VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

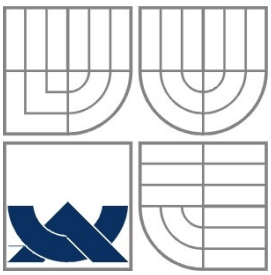
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

TVORBA HUDBY POČÍTAČEM

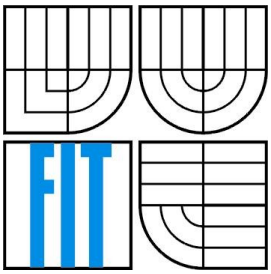
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE JAKUB LANC
AUTHOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

TVORBA HUDBY POČÍTAČEM

MUSIC COMPOSITION BY COMPUTER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAKUB LANC

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAL FAPŠO

BRNO 2009

Abstrakt

Práce se zabývá tvorbou hudby počítačem, konkrétně využitím neuronových sítí pro generování melodií. Nejprve stručně mapuje současné přístupy ke generování hudby a charakteristiky některých užívaných typů neuronových sítí pro tento účel. Na problém nahlíží z širšího úhlu – od získání dat, po jejich transformaci, vhodnou reprezentaci, až po trénování pomocí sítě. Následně popisuje implementaci experimentu pomocí skriptů v prostředí Matlab.

Abstract

Main topic of the thesis is musical composition by means of computers, specifically usage of neural networks for melody generation. Contemporary artificial music composition approaches are briefly mapped, and some types of neural networks used are presented. Implementation of Matlab scripts, allowing melody generation is presented, and the process is illustrated on a few examples.

Klíčová slova

melodie, neuronové sítě, tvorba hudby, midi,

Keywords

melody, neural networks, music composition, midi

Citace

Jakub Lanc: Tvorba hudby počítačem, bakalářská práce, Brno, FIT VUT v Brně, 2009

Tvorba hudby počítačem

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením...
Další informace mi poskytli...
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jakub Lanc
27.5.2009

Poděkování

Děkuji tímto svému vedoucímu Ing. Michalovi Fapšovi, za neutuchající ochotu konzultovat, případně poradit při dalším postupu a to v kteroukoliv denní dobu. Dále děkuji Gyiasettinu Ozcanovi a Cihanu Isikhanovi za ochotu poskytnout mi skripty, které vypracovali v rámci své studie týkající se extrakce melodie z MIDI souborů.

Obsah

1 Úvod

Myšlenky týkající se automatické tvorby hudby se datují již do mnohem vzdálenější minulosti, než kdy se začly objevovat první počítače jak je známe dnes. Už v roce 1600 Athanasius Kircher, ve své knize *Musurgia Universalis* (1600) popisuje mechanický přístroj, který „skládá“ hudbu. Užíval číselných a aritmetických vztahů pro reprezentaci stupnic, rytmů a tempa, kteréžto souhrnně nazýval *Arca Musarithmica*. Toto je však nepochybně pouze jeden z mnoha průkopníků napříč historií, kteří se dostali přirozeným během věcí až do našeho povědomí. Představuji si, že jistě již dřív v myslích těchto jedinců klíčily myšlenky, zda někdy stroj překoná člověka v umění kompozice, či snad dokonce i interpretace? Až dnešní doba však začíná být aktuální pro pokládání si těchto otázek, neboť tempo vývoje technologické i vědomostní úrovně částečně i díky informačním technologiím roste takřka exponenciálně. Při poslechu například některých generativních „kompozic“ Briana Ena již někdy z let minulých mi samotnému na mysli občas tato otázka vytanula – šlo již o sofistikovanou hudbu, která na člověka dokáže zapůsobit, zaujmout, přimět se zamyslet. V této práci odpověď na onu otázku nenabídnu, ani neposkytnu zdaleka srovnatelný hudební výstup. Pokusím se však tuto oblast přiblížit širšímu publiku, a pokusit se alespoň trochu pomoci nahlédnout do oblasti současné tvorby hudby počítači, ve které můžeme, když si to dovolíme, spatřit prolínání na první pohled nesouvisejících oblastí lidského poznání. Kognitivní psychologie, matematika, komplexní systémy, filozofie, to všechno jsou oblasti, o které jsem při shromažďování materiálů přinejmenším zavadil, a které tak chvilkami alespoň prchavě mohou dát nahlédnout do „všeprolínajícího“ charakteru hudby ve své komplexnosti.

Oblast neuronových sítí, na kterou se v této práci zaměřuji nejvíce, je jedním z překotně se vyvíjejících se odvětví poznání na poli informatiky, konkrétně oblasti umělé inteligence, a třeba právě kombinace s hudební vědou, vnímanou převážně jakožto vědou „estetickou“ pro mne reprezentuje onu přitažlivost „interdisciplinárních“ styčných ploch – neuronové sítě ve své umělé formě jakožto prostředek „estetické kreativity“, generování hudby, a naproti tomu biologické, „původní“ neuronové sítě našeho mozku, onu hudbu vnímající, interpretující. Rád bych tedy touto prací poodkryl částečku této poutavé oblasti, i z pohledu „ze zákulisí“ – jak to vlastně celé může probíhat, a případně jak si něco takového také zkusit stvořit.

1.1 Cíle a pojetí práce

Hlavním cílem mé práce je prezentovat nejdůležitější aspekty generování melodií pomocí neuronových sítí, a to od získání trénovacích dat, až po zpětnou reprezentaci výstupu v interpretovatelné formě, a demonstrovat výsledek tohoto přístupu na vybrané neuronové síti (zde konkrétně ESN síti). Dalším souvisejícím cílem bylo poskytnout stručné shrnutí přístupů ke generování hudby obecně a poskytnout tak čtenáři alespoň základní úvod do problematiky, které je hlavní cíl součástí.

Jak již jsem zmínil v úvodu, rád bych spíše postihl praktické pojetí procesu, pro utvoření představy jakým způsobem je třeba možné si takový hudbu generující systém vytvořit, bez zbytečného zabíhání do složitých matematických teorií a podobně (určitě podstatné detaily a základy však rozeberu). Většina studií, se kterými jsem se na podobné téma setkal se totiž většinou zaměřovala na jeden konkrétní aspekt – například experimentování s neuronovou sítí, jejím trénováním, případně srovnáním s dalšími typy, avšak neposkytovala již další bližších implementační detailů, či konkrétní popisu zpracovávaných dat, jejich získání, jejich transformací atp. Proto jsem se rozhodl popis celého procesu pojmout celistvěji – krok po kroku – od získávání vstupních dat, přes manipulaci s nimi, vytvoření a manipulaci s neuronovou sítí, až po zpětnou transformaci výstupu do „slyšitelné“ formy.

Toto „globálnější“ pojetí se možná do určité míry projeví na úkor množství demonstrovaných praktických experimentů, a tím pádem i kvality výsledného melodického výstupu, nicméně hlavní účel této práce vidím spíše spíše v jejím „zasazení do kontextu”. Přitom využívám sítě typu ESN, kterézto jsou v oblasti tvorby hudby zatím víceméně neprobádané, a tedy poskytnu alespoň trochu „čerstvý pohled“, i když opět možná trochu na úkor objektivní (či subjektivní?) kvality výstupu.

1.2 Rozdělení práce

Práce je rozdělena na tři hlavní oddíly – úvodní, obecný (teoretický), a specifický (praktický), přičemž každý z nich je dále rozdělen na několik koncepčních celků.

Obecný oddíl se sestává z následujících částí: první, která je věnována přiblížení tématu automatické tvorby huby počítačem, a následnému stručnému přehledu přístupů k automatickému generování hudby v dnešní době – zde popíši koncepce generování hudby v širším kontextu. Druhá část pokrývá teoretické základy problematiky neuronových sítí, jichž v této práci využívám. Třetí část se věnuje samotnému způsobu generování melodie, na začátku opět zmíním různé alternativní metody, a dále již konkrétní přístupy využívající neuronových sití, jejich výhody a nevýhody. Čtvrtá část je obecným shrnutím kroků, ze kterých se bude sestávat má implementace.

V praktickém oddíle nejprve popisuji konkrétní mnou použité nástroje a prostředí, včetně některých zvažovaných alternativ, zmiňuji jejich přednosti a nedostatky, a poté následují sekce, reprezentující jednotlivé nejdůležitější fáze implementace, ve kterých se věnuji předvedení generování jednoduché melodie na příkladu.

V závěru shrnuji celkové výsledky práce, co by šlo udělat jinak, lépe, a případně jakým směrem by se mohlo ubírat další experimentování.

2 Teoretický oddíl

2.1 Tvorba hudby počítačem

Existuje několik různých paradigmat, perspektiv a dělení této rozmanité oblasti. Například paradigma deterministické/nedeterministické (stochastické[1]), které rozlišuje přístup ke generování hudby na deterministický (tedy daný přesně stanovenými pravidly, a s výstupem vždy replikovatelným na základě počátečních vstupů) a přístup stochastický, kdy může jít o různé druhy pravděpodobnostních, stochastických procesů, a výsledky mohou být i pro stejné vstupní parametry rozmanité. Níže stručně zmíním některá specifičtější dělení a přístupy.

2.1.1 Algoritmické generování hudby

Jako algoritmické bychom asi mohli označit nejširší množinu způsobů generování hudby počítačem, pod nějž spadá několik kategori (které však svými aspekty/metodami/nástroji mohou vzájemně prolínat). Následující rozdělení je volně převzato z [1].

2.1.1.1 Knowledge-based systémy

Používají se hlavně při generování hudby určitého stylu – princip spočívá v izolaci charakteristických znaků stylu, a následném užití těchto parametrů při tvoření nových, podobných kompozic. Jsou tedy založeny na určité předpřipravené sadě pravidel, znalostí. Ty se však většinou netýkají pouze stylu,

1 stochastický – pravděpodobnostní, náhodný

ale i obecných hudebně-teoretických (například harmonizačních) pravidel, které mohou být definovány buďto explicitně, „ručně“, nebo určitým způsobem odvozeny a nashromážděny automaticky.

2.1.1.2 Matematické modely

Matematické modely jsou založeny na matematických rovnicích a náhodných událostech. Pravděpodobně nejpoužívanějším způsobem matematického komponování jsou stochastické a metody, kdy je výsledná skladba výsledkem nedeterministických procesů, a průběh skládání je „skladatelem“ ovlivňován jen do určité míry – například stanovováním vah náhodných parametrů a podobně. Jedním z příkladů budiž třeba Markovské řetězce. Dalšími z matematických metod jsou například různé druhy fraktálů, či generování sekvencí pomocí určitých vlastností celých čísel, prvočísel a podobně.

2.1.1.3 Gramatiky

Hudba může být také interpretována jako jazyk, z určité gramatické množiny. Skladby jsou tvořeny nejprve nalezením „hudební gramatiky“, jejíž pravidla jsou následně užita k vytvoření „smysluplné“ skladby, od lokálnějších syntaktických struktur, po globálnější. Gramatiky můžeme brát jako určitou podmnožinu knowledge-based přístupu.

2.1.1.4 Evoluční metody

V případě evolučního způsobu je algoritmická hudba tvořena evolučním algoritmem. Process začíná populací jedinců, kteří určitým způsobem zastupují určité aspekty hudby (kus melodie, smyčku, harmonický postup atp.), které jsou inicializovány buď náhodně, nebo jsou extrahovány z existujících skladeb. Poté se provádí kroky analogické biologické selekci, rekombinaci a mutaci, přičemž cílem je vytvoření „hudebnějšího“ díla než na počátku. Evolučních algoritmů lze využít i při syntéze. V procesu hraje hlavní roli typicky interaktivní evoluční algoritmus. Největším problémem je pravděpodobně stanovení kritéria fitness funkce, neboť je není snadné ohodnocovtit estetické kvalitu skladby výpočetně. Zdlouhavý, avšak občas užívaný způsob je tedy hodnocení obecentstvem, nicméně existují různé přístupy, alespoň do určité míry tato omezení obcházejí – například užití v kombinaci s neuronovými sítěmi a podobně.

2.1.1.5 Učící se systémy

Sem spadají například právě neuronové sítě, které jsou hlavním tématem této práce. Jde o systémy, kter se, buď na základě trénovacích dat, nebo zpětné vazby, „učí“ charakteristikám prezentovaného vstupu, a dle těchto „znalostí“ (reprezentovaných např. váhami a prahy neuronových spojů v neuronové síti) poté mohou produkovat „nové“ melodie.

2.1.1.6 Hybridní systémy

Jelikož výše uvedené systémy ještě v dnešní době většinou nedovedou samy o sobě produkovat hudbu na příliš vysoké úrovni, co se estetických kvalit, hyerarchičnosti struktury atp. týče, užívají se různé kombinace, které umožňují alespoň do určité míry neutralizovat slabiny jednotlivých metod.

2.1.1.7 Generativní přístup

Generativní přístup je specifická podmnožina algoritmické tvorby. Pojem „generativní tvorba“ nebo „generativní umění“ je široce rozšířený, a například jeden z průkopníků automatického generování hudby Brian Eno je nejznámější právě pro svou generativní torbu. Pojmem generativní hudba bývá označována neustále se měnící, plynoucí hudba, generovaná určitým systémem s určitými pravidly. Dle [10] se principy generování dělí na:

- Lingvistický/Strukturální – základy v užití generativních gramatik
- Interaktivní/Behaviorální – "netransformační", bez vnějších vstupů skladatele
- Kreativní/Procedurální – generována procesy iniciovanými nebo mediovanými skladatelem
- Biologický/Emergentní – nedeterministické biologické, či „ekologické" principy

2.2 Neuronové sítě

Jelikož generování melodie pomocí neuronových sítí je centrálním motivem této práce, budu se jimi zabývat obsáhleji než předchozímy přístupy, a pokusím se i přiblížit základy jejich fungování, pro snažší pochopení procesů, které se za učením a následným generováním melodie skrývají.

