment representations using sequence-to-sequence autoencoder," *arXiv preprint arXiv:1603.00982*, 2016.

[35] P.-S. Huang, M. Kim, M. Hasegawa-Johnson, and P. Smaragdis, "Singing-voice separation from monaural recordings using deep recurrent neural networks.," in *ISMIR*, pp. 477–482, 2014.

[36] A. Lahav, E. Saltzman, and G. Schlaug, "Action representation of sound: audiomotor recognition network while listening to newly acquired actions," *Journal of Neuroscience*, vol. 27, no. 2, pp. 308–314, 2007.

[37] D. Johnson, "Composing music with recurrent neural networks," 2015.

[38] J.-S. Kim, "Deepjazz." [online], Naposledy navštíveno 9. 1. 2018. Dostupné z `https://deepjazz.io/`.

[39] D. Eck and J. Schmidhuber, "Learning the long-term structure of the blues," in *International Conference on Artificial Neural Networks*, pp. 284–289, Springer, 2002.

[40] D. Britz, "Recurrent neural network tutorial, part 4-implementing a gru/lstm rnn with python and theano," *URL http://www. wildml. com/2015/10/recurrent-neural-network-tutorial-part-4-implementing-a-grulstm-rnn-with-python-and-theano*, 2015.

[41] J. Engel, C. Resnick, A. Roberts, S. Dieleman, D. Eck, K. Simonyan, and M. Norouzi, "Neural audio synthesis of musical notes with wavenet autoencoders," *arXiv preprint arXiv:1704.01279*, 2017.

[42] M. Blaauw and J. Bonada, "A neural parametric singing synthesizer," *arXiv preprint arXiv:1704.03809*, 2017.

[43] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[44] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[45] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, vol. 2, 2010.

[46] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?," *Journal of Machine Learning Research*, vol. 11, no. Feb, pp. 625–660, 2010.

[47] P. Masri and A. Bateman, "Improved modelling of attack transients in music analysis-resynthesis," in *ICMC*, 1996.

[48] J. ČERNOCKÝ, "Číslicové zpracování řeči: Lpc," *Brno: Vysoké učení technické v Brně, Fakulta informačních technologií*, 2003.

[49] J. Chitode, *Information coding techniques*. Technical Publications, 2007.

Příloha B

Seznam zkratek

AI Umělá inteligence/Artificial Intelligence

AND Logický součin

CNN Konvolucní neuronová síť/Convolutional neural network

CPU Centrální procesorová jednotka/Central processing unit

FFNN Dopředná neuronová síť/FeedForward neural network

GPU Grafická procesorová jednotka/Graphics processing unit

GRU Řízené rekurentní jednotky/Gated recurrent units

JSON JavaScript Object Notation

LPC Lineární prediktivní kódování/Linear Predictive Coding

LSTM Dlouhá krátkodobá paměť/Long Short-Term Memory

MNIST Modifikovaný národní ústav pro normy a technologie/Modified National Institute of Standards and Technology

NAND Negovaný logický součin

NN Neuronová síť/Neural network

NPC Nehráčská postava/Non-playable character

OR Logický součet

RAW surový, nezpracovaný soubory

RNN Rekurentní neuronová síť/Recurrent neural network

UI Umělá inteligence/Artificial Intelligence

WAV Waveform Audio File Format

Příloha C

Obsah přiloženého CD

TODO: dopsat

`|-- Development`	Zdrojový kódy, které byly potřeba k experimentům.
`|-- Drums-datasets`	Datasety, ze kterých se trénovaly modely NN.
`|-- Experiments`	Příklad, co každý jednotlivý experiment obsahoval.
`|-- Generate-sounds`	Vygenerované zvuky z experimentů
`|-- Models`	Příklady natrénovaných modelů z experimentů.
`|-- stibapa1-thesis-2019.pdf`	Tato diplomová práce v PDF
`|-- stibapa1-thesis-2019.zip`	Zdrojový kód této diplomové práce

Příloha D

Lineární prediktivní kódování enkodér-dekodér

LPC je umístěná zde, protože byla v rámci této diplomové práce prozkoumaná a naprogramovaná, ale neobsahovala žádnou práci s NN.

Lineární prediktivní kódování enkodér-dekodér

Tato sekce popisuje pojem Lineární prediktivní kódování, které je možné použít jako enkodér dekodér.

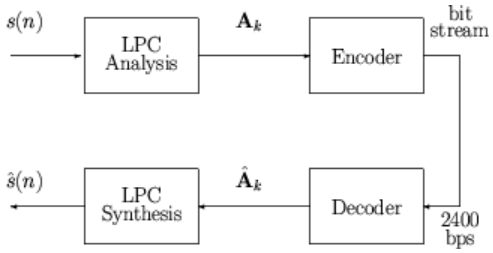
Aby bylo možné porozumět autoekodéru a NN v praxi, bylo potřeba vytvořit enkodér dekodér, který nebude používat NN. K tomuto účelu byl vybrán LPC enkodén-dekodér. Je zde také uvedený, protože je jednoduchý na sestrojení a mohl by být použit jako reference u nějakého experimentu (viz kapitola 4). Je důležité nejprve vysvětlit pojem lineární prediktivní kódování (dále jen LPC), abych mohl vysvětlit LPC enkodér-dekodér.