Pojem „neuronová síť" bude v kontextu této práce reprezentovat síť „umělých neuronů" (programové konstrukty, napodobující charakteristické klíčové vlastnosti biologických neuronů) [2]. Jde o výpočetní model, založený na „konekcionistickém" přístupu. Vě většině případů je neuronová síť adaptivním systémem, který mění svou strukturu a/nebo parametry na základě externích či interních informací, které procházejí sítí [9].

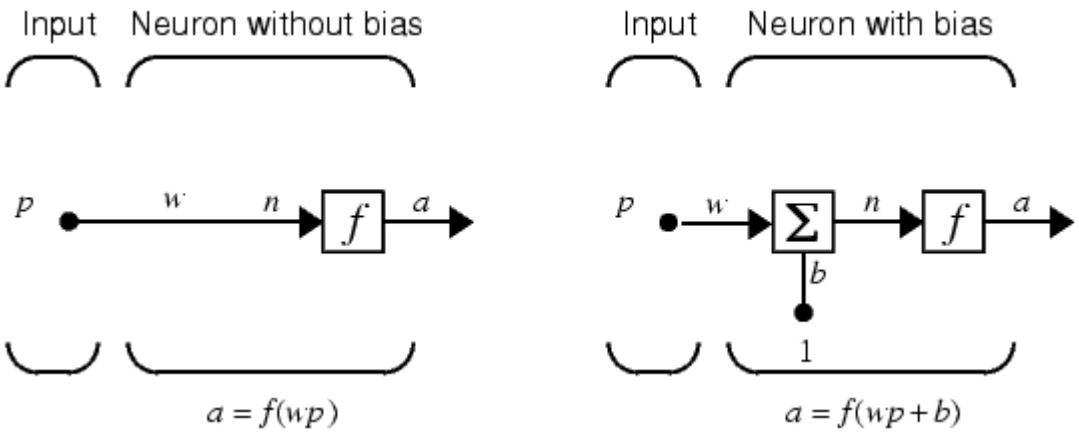
V oblasti umělé inteligence se neuronové sítě s úspěchem využívají například pro řešení problémů souvisejícíchs analýzou řeči (speech recognition), obrazovou analýzou, adaptivním řízením a podobně, a to ať již při řešení specifických úkolů, nebo v komplexnějších systémech, jako jsou různé autonomní roboty, či dokonce roboti. Obecně lze využití neuronových sítí rozdělit do 3 hlavních oblastí dle využití/způsobu činnosti.

- Klasifikace, rozpoznávání vzorů (pattern recognition) a sekvencí
- Zpracování dát – filtrování, separace šumu, komprese atp.
- Aproximace funkcí, regresní analýza, predikce časových řad.

Právě třetí oblast nejvíce odpovídá našemu problému (predikce časových řad). Je dokázáno, že dopředná statická (feed-forward) síť s jednou skrytou vrstvou neuronů dokáže, v případě, že aktivační funkce skrytých neuronů je sigmoidálního typu (tedy logsig či tansig), aproximovat libovolné spojité funkční (1 ku 1, či n ku 1) zobrazení z jednoho x rozměrného prostoru do druhého, je-i počet skrytých neuronů dostatečně vysoký a je-li x konečné (dle [4]). Forma melodie, jakožto časové řady hodnot výšek tonů tomuto odpovídá, a tedy by měla být pomocí neuronových sítí replikovatelná.

2.2.1 Základní struktura a fungování neuronové sítě

Jednoduchá neuronová síť se obvykle skládá ze vstupní, volitelně skryté, a výstupní vrstvy neuronů, které jsou mezi sebou dle určitých pravidel propojeny.

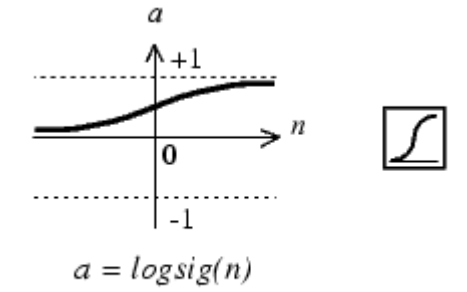


Obrázek 1: Matematický model neuronu bez a s bias vstupem [13]

Jednoduchý skalární neuron se obecně skládá ze vstupního signálu (*p*), váhy přiřazené spoji (*w*), volitelně prahové hodnoty *b* („bias“), aktivační funkce (nebo též přenosová - „transfer function“) a výstupu *a*. Aktivační funkce „mapuje“ výslednou vstupní hodnotu na hodnotu výstupní, dle svého typu a určení.

„Aktivační funkce mohou mít různý charakter průběhu funkce. Jednak je to skupina *lineárních aktivačních funkcí*, které bývají umístěny ve výstupních vrstvách neuronových sítí, a dále převažující skupina *nelineárních aktivačních funkcí*, které se využívají ve skrytých vrstvách neuronových sítí.“ cit. dle [10].

Pro příklad uvedu logsigmoidální aktivační funkci, která se často využívá v mnohovrstvých perceptronových[2] sítích, z důvodu její diferencovatelnosti. Vstupní hodnoty sigmoidální funkce nabývají celého rozsahu platnosti reálného čísla a výstupem je sigmoidální funkce, nabývající hodnot v rozsahu 0.0 až 1.0.



Obrázek 2: Logsigmoidální aktivační funkce (převzato z [13])

Po návrhu topologie a vytvoření neuronové sítě (tj. vytvoření jednotlivých vrstev neuronů, ustanovení spojů a nastavení počátečních váhových parametrů a případně prahových hodnot) následuje opakované provádění trénovací procedury, kterou zajišťuje *trénovací algoritmus*, jenž

2 perceptron – neuron s výstupními hodnotami 1 či 0

modifikuje nastavení jednotlivých váhových hodnot neuronové sítě tak, aby bylo dosaženo přesné klasifikace informací v neuronové síti. Přesnost algoritmu trénování je dána kritériem *chyby sítě*. Nejčastějším měřítkem této veličiny je MSE – mean squared error - střední kvadratická odchylka. Ta je vypočtena jako průměr druhých mocnin vzdáleností trénované hodnoty od predikované.

Proces trénování neuronové sítě probíhá následovně: Na začátku jsou parametry neuronové sítě (váhové a prahové hodnoty) zpravidla nastaveny náhodně v rozsahu hodnot od -0.5 do +0.5. Pro vlastní trénování neuronové sítě je nutné mít množinu dat pro trénování. Zástupci vstupních dat z tréninkové množiny jsou předány na vstup neuronové sítě a na základě průchodu dat sítí získáme výstupní data neuronové sítě. Tato data jsou porovnána s výstupními daty tréninkové množiny dat a na základě jejich rozdílu jsou vybraným algoritmem (pro příklad BP – backpropagation, BPTT – backpropagation-through-time u rekurentních sítí, Lavenber-Marquardt a podobně) modifikovány parametry neuronové sítě, většinou pomocí spočtení gradientu chyby a určitou formou jeho distribuce zpětným spěrem skrz síť. Tato procedura se opakuje pro všechna data z tréninkové množiny, která byla získána v jednotlivých časových okamžicích, tak dlouho, dokud nebude dosaženo potřebné přesnosti neuronové sítě [9].

2.2.2 Statické vs. Dynamické

Neuronové sítě se vyskytují v širokém spektru podob a vlastností, proto zde uvedu alespoň jedno z možných rozdělení – na statické a dynamické sítě.

2.2.2.1 Statické neuronové sítě

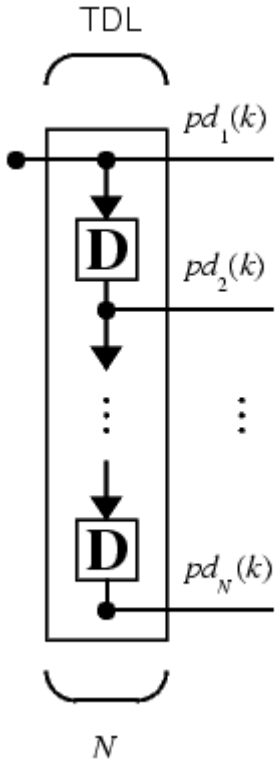
Statické neuronové sítě, nejčastěji reprezentovaní tzv. feed-forward (dopřednými) sítěmi neobsahují žádné rekurentní prvky - zpětnovazebné smyčky, či TDL[3]. Tyto sítě nemají paměť, a tedy konkrétní výstup závisí vždy na odpovídajícím vstupu. Jde tedy z matematického hlediska vpodstatě o funkci, ač složitou. Předchozí ani následující vstupy na něj nemají vliv – síť tedy nemá paměť, nerozpoznává kontext. Proto tedy není tento typ pro náš účel vhodný sám o sobě. Nicméně potřebná „kontextová“ paměť lze alespoň do určité míry nasimulovat pomocí vhodné reprezentace vstupních dat – například tzv. technika „sliding time window“, kterou přiblížím v sekci zabývající se reprezentací dat.

2.2.2.2 Dopředné dynamické (Feed-forward dynamic)

Tento typ sítí obsahuje zpožďovací smyčku či smyčky TDL, přivádějící opožděný vstupní signál zpět na vstup, či na další jinou vrstvu neuronů. Stále však nejde o „plně“ rekurentní síť, neboť pracuje neobsahuje zpětnovazebnou smyčku tvořenou výstupem sítě zpět na vstup. Výstupem TDL je *N*-rozměrný vektor, tvořený hodnotou signálu v aktuálním čase, předchozí hodnotou, atd. až do -N.

V určité konfiguraci bychom mohli přirovnat užití TDL jednotky k užití vstupů ve tvaru posuvných okének, přičemž míra „paměti“ (neboli délka kontextu vstupu, který ještě ovlivňuje aktuální výstup) je omezena délkou zvolené prodlevy TDL signálu, či velikosti časového okénka. Pro tuto omezenou kapacitu bývá tento typ sítí označován také jako FIR (finite impulse response) sítě [13].

3 TDL – tapped delay line



Obrázek 3: TDL jednotka se zpožděním N (převzato z [13])

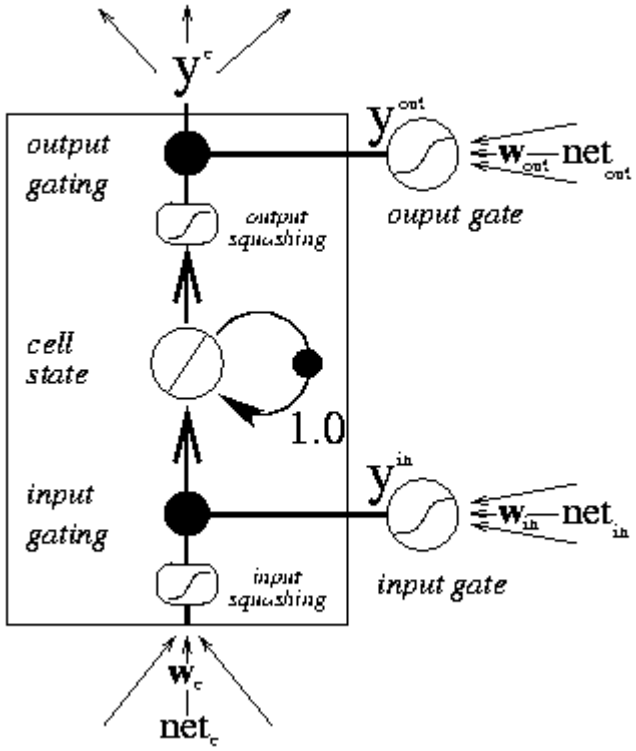
2.2.2.3 Rekurentní neuronové sítě (RNN)

Zásadní rozdíl oproti statickým, i dopředným dynamickým sítím je, že RNN jsou dynamickým, nelineárním systémem – díky svým rekurentním spojením v RNN může probíhat „samoudržující se“ (self-sustained) aktivita v průběhu času, a to i při momentální absenci vstupu. [14] Tato aktivita se tedy může projevovat jako určitá forma nelineární dynamické paměti, a určité typy těchto sítí se tedy také nejvíce (oproti ostatním typům sítí) blíží charakteristikám lidského mozku.

Tyto sítě bývají téže označovány jako IIR (infinite impulse response), nicméně toto označení je do určité míry zavádějící, jelikož problém délky kontextu, v němž je síť schopna si „pamatovat“, a reprodukovat strukturu vyšších měřítek, je u základních typů těchto sítí zásadním – gradient chybové funkce totiž má tendenci se postupně snižovat, až „vymizí“, a pro RNN je tedy obtížné zachytit dlouhodobější časové závislosti ve vstupní sekvenci, jak píše i [8].

2.2.3 Long Short-Term Memory (LSTM)

Jedním z typů rekurentních neuronových sítí, které problém rychlého „zapomínání“ řeší jsou tzv. Long Short-Term Memory sítě.



Obrázek 4: Blok LSTM sítě s jednou rekurentní buňkou [15]

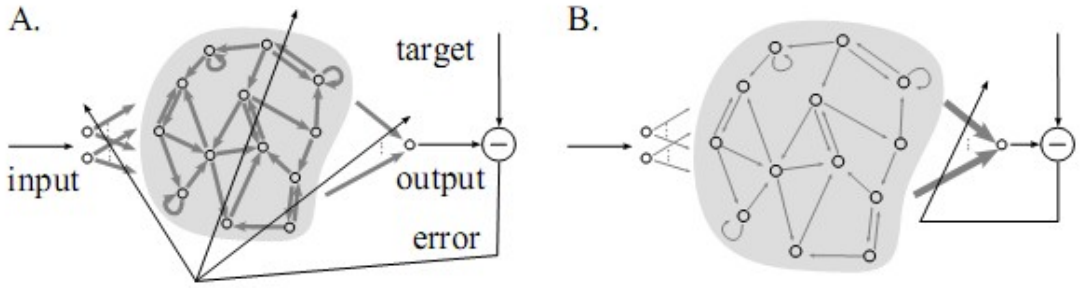
U vybraných problémů si LSTM síť dokáže zapamatovat informace po téměř libovolně dlouhou dobu, dle [15] např. i přes 1000 kroků (epoch), s čímž se klasické RNN nemohou srovnávat. Základem je paměťový blok, který obsahuje jednu nebo více rekurentních neuronů, které slouží právě k udržování svého stavu skrz jednotlivé kroky sítě. O tom kdy a nakolik se změní, rozhodují vstupy z brány, které jsou napojeny vzájamně na všechny další brány, případně vstupní neurony.

V různých studiích na které jsem narazil (např. [8]) bylo zjištěno, že LSTM poskytují lepší výsledky při generování melodií oproti většině ostatních rekurentních přístupů. Též se osvědčily v řešení problému Embedded Reber Grammar, což je test rozpoznávání pravidel určitého typu gramatiky, na kterém si klasické RNN „vylamují“ zuby (více viz. [15]), což by mohlo implikovat i případné schopnosti pojmout určitou „hudební sémantiku“.

2.2.4 Echo State Networks (ESN)

Echo State Networks (ESN) jsou poměrně nově se objevivším, avšak velice perspektivním odvětvím neuronových sítí. Jde o specifický podtyp RNN, konkrétně se jedná se o jednu z forem tzv. „Dynamic Reservoir computing“ (DR), v souvislosti se sítěmi tohoto typu se také někdy používá označení LSM (Liquid State Machine) [14]. Tento typ se oproti klasickým rekurentním sítím vyznačuje „řídkým“ propojeními mezi neurony, přičemž mohou být měněny jenom váhy spojů, které vedou ze skrytých vrstev na výstupy. To mimo jiné také znamená, že výpočty probíhají mnohem rychleji, než u klasických „hustě“ propojených sítí. Síť si zjednodušeně můžeme představit vpodstatě jako ohraničenou „nádobu“, jejíž struktura vzájemných spojení je vygenerována (obvykle) náhodně, a

uvnitř níž probíhají dynamické procesy, neovlivněné učením – to probíhá pouze na již zmíněném hranici s „okolím“ (vstupy/výstupy) [14].



Obrázek 5: Klasická RNN vs. ESN - tučné šipky znázorňují směr, ve kterém dochází k úpravě vah (převzato z [14])

Položil jsem si otázku, zda by se vhodnost tohoto typu sítě pro predikci časových řad mohla promítnout i do reprodukce melodií, a rozhodl se pro výběr tohoto druhu sítě pro provedení experimentů.

2.2.5 Problém over-fitting

Tento problém, který se často vyskytuje při trénování neuronových sítí zde zmiňuji z toho důvodu, že jsem ně nej při svých experimentech několikrát narazil, a je k problematice generování melodií relevantní, jak zmíním níže.

Over-fitting, tj. “přetrénování“, je jev, kdy se síť naučí daná data víceméně „nazpaměť“, na úkor schopnosti zobecnit charakteristické rysy. Chyba na trénovací množině dosahuje nízkých hodnot, avšak když jsou síti prezentována nová data, chyba neúměrně roste. Síť si „zapamatovala“ trénovací příklady, avšak nenaučila se zobecňovat pro nové situace, a v případě obdržení „neznámých „ dat jsou výstupy nestabilní, a např. v případě ESN sítí dojde k nekontrolovatelné oscilaci a výkyvům hodnot.

Problém přetrénování je aktuální hlavně v případech, kdy máme omezené množství trénovacích dat – pokud je počet parametrů sítě značně menší než počet vzorků trénovací sady, tento jev víceméně nehrozí. V našem případě však aktuální spíše je (melodie nejsou příliš „bohatým“ zdrojem dat), a tedy je na místě alespoň zmínit způsoby, jak přetrénování zabránit. Jedním z nich, který je přímo integrován v Neural Network toolboxu Matlabu, a implicitně je zapnutý při použití některých GUI funkcí, je tzv. „Early Stopping“, nebo též „Stop-search“ ([13]). Tato technika spočívá v rozdělení dat do tří sad – trénovací sada, pomocí které se počítá chybový gradient a updatují se parametry sítě. vůči druhé množině, validační, se provádí sledování chyby při procesu testování. Validační chyba se v počáteční fázi trénování obvykle snižuje spolu s trénovací chybou, avšak jakmile síť začne být přetrénovaná, validační chyba se obvykle začne zvyšovat [13]. Jakmile se zvýší po specifikovaný počet iterací, trénování je zastaveno. Tento způsob však pro naše účely není příliš vhodný, jak zmíním dále.Nastavování parametrů a vůbec volba architektury sítě je tedy velmi sofistikovaným a citlivým procesem, a na těchto počátečních podmínkách do velké míry závisí kvalita výstupu. Proto se mnohdy v kombinaci s neuronovými sítěmi používají ještě „meta-algoritmy”, které pomáhají optimalizovat počáteční nastavení parametrů, či architektury sítě. Pro tento účel se například používají různé formy evolučních elgoritmů, nebo např. PSO (Particle Swarm Optimization), pro optimalizaci sítí, generujících melodie. Takovýto přístup je však bohužel nad

časový/prostorový rámec této práce, proto se spokojíme s podklady, získanými na základě již provedených experimentů, případně s „ručním“ a do určité míry náhodným přístupem.

2.3 Neuronové sítě a melodie

Co je to, a jak bychom nejlépe mohli uchopit a předat melodii neuronové síti, aby ji dokázala věrně replikovat, a zároveň tvořit v podobném duchu v případě neznámých počátečních vstupů?

Když se podíváme na konturu melodie[4], můžeme spatřit „obyčejnou“ křivku v závislosti na čase, tedy jakousi časovou řadu, a mohli bychom si říci, že tedy je to jasné – neuronové síti bude stačt posílat jednotlivé datové body této řady (výšky not) na vstup, a „ať se učí. Nicméně tento zdánlivý charakter obyčejné časové řady může být do velké míry matoucí – může totiž implikovat, že za melodií stojí jakási „funkce“, která by se dala aproximovat například pouze na základě předchozí noty, nebo dokonce časového kroku. To je však daleko od pravdy – u každé noty je důležitý její kontext, který jí posléze dává v rámci celku „význam“ při naší kognitivní interpretaci, a tedy je závislá na dalších notách před ní. Je tedy důležité brát v potaz časoprostorový („spatio-temporál“) charakter hudby, potažmo melodie – má svou určitou strukturu v průběhu času. A to jak v měřítku lokálním (např. v rámci jednoho taktu), tak i globálnějším (segmenty, fráze, pasáže, sloky a podobně).

Rozdíl mezi tím, co „vidí“ neuronová síť, a tím, co vidíme a slyšíme my, se pokusím ilustrovat na příkladu krátké melodické sekvence. Informace, které totiž na první pohled či poslech z melodie dokážeme interpretovat, jsou mnohem bohatší, než by mohlo na první pohled vypadat. Tento rozdíl bychom mohli přirovnat třeba k pohledu na melodickou konturu, oproti poslechu melodie, a zároveň sledování její notové reprezentace v notovém archu. V prvním případě pro nás půjde pravděpodobně o nic neříkající křivku, v druhém případě okamžitě poznáme, jak je melodie rozdělená na jednotlivé takty, můžeme si všimnout případné modulace[5], poznáme významné intervaly (durové a mollové tercie, dominanty atp.), a případně i odhadovat kontext té které noty v harmonickou funkci, pokud je přítomen doprovod. Také pravděpodobně dokážeme rozlišit jednotlivé sekce melodie – refrén, sloku, závěr. Pro neuronovou síť je však aktuální případ první – nic z těchto informací „nevidí“ – pokud obdrží na vstup trvání a délky tónů, je to pro ni jen určitá nicneříkající řada výšek tónů, případné informací o jejich dělkách. Žádné další implicitní vztahy.

Abychom co nejvíce síti práci usnadnili, měli bychom na její vstup posílat co nejvíce informací, které navíc o dané datové sekvenci (melodii) víme. Čím více vztahů a implicitních informací, které totiž o dané sekvenci víme (které vyplývají z naší znalosti hudební teorie, intuitivní znalosti hudby, atp.) síti na počátku poskytneme, tím více možných „záchytných bodů“ bude mít síť pro odvozování obecných pravidel, a tím větší bude v delším horizontu schopnost sítě ony informace generalizovat. Čím více takových faktorů, tím méně bude pak potřeba trénovacích dat, což je v tomto případě důležitý faktor.

Některé z dodatečných informací, které mne napadají by mohly být označení počátků taktu, případně různých segmentů, změn tempa či rytmu, identifikace význačných intervalů, důležité je zachování intervalové při transpozici, označení modulací. Neuronová síť by sice teoreticky měla být schopná se „naučit“ i tyto charakteristiky, ale potřebovala by k tomu obrovká množství trénovacích dat. Způsobů, jakým bychom síti ony podpůrné informace dodávali může být více. Například pokud bychom chtěli síti navíc pomoci identifikovat počátky jednotlivých taktů, mohli bychom jako pomocnou informaci aktivovat samostatný neuron určený pro tuto informaci. Nebo bychom mohli přiřadit další příznak do vektoru, reprezentujícího notu, což by však už mohlo být pro síť více „matoucí“ (jelikož „mícháme“ do vstupního vektoru víceméně nesouvisející informaci).

4 kontura melodie - výšky jednotlivých not zanesené do grafu, případně i se zachováním poměrů jejich délek na ose x

5 modulace – změna tóniny

V implementaci ukázky budu neuronové síti předávat některé implicitní informace pomocí kódování výšky a trvání tónu.

2.4 Přístup zvolený v této práci

Celkové řešení, které zde budu prezentovat, můžeme rozdělit na následující obecné podproblémy:

- Stanovení vhodného typu a formátu vstupních dat
- Zajištění vhodného způsobu načtení těchto dat
- Stanovení vhodné reprezentace vstupních a výstupních dat pro neuronovou síť
- Převedení získaných dat na tuto reprezentaci
- Natrénování vybané sítě
- Generování nových dat pomocí natrénované sítě
- Zpracování výstupu a zpětné převedení do interpretovatelné formy

3 Praktický oddíl

Tento oddíl se zabývá konrétními postupy a experimenty, které jsem prováděl. Jelikož jsem implementaci prováděl v prostředí Matlabu, je výstupem soubor několika pomocných Matlab skriptů, které obstarávají extrakci melodie, vytvoření sítí, jejich trénování, zpětné generování MIDI souborů, pomocné funkce a podobně. Nejde však o celistvý „program“ - vzhledem k charakteru prostředí jsem funkce (skripty) používal víceméně „ručně“ pro jednotlivé úkony.

3.1 Použité nástroje

Před započetím implementace jsem stál před rozhodnutím, jaké pro dosažení zamýšlených cílů vůbec zvolit prostředky. Vzhledem k volné dostupnosti různých kvalitních nástrojů jsem hned zpočátku zavrhl manuální zpracovávání jak dat z MIDI souborů, tak i návrh a trénování neuronových sítí, či výsledné testování – přecejen by to bylo, vzhledem k tématu, zbytečné plýtvání časem a prostorem.

3.1.1 MATLAB 7.6.0 (R2008a)

Jakožto „hlavní“ platformu, na jejímž základě jsem nakonec realizoval celou implementaci, jsem si vybral prostředí matematického softwaru Matlab. Důvodů pro mou volbu je několik: Matlab v sobě obsahuje vlastní skriptovací nástroj, se syntaxí podobnou ostatním skriptovacím jazykům, ale s nespornou výhodou integrace s celým prostředím. Manipulace s daty tedy probíhají ve víceméně jednotném formátu, a odpadá tak velká část problémů s jejich konverzí mezi jednotlivými nástroji a podobně. Dalším „pro“ byla přítomnost toolboxu pro práci s neuronovými sítěmi (Neural Networks Toolbox). Když jsem pak zjistil, že ještě existuje samostatný toolbox pro manipulaci s MIDI soubory, byla to jasná volba. S prací v prostředí Matlabu jsem neměl mnoho zkušeností, nicméně díky přehledné a obsáhlé dokumentaci jsem se s prostředím a skriptovacím jazykem brzy sžil.

3.1.2 Neural Networks Toolbox

Neural Networks Toolbox je veskutečnosti součástí prostředtí Matlabu, nicméně jelikož jsem váhal mezi dalšími, samostatnými, nástroji pro práci s neuronovými sítěmi, uvádím ho jako samostatnou položku. Zahrnuje i uživatelské rozhraní pro trénování různých typů sítí, to jsem však příliš nevyužil.