LPC je technika, která se používá ke zkracování audiosignálů nebo řeči pro představující spektrální obálku digitálního signálu z řeči v komprimované podobě. Tato metoda je jedna z nejužitečnějších pro kódování audia. Zejména při nízké přenosové rychlosti umožňuje velmi přesné odhady parametrů. LPC je využívána pro odhad výsledného vzorku signálu ze vzorků předešlých. Nejčastěji se LPC realizuje tak, že se rozdělí audiosignál na segmenty, které mají velikost 10 - 25 ms, jednotlivé segmenty jsou následně analyzovány. Výsledkem analýzy jsou koeficienty prediktoru, které pak umí další vzorky předpovědět, protože u LPC předpokládáme, že každý vzorek je lineární funkcí vzorků předešlých [48].

LPC vychází z tzv. ARMA modelů, které se vyznačují velkou přesností predikce. ARMA vznikl ze sloučení dvou dílčích modelů MA a AR. MA je model klouzavých průměrů, což si můžeme přestavit jako lineární kombinaci minulých chyb. AR je autoregresní model, což možné popsat jako lineární kombinace vlivů minulých hodnot. [48]

Výše jsem zjednodušeně popsal co je to LPC/LP, ale nejdůležitější část LPC/LP jsem nepopsal, tou je určení LP koeficientů. Důležitým krokem je určení koeficientů tak, aby měly reálný význam a byly co nejjednodušší. K tomu se vybízí dvě metody řešení: kovarianční metoda, korelační metoda. Vybíráme metodu podle toho, jak známe signál ve zkoumaném segmentu. U kovarianční metody neznáme signál v segmentu, pak by řešení vedlo k soustavě lineárních rovnic, jejímž výsledkem by byl nestabilní filtr. U korelační metody známe signál v segmentu, protože je nulový. Tedy porovnávané vzorky mají vlastnost a na diagonálách budou stejné hodnoty, což velmi usnadní výpočet, protože je právě jeden signál v jednom segmentu. Díky tomu lze provádět autokorelaci (Autokorelace je operace se signálem, kdy zkoumáme podobnost signálu samotného se sebou.), což vede k určení autokorelačních koeficientů. [48]

LPC enkodér-dekodér se rozděluje na dvě části: LPC enkodér a LPC dekodér. LPC enkodér-dekodér je znázorněný ve zjednodušeném blokovém schématu (viz obrázek D.1). V blokovém schématu jsou znázorněny bloky LPC analýzy a enkodéru, které se spojují LPC enkodér. LPC dekodér zahrnuje dva bloky ve schématu, které se nazývají LPC syntéza a dekodér.



Obrázek D.1: Blokové schéma LPC enkodéru-dekodéru (převzato z [13])

LPC enkodér dostane na vstupu digitalizovanou formu audia (velmi zjednodušeně dostane na vstup soubor hodnot mezi -1 a 1), která je následně analyzována pro stanovení LP koeficientů. Enkodér následně zakóduje digitalizovaný signál pomocí LP koeficientů a vlastností zvuku, mezi které patří např.: frekvence, atd. [49].

LPC dekodér dekóduje zakódovaný digitalizovaný signál, neboli k zakódovanému signálu jsou generovány LP koeficienty. Poté se signál s LP koeficienty snaží replikovat výsledný signál tak, aby zněl jako originál, což se dělá tak, že výsledný signál je generován ze současných i předchozích LP koeficientů. [49]

Implementace LPC kodér-dekodér

V této sekci je stručně popsaná moje implementace LPC kodér-dekodér (viz sekce D).

Tuto implementaci jsem vytvořil jako možnou referenci k nějakému experimentu. Také mě zajímalo využití několika knihoven mimo NN. Hlavním

důvodem vytvoření této implementace bylo praktické vytvoření enkodéru a dekodér. Rád bych uvedl, že tuto implementaci popíšu především slovně, protože nevyužívá žádným způsobem NN. Budou zde popsané knihovnami/moduly, které byly použité téměř ve všech implementací:

- argparse - Pomocí této knihovny definujeme, jaké argumenty vyžadujeme od uživatele. Pomocí definování argumentů je argparse schopný analyzovat, zda uživatel vyplnil argumenty správně. Velká výhoda této knihovny je, že z definovaného argumentu umí automaticky vygenerovat zprávy o nápovědě a použití, kterou uživatel může spustit příkazem: *python nazev-programu.py -h*. Uživatel tedy má možnost pouštět program bez argumentů, ale i s argumenty např.: *python nazev-programu.py –argument=hodnota-argumentu*
- matplotlib.pyplot - pracuje jako MATLAB, neboli každá funkce pyplot dělá nějakou změnu na obrázku: např. Vytváří obrázek, vytváří plochu pro vykreslování v matplotlib.pyplot, vykresluje některé čáry v ploše vykreslování, zdobí graf s popisky apod.
- wave - slouží k práci s formátem zvuku WAV např.: pro načtení souboru, uložení souboru, atd. Bohužel nepodporuje kompresi a dekompresi, ale podporuje práci s mono souborem i stereo souborem.
- soundfile - slouží ke čtení a zápisu souborů do mnoha různých zvukových formátů Tedy podporuje kompresi, dekompresi, mono souborem a stereo souborem.
- numpy - je základní knihovna pro vědce a analytiky, kteří pracují s Pythonem. Obsahuje např.: n-rozměrné homogenní pole, nástroje pro integraci C/C++ kódu, atd. Určitě stojí za zmínku, že knihovny, ve kterých se objevují větší matice či tabulky, jsou postavené na NumPy.

Navíc v souborech encoder a decoder jsem pracoval s knihovny/moduly scipy.signal (konkretně s modulem lfilter) a scikits.talkbox (konkretně s modulem lpc), které jsou pro uvedenou implementaci velmi důležité. Tyto knihovny velmi usnadňují vývoj LPC kodér-dekodér. Knihovny zde budou krátce popsané:

- lfilter - scipy.signal obsahuje např.: filtrační funkce, interpolační algoritmy, atd. Konkrétně modul lfilter filtruje jednorozměrná data pomocí filtru IIR nebo FIR
- lpc - Talkbox je obecně sada pythonových modulů pro zpracování řeči a signálů. Modul LPC slouží k výpočtu koeficientů lineární predikce.

Implementaci LPC kodér-dekodér jsem rozdělil do tří souborů: main, encoder, decoder, které následně popíši.

Main soubor má za úkol obsluhovat celý program. V této implementaci jsem pracoval s knihovnami/moduly argparse, os, matplotlib.pyplot, wave, soundfile a numpy, většina byla na začátku sekce. V main souboru jsou za definované argumenty, které muže uživatel vyplnit:

- cesta vstupnímu souboru (filename) - jedná se o řetězovou hodnotu, která reprezentuje cestu ke vstupnímu souboru typu WAV. Výchozí hodnotu má nastavenou na soubor s názvem *wavfile.wav*, který je ve stejném adresáři jako main soubor
- cesta výstupnímu souboru (out) - jedná se o řetězovou hodnotu, která reprezentuje cestu k výstupnímu souboru typu WAV. Výchozí hodnotu má jako *wavfile-1.wav*, který se vytvoří do stejného adresáře, jako kde se nachází main soubor.
- počet vzorku na jeden frame (frame) - jedná se o celočíselnou hodnotu, která reprezentuje počet vzorku na jeden frame (rámec), Defaultní hodnota je 0.

Díky výchozím hodnotám u všech vstupních argumentů uživatel nemusí vyplňovat ani jeden parametr. Následně main ověří, zda existuje vstupní soubor, pokud ne program vypíše errorovou hlášku a následně se program ukončí. Pokud zadaný soubor existuje, tak načte pomocí wave a soundfile zadaný soubor typu WAVE (viz fragment kódu D.1).

```
def get_data_from_file(filename):
      data, samplerate = soundfile.read(filename)
      wave_data = wave. open(filename, 'r')
      return data, samplerate, wave_data
```

Fragment kódu D.1: Načtení vstupního souboru

Poté main vypočítá velikost frame a počet framů na základě počtu vzorků a zadaného atributu. Poté zavolá encoder. Po vykonání enkódování se zavolá decoder. Následně main zavolá funkce pro vykreslení grafů a spektrogramů, které pracují s knihovnou matplotlib.pyplot. Nakonec main uloží soubor typu wav na místo, které bylo zadané jako vstupní atribut, pomocí knihovny soundfile `soundfile.write(args.output_filename, decode_data, samplerate)`.

V Enkodéru jsem zvolil nejjednodušší implementaci. Neboli encoder se nainicializuje, následně vypočítá přírůstek, který bude potřebný k určování amplitudy. Vypočítá LP koeficienty za pomoci modulu lpc, z tohoto výsledku vypočítá rozdíl mezi skutečnými hodnotami a vrácenou hodnotou z modulu lfilter a tyto hodnoty zakóduje. V poslední řadě vrátí potřebné informace (`return lpcOrder, amp, residual, coeff, encode_data`)

U decoderu jsem také zvolil nejjednodušší způsob implementace. Neboli decoder se nainicializuje, následně snaží replikovat zakódovaný signál do původního pomocí modulu lfilter, vypočtu přírůstku a propočítání amplitudy jaká by měla být. Poté vytvoří signál, který pošle do mainu (`return decode_data`).