3.1.3 LSTM Toolbox

LSTM Toolbox jsem stáhl ze stránek Halmstadtské Univerzity ([17]) jakožto samostatný toolbox pro Matlab. Jde spíše o interní projekt univerzity, který byl dán k dispozici veřejně - proto je jeho dokumentace spíše sporadická, a ani na internetu jsem bhužel nenalezl téměř žádné ukázky, či tipy pro řešení problémů (se kterými jsem se potýkal bohužel často).

3.1.4 ESN Tools

Jde též o toolbox pro prostředí Matlabu, a umožňuje práci s ESN sítěmi základních forem. Jde taktéž spíše o experimentální projekt , nicméně do prostředí Matlabu se mi ho podařili integrovat bez problémů. Toolbox je dostupný na stránkách publikací Herberta Jaegera [18].

3.1.5 MIDI Toolbox 1.0.1

MIDI Toolbox je volně stažitelný toolbox pro Matlab, ze stránek finské University of Jyväskylä [20] Zahrnuje několik desítek funkcí pro analýzu a vizualizaci MIDI souborů, rozdělené na funkce konverzní, generační, filtrovací, kreslící, statistické, funkce pro určování klíče, kontur, segmentační, melodické, a funkce pro práci s metrem. V této práci je však nejhojněji užíváno funkce pro načtení MIDI souboru do matice, zápisu matice do MIDI souboru.

Bohužel funkce pro načtení/ukládání MIDI souborů v nespolupracují správně s posledními verzemi Matlabu, proto bylo třeba stáhnout updatované verze a místo externího MEX rozhraní použít samostatné Matlab skripty.

3.1.6 Skripty pro extrakci melodie

Když jsem se rozhodl zvolit si extrakci melodií z polyfonních MIDI souborů, a začal shánět program či skript vhodný pro tento účel, zjistil jsem, že oblast extrakce melodie tímto způsobem je, podobně jako generování hudby, stále ještě vpodstatě „v plenkách“. Nikde jsem tedy žádný vhodný samostatný nástroj či skript nenašel. Jako alternativa se nabízelo použití funkce MIDI Toolboxu `extreme`, která z polyfonního souboru vyextrahuje „extrémní“ noty, co se výšky týče – tedy vpodstatě vrchní či spodní hlas. Tóny melodie bývají často vrchními tóny, vyextrahoval jsem tedy na zkoušku několik stop. Bohužel se ukázalo, že tato funkce je pro extrakci melodie z větší části nevhodná – melodie je totiž ne vždy tvořena pouze vrchními tóny, a situaci navíc komplikují melodie, které jsou rozprostřeny ve více MIDI kanálech. Z mého pohledu tedy v tu chvíli zbývala buď možnost spokojit se s hledáním monofonních MIDI souborů na internetu, nebo nějakým způsobem melodii ze souborů dostat ručně. Naštěstí jsem ale narazil na studii [11], jejíž tématem byla extrakce melodie z MIDI souboru. Autoři shodou okolností vyvíjeli svůj systém v prostředí Matlabu, a dokonce užívali i funkcí

MIDI Toolboxu. Napadlo mne tedy je kontaktovat, a požádat, zda by mi výstup své práce neposkytli. Po menších komplikacích se sháněním aktuální e-mailové adresy mi byly zmíněné skripty zaslány.

Bohužel extrakce fungovala pouze na omezeném množství MIDI souborů, u většiny skript skončil chybou. Po delším bádání se mi však podařilo zjistit, že chyba se týká MIDI souborů, které mají kanály číslované od nuly. Matlab totiž používá indexování buněk („cells“) od jedičky, a v určité fázi zpracování používá číslo kanálu právě jako index těchto buněk – zde tedy vznikala chyba, jelikož docházelo k nepovolovanému indexování od nuly. Stačilo tedy skript upravit přidáním příkazu, který v matici not před zpracováním zvýšil číslo kanálu u všech not o jedničku.

3.2 Trénovací data

Trénovací data by měla sestávat z různých monofonních sekvencí - úseků melodií, přičemž jedním z požadavků na tato data je jejich alespoň částečná homogenita, ve smyslu struktury, „stylu“, či formy - čím pestřejší, variabilnější je totiž soubor melodií, tím je menší množina polečných prvků, a tím pádem i charakteristik, „záchytných bodů“, ze kterých může neuronová síť generalizovat.
Jako ideální se mi jevilo vyextrahovat melodie z několika desítek Bachových kompozic, nicméně bohužel vzhledem k nedostatkům extrakčního algoritmu byly výsledné melodie víceméně nepoužitelné.

Volba vhodného souboru vstupních dat tedy byla celkem obtížná. Navíc pro různé typy se ukazují jako vhodná různá data, nakonec jsem tedy zůstal u jednoduchých melodií, a několika testovacích melodií, které reprezentují určitou charakteristiku (například lokální strukturu, formu AABB, dur/moll) a pomocí jejich užití se dá objektivněji zhodnotit výkon neuronové sítě (nakolik se podařilo síti onen aspekt zachytit či generalizovat).

3.3 Načtení MIDI souboru

Data načteme z MIDI souboru pomocí funkce MIDI Toolboxu `midi2nmat`, kterážto jako argument přijímá pouze název souboru pro načtení. Výstupem je matice not

```
mat =
[
   0.000    0.900    1.000   64.000   64.000    0.0000    0.5510
   1.000    0.900    1.000   71.000   64.000    0.6122    0.5510
   2.000    0.450    1.000   71.000   64.000    1.2245    0.2755
     ...      ...      ...      ...      ...       ...       ...
]
```

Kdždý řádek matice reprezentuje jednu notu.

- V prvním sloupci se nachází její počátek, v jednotkách taktů – tedy 1.5 znamená, že nota začíná v polovině druhého taktu.
- Druhý sloupec obsahuje trvání noty, též v jednotkách taktů.
- Třetí sloupec vyjadřuje číslo kanálu, neboli „stopy“ kterého je nota součástí.
- Čtvrtý sloupec je pro nás nejdůležitější – jde o tzv. “MIDI note number“, tedy číslo noty, vyjadřující její pozici na škále 10 oktáv (viz. *Tabulka 1*).
- Pátý sloupec obsahuje tzv. „velocity“, rychlost. S tímto údajem však pracovat nebudeme.
- Šestý sloupec reprezentuje počátek noty v sekundách.
- Sedmý sloupec reprezentuje délku trvání noty v sekundách.

Poslední dva sloupce tedy reflektují údaje prvních dvou, nicméně v absolutních časových hodnotách, oproti relativním hodnoty prvních dvou sloupců reprezentujících čas v taktech. Výšku a trvání tónu jsem vyznačil barevně, jelikož jsou to nejdůležitější údaje, se kterými pracujeme.

Oktáva	MIDI Note numbers											
	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
-1	0	1	2	3	4	5	6	7	8	9	10	11
0	12	13	14	15	16	17	18	19	20	21	22	23
1	24	25	26	27	28	29	30	31	32	33	34	35
2	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59
4	60	61	62	63	64	65	66	67	68	69	70	71
5	72	73	74	75	76	77	78	79	80	81	82	83
6	84	85	86	87	88	89	90	91	92	93	94	95
7	96	97	98	99	100	101	102	103	104	105	106	107
8	108	109	110	111	112	113	114	115	116	117	118	119
9	120	121	122	123	124	125	126	127				

Tabulka 1: MIDI Note numbers - čísla MIDI not [12]

3.4 Extrakce melodie

Jelikož trénování sítě probíhá na úsecích melodií, potřebujeme data nějakým způsobem serializovat, a převést na monofonní podobu. Nástroj, pomocí kterého melodie budemeextrahovat zmiňuji v sekci *3.1.5*. Pro automatizaci extrahování a normalizaci vyextrahovaných melodií jsem vytvořil skript `bc_batchExtract.m`, který se užívá v následující formě

```
batchExtract('directory');
```

Výstupem je adresář „melodies“, který je vytvořen uvnitře adresáře „directory“, a do něho jsou uloženy MIDI soubory, obsahující melodie vyextrahované ze všech MIDI souborů, nacházejících se v adresáři „directory“. Tyto melodie jsou dále “normalizovány“ pomocí skriptu `bc_normalizemel.m` – tzn. kvatnizovány[6] na úseky o délce 1/16 sekundy, převedeny do tóniny C (dur či moll, v závislosti na původní tónině), a jejich tempo je nastaveno na 120bpm.

Je nutné zmínit, že ač zmíněné skripty poskytují dosti pokročilý způsob extrakce [11] můžeme nalézt hlubší rozbor existujících metod), zvláště u složitějších skladeb se stává, že výsledná vyextrahovaná stopa nekopíruje přesně melodickou linku (která může být někdy obtížně identifikovatelná i lidským „uchem“), ale obsahuje mírné odchylky směrem k notám harmonického doprovodu. Dalším problémem je, že výstupem mnohdy bývá polyfonní sekvence, která mimo melodické linky obsahuje i občasné harmonické přízvuky, což pravděpodobně vyplývá z nedokonalosti algoritmu. Bohužel takovéto melodie se nedaly vhodně použít, proto jsem byl nucen značně slevit ze svých nároků na trénovací data, jak zmiňuji v sekci *3.2*.

6 kvantizace – sjednocení počátků taktů, a délek jednotlivých not pomocí určeného koeficientu

3.5 Reprezentace trénovacích a výstupních dat

Otázku stanovení reprezentace trénovacích a výstupních dat můžeme rozdělit na dvě podčásti: jak stanovit vhodné kódování výšky a trvání tónu? A jaký bude vhodný formát těchto dat pro neuronovou síť?

Abychom mohli zodpovědět otázku natolik dobře, abychom byli schopni vybrat v našem kontextu „nejlepší“ reprezentaci, bylo by třeba do hloubky prozkoumat jak problematiku vlivu výběru formátu dat pro neuronovou síť na její výkon, tak i konkrétní specifika reprezentace hudebních dat. Tato samotná problematika by vydala na samostatnou práci. Proto jsem se ustálil na reprezentaci, zmíněné v dalších sekcích, podložené materiály z dalších studií.

Jeden problém který však zmíním, a který může provázet nevhodnou volbu vstupních dat pro neuronovou síť, je ztráta informací na základě chyb při zaokrouhlování. Aktivační funkce neuronu nabývají typicky výstupních hodnot -1.0 – 1.0 (v případě tangent sigmoidální) či 0.0 – 1.0 (v případě logsigmoidální), a vstupní hodnoty jsou zobrazovány na tento interval. Čím vyšší počet možných hodnot tedy pro vstup neuronu použijeme, tím menší jsou dílčí intervaly jeho „stavů“, a tím vyšší jsou možné ztráty při zaokrouhlování. Proto je tedy vhodnější volit spíše nižší rozsah vstupních hodnot [13].

3.5.1 Kódování výšky tónu

Hudební data jsou na formát vstupu a výstupu zvláště citlivá – u většiny druhů dat, se kterými se pracuje a jichž např. používáme pro predikci, můžeme obecně říci, že velikost chyby lineárně (či jiným funkčním zobrazením) roste s velikostí odchylky od požadované hodnoty. Nicméně to o reprezentaci výšky tónu v kontextu melodie říci nemůžeme. Ona hodnota výšky tónu, totiž v kontextu melodie má svůj vlastní význam, a odchylka stejné velikosti může mít v různých místech melodie, a tedy kontextech, různý význam.

Toto ilustruji na příkladu: Mějme neuron, na jehož výstupu očekáváme tón G4, který má hodnotu 67 v MIDI tabulce not. Vzhledem k šumu, nedokonalosti sítě, zaokrouhlování atp. však dostaneme na výstupu např. hodnotu 70.123, reprezentující po zaokrouhlení notu G#4. Tedy, v rámci melodie v C dur se najednou z dominanty stává tón nedoškálný[7], pravděpodobně bez harmonického významu. Z tohoto pohledu je tedy chyba o velikosti jedné jednotky naší reprezentace kontextuelně mnohem větší, než kdyby výstupem bylo například F4, které má sice jinou harmonickou funkci, nicméně tón už je alespoň doškálný, a melodii tolik nenaruší. Z tohoto důvodu existují různé způsoby kódování výšky tónu, které alespoň do určité míry tento problém řeší či obchází a ty zmíním dále.

Nejprve ale zmíním možné reprezentace výšky tónu obecně, ne z pohledu vstupu pro neuronovou síť. Tyto jsou dle [8] spektrální reprezentace, intervalová reprezentace, na základě výšky tónu, a reprezentace na základě tzv. „pitch class“ sad.

Tato dělení obsahují jak absolutní, tak relativní reprezentace. Absolutní například jejím MIDI note number (viz. *Tabulka 1.*). Relativní reprezentace by znamenala označit každou notu číslem, představujícím její vzdálenost od noty minulé – intervalová reprezentace. Tento způsob reprezentace už v sobě zahrnuje určitou informaci „navíc“ oproti reprezentaci absolutní – v jednom vstupu už totiž máme zahrnut „vztah“ 2 not, kterýžto sám o sobě už je na této úrovni nositelem určité elementární lokální harmonické informace. Na druhou stranu tato reprezentace přináší problém v tom, že pokud je chybně určen jeden interval, již jsou pak chybně posunuty i všechny další následující intervaly, které na tomto intervalu staví, což může představovat velký problém, jehož řešení může být například vložení určitých kontrolních míst, ve kterých vždy dojde k nápravě. Variantou intervalové reprezentace je tzv. invariantní intervalová reprezentace, kdy jsou hodnoty počítány vzhledem k předurčené notě daného módu. Těchto intervalových způsobů reprezentace však v této práci využívat nebudu.

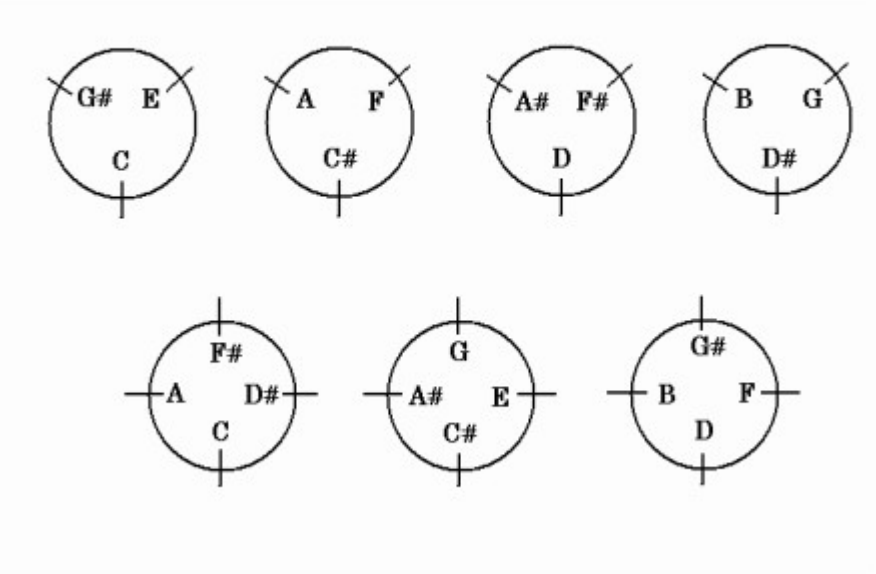
7 nedoškálný – není součástí stupnice

Jedním ze způsobů kódování, které zamezuje tomuto druhu chyb je užití kombinace chromatického a kvintového kruhu, kdy 6 bitů reprezentuje pozici tónu na chromatickém kruhu, a 6 na kvintovém kruhu. Výhoda tohoto kódování spočívá mimo jiné v zachování invariance intervalů při transpozici [4].

3.5.1.1 Kódování „Circle of thirds“

Tento způsob kódování ve své práci zmiňuje např. [5], či [6]. Při empirickém ověřování došly studie k závěru, že pro užití s neuronovými sítěmi jde o efektivnější reprezentaci, než výše zmíněné reprezentace. Jde o kódování, kdy je oktáva reprezentována samostatnou hodnotou, a pozice tónu je určena „souřadnicemi“ kruhu velkých a malých tercií, jejichž je součástí. Velké a malé tercie skládané na sebe totiž reprezentují způsob tvorby akordů, a tedy při tomto kódování „vedle sebe“ leží víceméně souzvučné tóny.

Toto kódování dále označuji zkratkou „OTT“ - “O“ pro oktávu, a „TT“ pro tercie. Rozsah podporovaných oktáv jsem se rozhodl z důvodu lepší rozlišitelnosti hodnot omezit na 2 až 6. Číslo oktávy je kódováno jakožto hodnota v rozsahu 0 – 1, kdy 0 reprezentuje druhou oktávu, 1 šestou oktávu, a oktávy mezi tím po krocích 0.25. 0.5 je tedy například čtvrtá oktáva.



Obrázek 6: Terciové kruhy - velké tercie nahoře, malé dole (převzato z [8])

Samotná hodnota tónu (bez zahrnutí oktávy) je vyjádřena pomocí 7bitového kódu, kdy první čtyři bity reprezentují pořadí kruhu velkých tercií, a následující tři bity pořadí kruhu malých tercií, jehož je nota součástí. Výhodou tohoto kódování je, že pokud nastane chyba v jednom bitu, bude výsledný tón stále ještě malou nebo velkou tercii od správného tónu, a nenastane tedy např. nelibozvučný posun o půltón. Níže ilustruji kódování na příkladu a obrázku.

Tón		Oktáva	Kruh v. tercií				Kruh m. tercií		
C4		0.75	1	0	0	0	1	0	0
A#3		0.5	0	0	1	0	0	1	0

Tabulka 2: Úkázka kódování pomocí "Circles of thirds"

3.5.2 Kódování délky tónu

Nejjednoduším způsobem, jak zahrnout délku tónu do vstupu, se jeví opakování dané noty v daném kódování po zvolený počet časových úseků. Tedy například čtvrťová nota C4 by byla při kvantizačním intervalu 1/32 reprezentována jejím osmerým opakováním (60 60 60 60 60 60 60 60). Tento způsob by však znamenal, že samotná jedna nota „sežere“ při trénování několik (v tomto případě 8) trénovacích cyklů (epoch), což může amplifikuje zmíněný problém „error decay“ - a tedy zvětšením intervalů mezi „význačnými“ událostmi omezuje schopnost sítě postihovat strukturu ve větším měřítku [4].

Dalším způsobem kódování může být prostá reprezentace délky trvání noty v sekundách. Tento přístup není příliš vhodný z toho důvodu, že výstupní hodnoty sítě v tomto formátu budou nabývat víceméně libovolných hodnot v rámci určeného rozsahu, a to není pro zachování rytmiky vhodné.

Jedním z dalších běžných typů je tzv. *„localist“* reprezentace (nepodařilo se mi najít vhodný český překlad, proto budu používat pojem v originálním znění) [5]. Ta spočívá v přiřazení vlastního vstupu/bitu každé délce noty, kterou můžeme reprezentovat. V tomto případě je tedy opět vstupem pro síť vektor binárních hodnot. Tedy například jednička na první pozici může reprezentovat celou notu, na druhé pozici třičtvrťovou a podobně. Rozlišení dělení závisí na nás, zda například chce me dokázat rozlišit a reprodukovat trioly.

Dalším způsobem, používaným je tzv. modulární reprezentace. Název modulární vystihuje její charakter – využívá totiž operace modulo. U tohoto způsobu je délka doby rozdělena na *n* krátkých časových úseků – např. 96. Každá délka noty je reprezentována počtem těchto úseků. Čtvrťová doba tedy trvá 96 těchto jednotek, tříčtvrťová 288 atp. Nejprve je tedy stanoven vektor modulo-dělitelů, a pomocí něho je určována pozice bitu pro danou délku noty. V našem příkladu vypadá vektor následovně:

```
[ 384, 288, 192, 96, 64, 48, 32, 24, 16, 12, 8, 6, 4, 3, 2, 1 ]
```

Jde postupně o reprezentace délky noty celé, tříčtvrťové, půlové, čtvrťové, osminové s tečkou, osminové, osminové trioly, šestnáctiny, šestnáctinové trioly, dvaatřicetinové, a zbytek (od šestky níže) pro doplnění, případně zachycení ještě kratších úseků.

V případě volby n = 96 reprezentace tedy zabírá celkem 16 bitů. Jelikož celá nota (a tedy jeden takt) trvá 96*4 = 384 jednotek, první bit nabývá hodnoty 1, pokud jde o notu celou (délka_noty %[8] 384) / 288 >= 1. Podobně bit 2 reprezentuje notu tříčtvrťovou (půlová „s tečkou) – *(délka_noty% 384 % 288) / 192 >= 1* a takto dále podle tohoto pravidla, až jsou postihnuty všechny hodnoty vektoru.

Výhodou konkrétně takto zvoleného *n* je jeho vysoké rozlišení – tedy že dokáže postihnout i nuance lidské intepretace – například kdybychom používali trénovací data z MIDI kontroléru, na který by hrál „opravdový“ muzikant. Další výhodou je, že je to násobek jak 4, tak i 3, a tedy dokáže postihnout mimo obvyklé délky reprezentované mocninami 2 a jejich kombinacemi i trioly, sextoly a podobně. Základ popisu této reprezentace jsem převzal z [8].

I jsem se rozhodl vyzkoušet implementaci tohoto způsobu, avšak v podobě s nižším rozlišením – s rozdělením jedné doby na 16 úseků, a s 8 možnými délkami, kterých může nota nabývat. Vynechal jsem pro zjednodušení trioly, dvaatřicetinové noty, a některé další možné délky, zmíněné výše. Reprezentace je tedy pokryta 8bitovým vektorem, kde aktivní bit reprezentuje postupně notu

8 % - operátor modulo

celou, tříčtvrťovou, půlovou, čtvrťovou s tečkou, čtvrťovou, osminovou s tečkou, osminovou, a šestnáctinovou. Pro převod na tuto reprezentaci slouží skript `bc_todurreps.m`.

Bohužel jsem se poté při manipulaci s MIDI soubory setkal s problémy, týkající se nekonzistentní reprezentace délek not (například při absenci relativních hodnot, a přítomnosti pouze absolutních, časových), či tempa, kdy nebylo možné jednoduše data normalizovat, a určit délky not – například se v reprezentaci i po kvantizaci vyskytovaly noty trvající 1.4 taktu, a podobně. Řešení těchto nekonzistencí a „opravování“ souborů by se časově nevyplatilo, proto jsem si alternativní reprezentaci délky noty – je jí jedna hodnota, reprezentující počet dílčích časových úseků (podobně jako je hodnota *n* zmiňovaná výše). V případě mých experimentů je to konrétně 1/16, neboť jsem se snažil pracovat s jednoduššími melodiemi, které neobsahují rozmanitější rytmickou diferenciaci. Při „zaokrouhlování“ výstupu sítě je potom následná hodnota „namapována“ na jednu z hodnot z rozsahu `[128 64 48 32 24 16 12 8 6 4 3 2 1 0]`, reprezentujícím počet šestnáctin sekundy, aby se zamezilo případným výskytům not s hodnotou například 9, 7 a podobně. 128 jakožto nejvyšší hodnotu jsem zvolil libovolně, avšak aby byla případně dostatečně vysoká pro pokrytí delky nejdelších vyskytujících se not. Skripty pro manipulaci s maticí not přijímají parametr `bDurations`, který reprezentuje právě volbu použitého kódování – hodnota 0 pro matice bez reprezentace rytmu, 1 pro modulární reprezentaci a 2 pro tuto alternativní. U většiny svých experimentů používám hodnotu 2.

3.5.3 Formát dat pro neuronovou síť

Jak nejlépe tedy předat neuronové síti už „zakódovaná“ data? Zvolit vlastní síť zvlášť pro rytmus i melodii, nebo zkombinovat tyto hodnoty do vstupů jedné sítě?

Jením z možných způsobů reprezentace vstupu pro jednotlivé epochy je rozdělení vstupní melodie na „okénka“. Jak již jsem zmínil v sekci pojednávající o neuronových sítích, statické typy neuronových sítí si „nepamatují“ vztah mezi postupně následujícími vstupy, a svůj výstup odvozují pouze z aktuálního vstupu. Tuto „paměť“ můžeme použitím okének do určité míry nasimulovat. Princip je následující: melodii rozdělíme na okénka o *n* notách, přičemž každé následující okénko je utvořeno posunutím jeho začátku o jednu notu. Tedy pro $n = 3$ budou první tři okénka z melodie `[C D E F G]` vypadat následovně: o_1 = `[C D E]`, o_2 = `[D E F]`, o_3 = `[E F G]`. Tento způsob reprezentace vstupních dat bývá nazýván mimo jiné „sliding time window“, tedy „posuvné okénko“. Tímto způsobem síti dodáváme další notu sekvence vždy notu v kontextu předcházejících *n* not, a zachováváme tak do určité míry kontextuální konzistenci.

Pro účel konverze na tento formát jsem vytvořil skript `bc_tospans.m`, který jako argumenty přijímá matici ve tvaru OTT a délku okénka, a vrací 2 hodnty: „cell-array” (specifický druh pole v Matlab skriptovacím jazyce) obsahující vstupy naformátovaná do daných okének, a matici korespondujících výstupů (tj. pro každé okénko o *n* notách jedna „trénovac“ nota, reprezentující tón následující po poslední notě v okénku). Bohužel, tento skript v experimentech popsaných v této práci nakonec nenalezl uplatnění, neboť Matlab odmítl výsledné pole s okénky akceptovat, i přes jeho naformátování dle dokumentace. Tento problém zmíním v sekci s experimenty.

3.5.4 Převod dat na danou reprezentaci a zpět

Pro účel převedení dat na reprezentaci [PH D] a [O Tv Tm [D]] jsem vytvořil krátké skripty `bc_nmat2phd.m` a `bc_nmat2ott.m` respektive. Oba jako parametr přijímají matici not ve formátu MIDI Toolboxu, a jako druhý parametr boolean příznak, zda do výchozího vektoru zahrnout i trvání tónu (0 – nezahrnout, 1 – zahrnout v modulární formě, 2 – zahrnout v mnou používané

alternativní formě). Skript `nmat2phd.m` pouze vyjme z matice sloupce 4 a 7 (výšku noty a její trvání) a sloučí je dohromady. Tuto reprezentaci jsem používal spíše pro „hrubší účely“ - tedy např. při prvotním testování a experimentování se sítí, nebo když jsem narychlo chtěl shlédnout, jak se liší kontura vstupní melodie od natrénované atp.

Funkce `bc_nmat2ott.m` převádí matici not do formátu `[O Tv Tm [D]]`, který popisuji v předchozí sekci. Výstupem je tedy matice vektorů ve formátu:

```
mat =
[
      0.25    0.0    0.0    1.0    0.0    0.0    0.0    1.0    [4.0]
      0.5     1.0    0.0    0.0    0.0    0.0    1.0    0.0    [4.0]
]
```

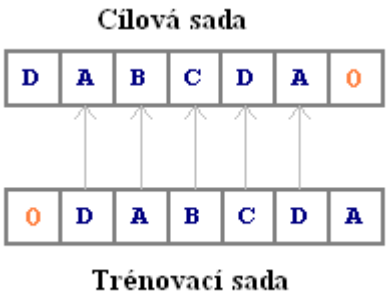
Poslední hodnota, v hranatých závorkách reprezentuje volitelné pole pro hodnotu délky tónu. Je třeba podotknout, že výsledná výstupní matice málodky vrací přesně hodnoty 1 a 0 – většinou se nechází hodnoty někde mezi, a k těmto hraničním hodnotám se blíží. Proto je nutné ještě před převodem zpět na matici not hodnoty zaokrouhlit. Zaokrouhlení probíhá samostatně pro sloupec oktávy, který je zaokrouhlen na jednotlivé desetiny, a pro zbylé sloupce. Z druhého až pátého bitu, reprezentující pozici na terciovém kruhu je vybrána nejvyšší hodnota, ta je zaokrouhlena na 1 a ostatní na 0. Ta samá procedura je provedena pro sloupce šest až osm. Při převodu na OTT reprezentaci provádí skript ještě zároveň normalizaci melodie.

Převedení OTT matice zpět na MIDI toolbox formát notové matice je prováděn pomocí skriptu `bc_ott2nmat.m`, jenž opět za parametry přijímá matici, a typ reprezentace délky tónů.

3.5.5 Trénovací a testovací data

Klasický přístup, kdy jsou data rozdělena na 2 nebo 3 části (trénovací, testovací a případně validační) je zde vzhledem k zachování celistvosti melodie nevhodné použít. Proto jsem zvolil přístup, kdy je na začátek celé melodie, přiváděné na vstup, přidána „nulová nota“, stejně tak jako na konec „cílové“ melodie.

Cílová melodie, oproti které síť svůj výstup testuje, a na základě něhož počítá chybu a zpětně se modifikuje, je posunutá o jednu notu „doleva“ oproti trénovací, aby bylo zachováno mapování *n*-tá nota vstupní melodie → *n+1* nota “cílové“ melodie – tedy síť se z *n*-té noty učí předpovídat notu *n+1*.



Obrázek 7: Posunutí trénovací a cílové sady

4 Vlastní experiment

Rozhodl jsem se při experimentu zaměřit hlavně na síť typu ESN, nicméně jsem v rámci předpřípravy zkoušel i ostatní druhy sítí (dopředná, NARX, LSTM), a v implementaci zahrnuji skripty pro manipulaci s nimi, proto zde zmiňuji krátce i popis těchto pokusů.

4.1 Experiment s ESN sítí

Jak jsem se již zmínil v teoretické sekci, ESN sítě nebyly zatím ve větší míře studovány v kontextu generování melodií. To mne motivovalo se o nich dozvědět trochu více. Jednou z hlavních „nástrah“ těchto sítí je, vzhledem k diverznímu charakteru jejich možné dynamiky, i velká citlivost výstupu na na vstup, tak na nastavení různých parametrů sítě. To platí u neuronových sítí obecně, nicméně u tohoto typu „dvojnásob“. Sám Herbert Jaeger, jeden z průkopníků v oblasti výzkumu těchto sítí říká: „Úspěšná aplikace ESN přístupu tedy zahrnuje dobrý úsudek co se týče důležitých charakteristik dynamiky excitované v DR. Tento úsudek však může růst pouze s osobní zkušeností experimentátora.“ [14]

4.1.1 Nastavení a vlastnosti simulace

Pro účel experimentu jsem vytvořil skript `bc_example.m`, který je zjednodušeným „rozhraním“ pro testování sítě s různými parametry. Lze v něm manipulovat s následujícími proměnnými:

```
NUM_NEURONS = 12;              % Počet neuronů
NEURON_STEP_SIZE = 12;         % Velikost kroku při postupném navyšování
MAX_NEURONS = 49;              % Maximální počet neuronů
LEAKY_INTEGRATOR = 1;          % Zda použít metodu Leaky-integrator
ITERATIONS = 24;               % Počet iterací pro iterativní generování
RADIAL_BASIS = 0.5;            % RB parametr ESN neuronů
SEED_LENGTH = 8;               % Délka "seed" sekvence
FILENAME = 'training/mel_elise.mid';
TEST_FILENAME = 'training/chroma2oct_trans.mid';
SEED_DATA = bc_getseed('training/mel_elise.mid');
```

Skript 1: Experimentální proměnné

`NUM_NEURONS` značí počet vnitřních neuronů rezervoáru. `NEURON_STEP_SIZE` a `MAX_NEURONS` využijeme v případě, že chceme celou simulaci provést několikrát za sebou, a to pokaždé s jiným počtem neuronů – zvýšeným vždy o `NEURON_STEP_SIZE`. `LEAKY_INTEGRATOR` určuje, zda síť pro učení použije algoritmus leaky-integration. Proměnná `ITERATIONS` značí počet iterací při iterativním generování melodie. `RADIAL_BASIS` je parametr aktivační funkce neuronu ESN, `FILENAME` označuje soubor, ze kterého se načtou trénovací a cílová data, `TEST_FILENAME` „testovací“ soubor (účel popíši dále), a `SEED_DATA` reprezentuje data ve formátu OTT, která budou použita jako výchozí krátká sekvence („seed“) při iterativním generování.`SEED_LENGTH` značí jejich délku.

Funkce `bc_getseed.m` slouží k tomuto účelu – vrátí určený počet not z náhodného či určeného místa v daném souboru. Skript na základě dodaných parametrů generuje melodii třemi způsoby. A to:

1) Generování na základě trénovacích dat
2) Generování na základě odlišné, „testovací“ sady dat
3) Iterativní generování

První způsob znamená, že síti natrénované na trénovací sadě je v simulační fázi opět na vstup dodána trénovací sada. Výstup tohoto způsobu bývá nejvíce blízký (v některých případech totožný) s vstupními daty – síť se ho v různé míře„naučí“.

Druhý způsob od prvního liší pouze daty, která síti poskytneme na vstup – jde již o odlišná data od trénovací sady, může jít o jinou melodii, sekvenci náhodných tónů atp., nicméně i zde platí mapování 1:1 – tedy jedna vstupní nota na jednu výstupní.

Třetí způsob je pravděpodobně potenciálně nejzajímavější, nicméně také nejméně stabilní a nejcitlivější na nastavení všech možných parametrů. Spočívá ve spuštění simulace na krátké sadě dat („seed“), přičemž při každé iteraci je na její konec přidána poslední nota posledního výstupu, a v dalším cyklu je již vstupní sada o tuto notu delší – sekvence tedy roste po `ITERATIONS` kroků. *Skript 2* je ukázkou zdrojového kódu pro tento cyklus.

```
if (numIterations)
   for (k = 2:numIterations)
      % disp(recurrentOutput);
      tmp = test_esn(iteratedOutput, trainedEsn, 0);
      tmp = tmp(end, :);
      tmp = bc_roundott(tmp, 2);
      iteratedOutput = vertcat(iteratedOutput, tmp);
   end
end
```

Skript 2: Ukázka kódu pro iterativní generování výstupu

Po spuštění skriptu, kdy proběhne natrénování a následná simulace, jsou na obrazovku vykreslena 3 okna. První, s nadpisem „Trénovací fáze“ obsahuje modrou barvou vyvedený graf kontury trénovací melodie, a červenou barvou konturu výsledné melodie, kterou síť vygeneruje z trénovací sady dat, tedy 1. způsob vygenerování melodie. Ve druhém okně, s nadpisem „Testovací fáze“ se nachází téže dva grafy, modrá reprezentuje konturu „testovacích dat“ na vstupu, červená opět výstup sítě. Třetí okno obsahuje pouze zelenou křivku kontury dosažené iterativním generováním. Alternativně, při provedení simulace s postupným navyšováním neuronů, se otevře okno se dvěmi křivkami – průměrná RMSE trénovací (zeleně) a testovací fáze (červeně) v závislosti na počtu neuronů v síti.

Všechny tři výstupy jsou uloženy do složky „output“, s názvem souboru `{train, test, iterated}_x.mid`, respektive, kde x reprezentuje pořadí simulace, v případě že provádíme cyklus s postupným navyšováním neuronů.

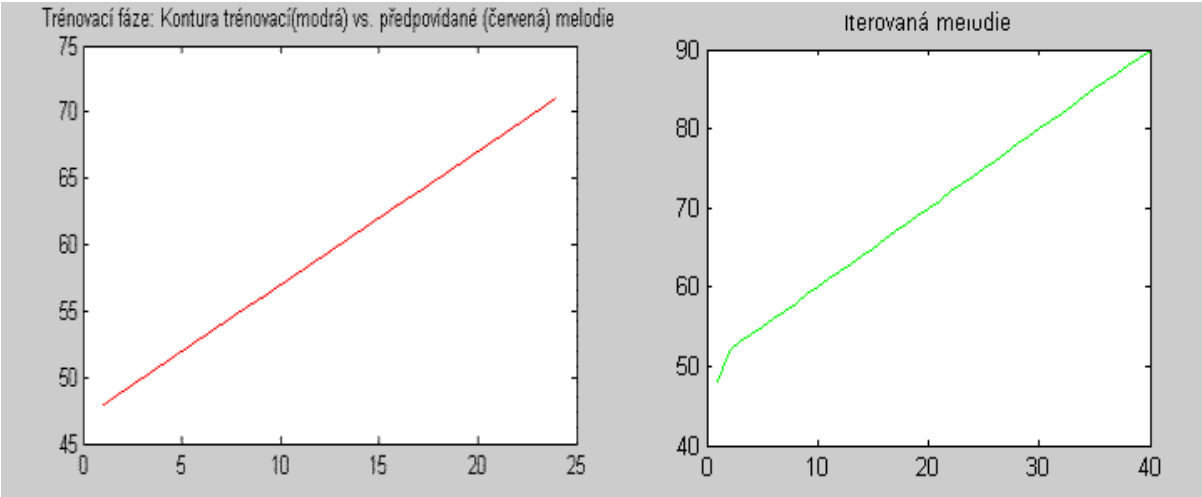
Ač simulace nevypisuje časové údaje, trénování probíhá znatelně rychleji, než u „klasických“ sítí se srovnatelným počtem neuronů ve skryté vrstvě – i v případě desítek až stovek neuronů řádově v sekundách.

4.1.2 Jednotlivé experimenty

Rozhodl jsem se síť testovat postupně – nakoli je schopna si poradit s jednodušším úkolem, při jakých nastaveních (neuvádím zde všechna, která jsem zkoušel, pouze ta vhodnější, na která jsem přišel). Předem podotýkám, že když píši „síť se byla schopna naučit pasáž *x* při konfiguraci s *n* neurony“, jde spíše o orientační údaj – „rychlost“ učení kolísala od simulace k simulaci, v závislosti na náhodné počáteční konfiguraci rezorvoáru.

4.1.2.1 Iterovaná reprodukce chromatické řady

Nejprve jsem chtěl vyzkoušet, zda bude síť schopná vůbec schopná „naučit“ se alespoň jednoduchou chromatickou řadu. Neměl jsem příliš představu o potřebném počtu neuronů, proto jsem spustil simulaci s parametry `NUM_NEURONS = 2; NEURON_STEP_SIZE = 5; MAX_NEURONS = 33;`



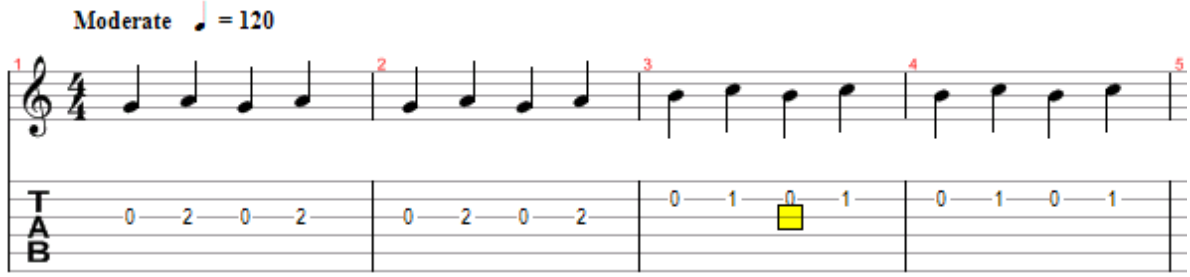
Obrázek 8: chromatická řada, 7 neuronů - trénovací fáze vs. iterovaný výstup

jako trénovací soubor zvolil „chroma2oct.mid“, testovací „chroma2oct_mid.mi“ a „seed“ soubor „ chroma_seed_trans.mid” (což je soubor s výsekem chromatického postupu, transponovaným o 4 půltóny, pro ověření, zda si síť poradí s takovouto transformací).

Jak je vidět z obrázku, již při 7 neuronech byla síť schopna se „naučit“ (červená linie přesně překrývá modrou) a zobecnit – zelená linie pokračuje po ose y dále, než původní trénovací data (na ose Y je vynesena výška noty, jednotkou je MIDI note number, dle tabulky v úvodní sekci). “Zalomení“ linie dole vyplývá z transponované „seed“ sekvence. Výstupem je soubor „iterated_7.mid”.

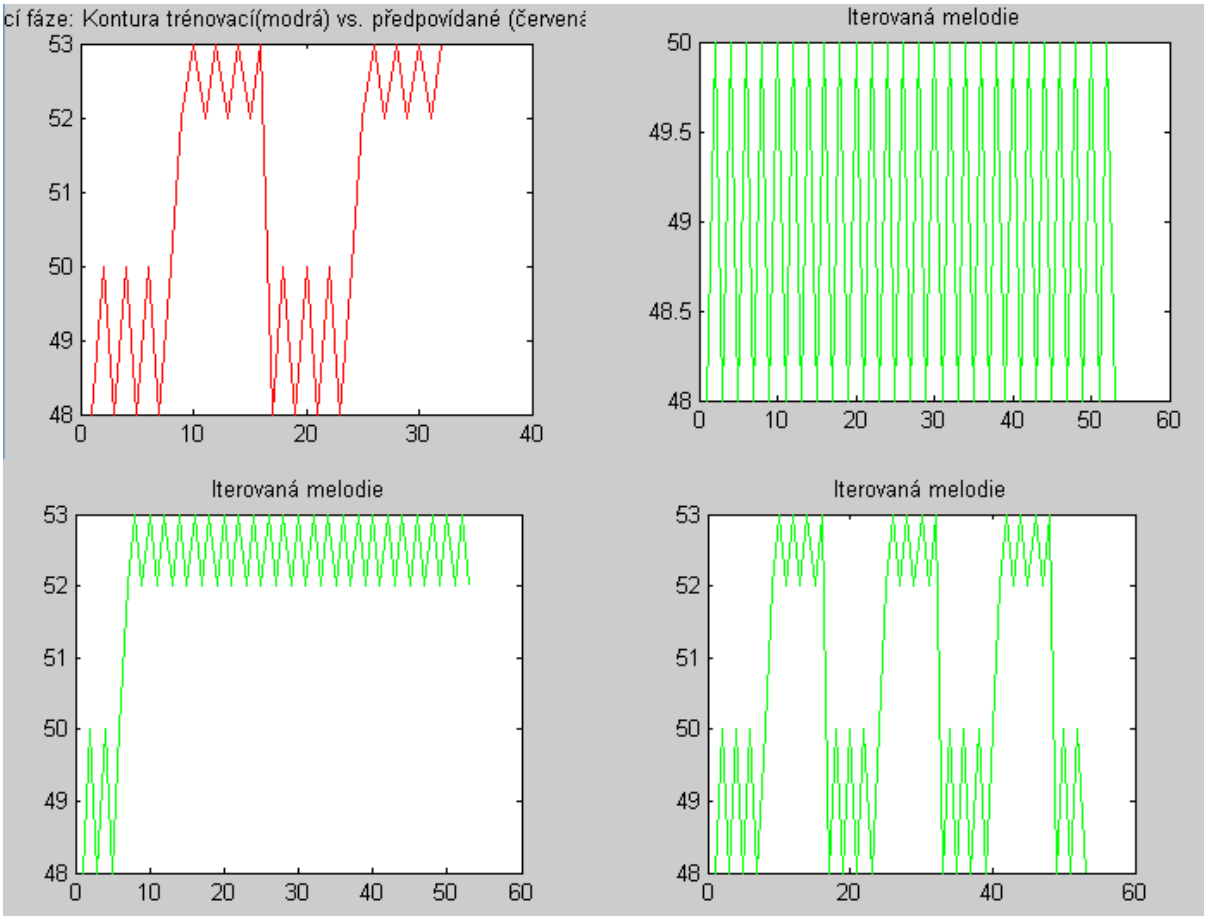
4.1.2.2 Iterovaná reprodukce AABB formy

Takováto „složitější“ forma už bývá pro některé jednodušší typy sítí těžkým oříškem, proto tuto formu otestuji.



Obrázek 9: Jednoduchá AABB forma

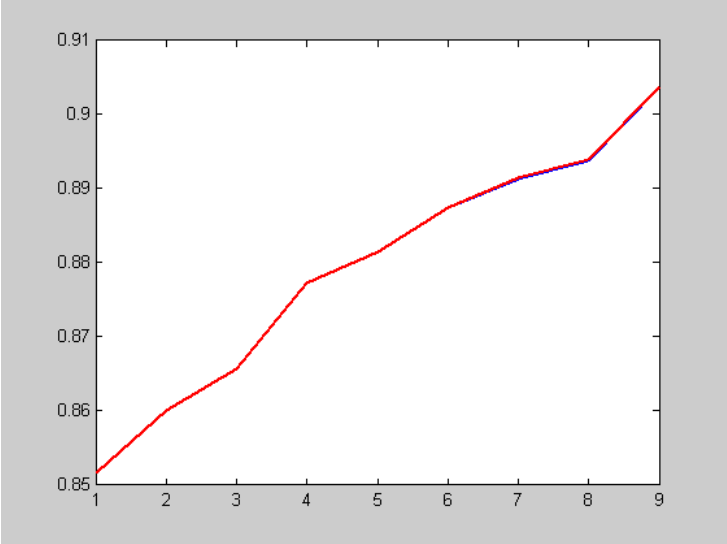
Trénovací soubor „`aabbb.mid`“ obsahuje jednoduchou „melodii formy AABB. Síť se ji byla schopna naučit již zhruba při 20 neuronech, a dokázala ji i zobecnit v iterativním generování (někde zhruba okolo 40 neuronové konfigurace). Tento pokus ukazuje na přítomnost určité formy krátkodobé paměti v síti, jelikož je schopna reprodukovat časově členitou formu i při generování „notu po notě“ .



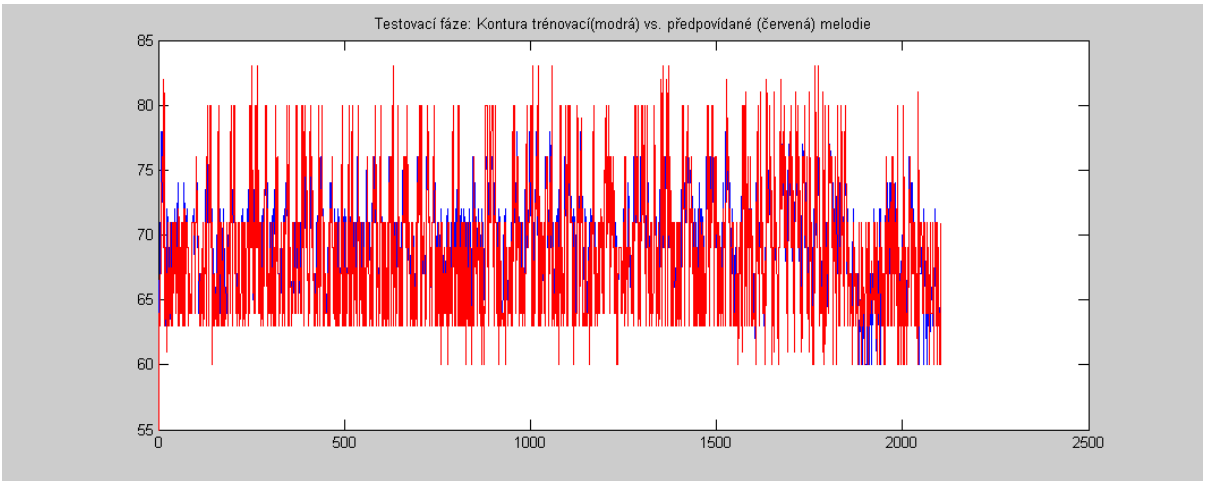
Obrázek 10: AABB - původní sekvence; iterovaná sítí o 15, 30 a 40 neuronech (zleva do prava)

4.1.2.3 Reprodukce z velkého souboru dat

Součástí MIDI Toolboxu je soubor 50 jednoduchých finských melodií, které jsem sloučil do jednoho dlouhého MIDI souboru, který se skládá zhruba z 2200 not. Provedl jsem experiment, kdy jsem nejprve zvyšoval počet neuronů po desítkách, a poté po dvacítkách, a síť trénoval na tomto datovém souboru.



Obrázek 11: RMSE na trénovacích datech po 8 simulací, počáteční počet neuronů: 8, krok: 10 neuronů

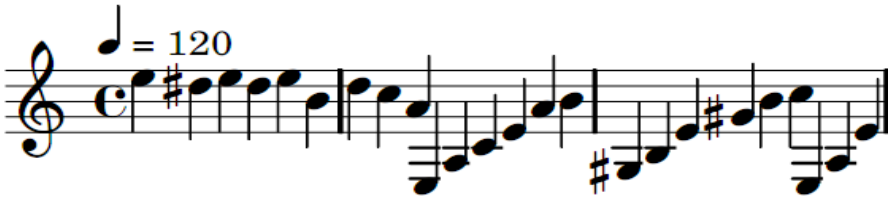


Obrázek 12: Soubor melodií; modře původní kontura, červeně generovaná sítí

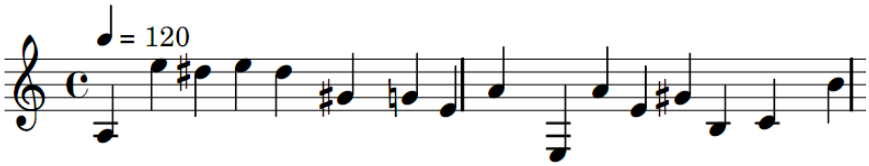
Vzestupný trend křivky ukazuje na neefektivní trénování sítě. Tento trend pokračoval i při dalším zvyšování počtu neuronů. Z grafu i poslechu výsledné melodie vzniklé prvním způsobem generování (z trénovacích dat) bychom mohli identifikovat v některých rysech určitou podobnost s trénovacími daty, nicméně při iterativním generování se bohužel téměř vždy melodie zacyklila, či uvázla na jednom tónu.

4.1.2.4 Reprodukce melodie „Fur Elise“

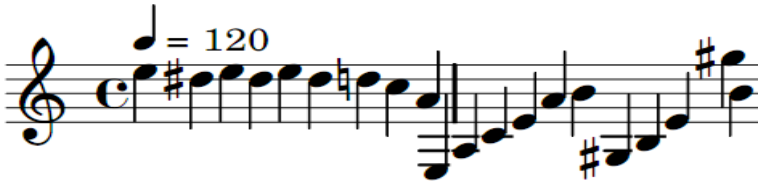
Pro ilustraci, a abych také nepoužíval jen možná málo vypovídající kontury uvedu ještě výsledky sítě při učení se Beethovenovy skladby „Fur Elise“. Zahrnuji pouze vždy první dva takty, jelikož poměrně dostatečně zachycují přesnost zachycení melodie celkově. Jde o melodie reprodukované postupně sítěmi o 5, 35 a 60 neuronech. I přes roztodivné zobrazení na notové osnově, způsobené nepřesnými hodnotami not na výstupu, a tedy působící problémy notovacímu programu je vidět (či slyšet – soubory přikládám na CD), že (až na případ o 5 neuronech) se melodie původní verzi velmi blíží, a případné chyby mohou být vnímány spíše jako zpestření, či „improvizace“, neboť dle mého názoru skladbu výrazně nenarušují.



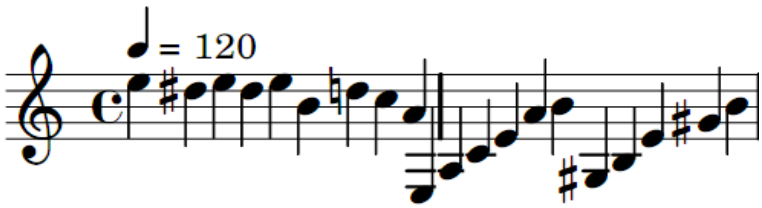
Obrázek 13: Fur Elise - původní verze



Obrázek 14: Fur Elise - výstup sítě o 5 neuronech



Obrázek 15: Fur Elise - výstup sítě o 37 neuronech



Obrázek 16: Fur Elise - výstup sítě o 60 neuronech

4.2 Nezdařené experimenty

4.2.1 Dopředná síť

Neural Network toolbox sice poskytuje základní GUI pro tvorbu základních typů sítí, nicméně není příliš flexibilní, a nenabízí tolik možností úprav sítě pro experimentální účely. Proto jsem tuto síť implementoval ručně ve skriptu `bc_mynet.m` - od definování vrstev a jejich propojení, přes stanovení vah, až po nastavení filtrovacích funkcí a podobně.

To jsem však netušil, že mi NN Toolbox užívá odlišné syntaxe zápisu vstupních dat do vstupních neuronů v případě, že jich je více než jeden. Tato možnost je v sice „plně podporována“, nicméně v dokumentaci je zmíněna velmi spoře, na nezobecnitelně malém vzorku dat, a ani na internetových stránkách Mathworks jsem odpověď nenalezl. Po marném zkoušení všech možných kombinací buněk v cell-maticích jsem se rozhodl ještě propátrat některá fóra zaměřená na Matlab, nicméně na nich jsem nalezl jen několik nešťastníků s podobným problémem.. S tímto jsem tuto větev pokusů odstřihl (a stejně tak i pokusy se sítěmi NARX, které, jak jsem zjistil, vykazovaly stejný problém), nicméně skript je, po vyřešení zmíněných problémů, případně použitelný pro další užití.

4.2.2 LSTM Síť

LSTM síť jsem chtěl otestovat hlavně z důvodu pozitivních výsledků při rozpoznávání melodie v některých provedených studiích, např. [5]. Nicméně LTSM Toolbox však bohužel poskytuje jen sporou dokumentaci, a ani jinde na internetu nejsou žádné podrobnější detaily ohledně jeho užití. Podařilo se mi sice po různých komplikacích sestavit a otestovat síť na jednoduchých datech, bohužel však síť nebyla schopna správně reagovat, a to i po rozsáhlejším experimentování s různými parametry sítě a počátečními nastaveními, proto jsem se jim už také dále nevěnoval.

4.3 Shrnutí

I přesto, že jsem se pokoušel postupovat systematicky, jsou kombinací možných parametrů s možnými vstupy obrovská množství, a jelikož ještě nemám tolik zkušeností s neuronovými sítěmi a jejich chováním obecně, bylo mé počínání často spíše náhodné, než konvergující k nějakému konkrétnímu cíli. Napadaly mne různé formáty trénovacích dat a způsoby jejich kombinování, nicméně realizovat učení pomocí nich systematicky by byl časově náročný úkol, vyžadující větších zkušeností, proto jsem zde uvedl jen pár jednoduchých. Nicméně k některým postřehům jsem i tak dospěl:

- ESN sítě jsou schopné velmi dobrého zapamatování melodie, v závislosti na délce a složitosti melodie, v konfiguraci od 10 neuronů výše, a při vstupu počátečních tónů sekvence jsou schopny si melodii „vybavit“. Při konfiguraci nad 40 neuronů už si síť zapamatovala i členitější melodie, a nad 100 s občasnými odchylkami i celé „skladby“. Tato vlastnost však souvisí také s velkou náchylností k přetrénování, přičemž síť není schopna reagovat adekvátně na nová data, a je velmi nestabilní. To může být do velké míry způsobeno malým vzorkem trénovacích melodií, které jsem měl k dispozici – pokud síť postupy a tóny v melodii zná, má menší tendenci „zaseknout se“ ve smyčce, či oscilačně „explodovat“, nicméně i při použití velkého souboru dat docházelo při iterativní reprodukci k zacyklením, což však vysvětlit zatím nedokáži.
- Dále, podle očekávání, si melodie nějakým způsobem symetrické, s vysokou autokorelací, zapamatovávala snadněji (vzhledem k jejich pravidelném časoprostorovém rozložení, ve kterém se snadněji zobecňují souvislosti).
- Změna parametru radial base neměla v rámci těchto pokusů přílišvelký vliv.
- Osvědčilo se kódování pomocí kruhu tercií – pokud se vyskytla chyba, téměř nikdy nešlo o sousední dissonantní půltón.
- Potvrdila se též hypotéza, že výstup do velké míry závisí na počátečním vnitřním stavu rezervoáru – i při zachování všech parametrů konstantních poskytovala simulace od simulace odlišné výsledky – někdy byla schopna např. schopna se sekvenci naučit „zázračně“ rychle, jindy naopak.
- Je pravděpodobné, že dlouhodobějším experimentováním s nastavením vhodných parametrů a širším spektrem testovacích dat by bylo možné dosáhnout mnohem použitelnějších výstupů.

5 Závěr

Na závěr shrnu své pozitivní i negativní zkušenosti a hodnocení v průběhu práce, a nakonec nastíním další možnosti vývoje ve směru naznačeném touto prací.

5.1 Přínosy

Co se týče přínosů, myslím, že se mi do určité míry podařilo shrnout hlavní aspekty práce s neuronovými sítěmi v kontextu generování melodií, ukázat částečně pohled „jak na to“ případnému „počítačově gramotnému“ zájemci o vlastní experimentování, a usnadnit mu případné pokusy poskytnutím některých skriptů. Také jsem zdůraznil význam předávání co nejvíce implicitních informací z melodií neuronovým sítím, a nastínil možné způsoby. Za zmínku také stojí užití a alespoň základní otestování sítí ESN v této oblasti, kde zatím nebyly příliš uplatňovány.

5.2 Co by šlo udělat lépe

Nejprve zmíním nedostatky práce, které si uvědomuji, ale již se mi nepodařilo je odstranit nebo se jim vyhnout, a jichž by bylo vhodné se v budoucích pracích vyvarovat. Většina těchto má pravděpodobně společný základ „nestihl“. Nicméně než tím, že bych „neměl čas“, je to je dáno spíše tím, že jak jsem postupně pronikal do tajů zpracování hudby neuronovými síťemi, dostával jsem nové a nové nápady, co bych mohl a chtěl do práce zahrnout, a s čím experimentovat, což způsobilo, že jsem měl „velké oči“ co se týče možného pokrytí a rozsahu. Nakonec to tedy často vypadalo tak, že jsem začal nápad zkoušet, implementovat, a poté zjistil, že měl-li bych toto všechno do práce zahrnout, překročil bych mnohanásobně jak její rozsah, tak i termín odevzdání, a tedy jsem rozdělaný kus nechal být. Z tohoto důvodu se mi nakonec pravděpodobně v rámci zadání „relevantním“, praktickým ukázkám povedlo věnovat méně času a prostoru, než bych chtěl. I přestože kladu důraz spíše na celkové uchopení problému, více názorných ukázek by práci obohatilo.

Nedostatek vidím též v trénovacích datech. Bohužel se mi nepodařilo sehnat jak dostatek žánrově podobných melodií, tak i dostatečně dlouhých melodií, a tak jsem se musel spokojit pouze s testovacími melodiemi co jsem vytvořil, a několika málo dalšími, což se také v důsledku projevilo na kvalitě výsledného výstupu.

5.3 Do budoucna

V sekci *2.3 Neuronové sítě* rozebírám melodii, jakožto datovou sekvenci, která pro člověka na první pohled a poslech nese rozmanité spektrum implicitních informací, které však neuronová síť nepostřehne. Tento fakt jsem si však začal více uvědomovat až později. Stejně tak pozdě jsem si tedy uvědomil i to, že ve většině studií, se kterými jsem se na téma generování melodie setkal, s těmito implicitními informacemi též příliš nepracují. To je tedy například jedna z cest, kterou bych se případně dál rád vydal při dalším zkoumání – zahrnování určitých sémantických, gramatických informací o struktuře, způsobů vhodných harmonických reprezentací atp.

V rámci možného vývoje projektu do budoucna, bylo by bylo také vhodné skriptům dát ucelenější formu, a z jednotlivých skriptů vytvořit celistvější „framework“ pro usnadnění experimentování, nastavování parametrů a vizualizace, kterézto nyní byly prováděny často víceméně ručně. Dále by určitě pomohlo získat onu Bachovu či jinou sadu „konzistentních“ melodií – například “ručně“ předupravit soubory do formy, ze které už by šla melodie vyextrahovat našimi skripty lépe, a bylo by možné provést smysluplnější experimenty a to třeba v kontextu širší hudební teorie.

Případně melodický výstup takovéto práce dále využít jakožto první blok nějakého „zastřešujícího systému“ (který by automaticky harmonizoval dodanou melodii, a poskytoval improvizovaný doprovod).

Samotná tematika využití ESN sítí v kontextu generování melodií by mohlo vydat na celou další samostatnou práci, a myslím, že by jistě byla přínosná.

6 Zdroje

[1] Algorithmic composition - Wikipedia, the free encyclopedia. [online], This page was last modified on 15 April 2009, at 12:23 (UTC). , [cit 12. 5. 2009].
URL http://en.wikipedia.org/wiki/Algorithmic_composition

[2] Neural network - Wikipedia, the free encyclopedia. [online], This page was last modified on 19 May 2009, at 15:53 (UTC). , [cit 12. 5. 2009].
URL http://en.wikipedia.org/wiki/Neural_network

[3] Generative music - Wikipedia, the free encyclopedia. [online], This page was last modified on 19 Jan 2009, at 10:28 (UTC). , [cit 12. 5. 2009].
URL http://en.wikipedia.org/wiki/Generative_music

[4] MOZER, Michael C. Neural network music composition by prediction: exploring the benefits of psychoacoustic constraints and multi-scale processing. *Connection Science* [online]. 1994 [cit. 2009-05-17]. Dostupný z WWW: <ftp://ftp.cs.colorado.edu/users/mozer/papers/music.ps>.

[5] CORREA, Debora C., SAITO, Jose H., ABIB, Sandra. Composing Music with BPTT and LSTM Networks : Comparing Learning and Generalization Aspects. *Computational Science and Engineering Workshops, 2008. : CSEWORKSHOPS apos;08. 11th IEEE International Conference* [online]. 2008 [cit. 2009-05-17], s. 95-100. Dostupný z WWW: <ieeexplore.ieee.org/iel5/4625018/4625019/04625046.pdf?arnumber=4625046>.

[7] ELMAN, J.L., Finding structure in time, *Cognitive Science*, Vol. 14, 1990, [cit. 2009-05-17], s. 179 -211. Dostupný z WWW: <ieeexplore.ieee.org/iel5/4625018/4625019/04625046.pdf?arnumber=4625046>.

[8] FRANKLIN, J. A., 2004, Computational Models for Learning Pitch and Duration Using LSTM Recurrent Neural Networks, in Proceedings of the 8th International Conference on Music Perception and Cognition (ICMPC8), Chicago, IL.

[9] ŠKUTOVÁ, Jolana. *Neuronové sítě v řízení systémů* [online]. 2004. Ostrava: Vysoká škola báňská - Technická univerzita Ostrava, 2004 [cit. 2009-05-14]. Dostupný z WWW: <http://www.fs.vsb.cz/books/NeuronoveSite/>.

[10] - WOOLER, R., BROWN, A. R., MIRANDA, E. R., BERRY, R. and DIERICH, J. (2005). A framework for comparison of process in algorithmic music systems. In: *Generative Arts Practice 2005 - A Creativity & Cognition Symposium*, University of Technology, Sydney, Australia, (). 5-7 December, 2005. [cit. 2009-05-17] Dostupný z WWW: <http://books.nips.cc/papers/txt/nips10/0887.txt>

[11] OZCAN, Gyiasettin, ISIKHAN, Cihan. Melody Extraction on MIDI Music Files. *Proceedings of the Seventh IEEE International Symposium on Multimedia* [online]. 2005 [cit. 2009-04-25], s. 414-422. Dostupný z WWW: <http://portal.acm.org/ft_gateway.cfm?id=1107183&type=external&coll=GUIDE&dl=GUIDE&CFID=36896269&CFTOKEN=75525648>.

[12] *Table 2: Summary of MIDI Note Numbers for Different Octaves* [online]. [1988] , 1995 [cit. 2009-05-04]. Dostupný z WWW: <http://www.harmony-central.com/MIDI/Doc/table2.html>.

[13] *Neural Network Toolbox : Documentation* [online]. 2009. 2009 , 2009 [cit. 2009-05-18]. Dostupný z WWW: <http://www.mathworks.com/access/helpdesk/help/toolbox/nnet/index.htm>.

[14] JAEGER, H. Tutorial on training recurrent neural networks, covering BPTT, RTRL, EKF and the "echo state network" approach. GMD Report 159, German National Research Center for Information Technology, 2002 (48 pp.) [cit. 2009-05-12] Dostupné z WWW: <http://www.faculty.iu-bremen.de/hjaeger/pubs/utorialRev.pdf>

[15] ORR, Genevieve, SCHRAUDOLPH, Nici, CUMMINS, Fred. *CS-449: Neural Networks* [online]. 1999. 1999 [cit. 2009-05-20]. Dostupný z WWW: <http://www.willamette.edu/~gorr/classes/cs449/intro.html>.

[16] MIDI Toolbox, <http://www.jyu.fi/musica/miditoolbox/>.

[17] LSTM Toolbox, <http://idelnx81.hh.se/bioinf/index.html>.

[18] ESN Tools, <http://www.faculty.iu-bremen.de/hjaeger/pubs.html>

Seznam příloh

A Stručný návod

Toto je stručný návod jak zprovoznit skripty popsané v tomto návodu a vygenerovat testovací melodii. Pro tento návod předpokládám nainstalované prostředí Matlab (nejlépe ve verzi 7.6.0 R2008).

Pro spuštění pro tuto práci vytvořených skriptů je třeba nainstalovat následující součásti: MIDI Toobox [16], ESN Tools [18] a případně LSTM [17].

U každého z nich je uveden stručný návod k instalaci, nicméně s MIDI Toolboxem se mohou vyskytnout problémy se zápisem a čtením MIDI souborů v novějších verzích Matlabu, a je tedy třeba v sekci FAQ MIDI Toolbox tránek nalézt návod pro tento případ. Řešením je stažení 2 opravených souborů pro převod MIDI souboru do matice not a zpět (`midi2nmat.m, nmat2midi.m`)

Dále je třeba zkopírovat složku „`bc_scripts`“ též do složky „`toolbox`“ v Matlab instalačním adresáři. Následuje nastavení cest k nově nainstalovaným složkám toolboxů v Matlabu – to se provede v menu File->Set path.

Poté už je možné spustit skript bc_example, který spustí ukázkovou simulaci. Otevřeme-li si ho, můžeme manipulovat se zmíněnými proměnnými.

B Seznam skriptů

Bližší podrobnosti příkazem „help bc_xxxxx.m“ v prostředí Matlab.

```
bc_batchextract.m
bc_durrep2.m
bc_esn.m
bc_example.m
bc_fttdn.m
bc_getseed.m
bc_lstm.m
bc_midi2traindata.m
bc_mytdl.m
bc_nmat2ott.m
bc_ott2nmat.m
bc_normalizemel.m
bc_nmat2pd.m
bc_pd2nmat.m
bc_roundott.m
bc_setrowsmax.m
bc_spanff.m
bc_tospans.f
```