TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Generování harmonického doprovodu melodie pomocí neuronových sítí

Bakalářská práce

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

Autor práce: **Jonáš Malena**
Vedoucí práce: prof. Ing. Jan Nouza CSc.

Liberec 2018

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Neural networks used for generating harmonic accompaniment to melody

Bachelor thesis

Study programme: B2646 – Information technology
Study branch: 1802R007 – Information technology

Author: **Jonáš Malena**
Supervisor: prof. Ing. Jan Nouza CSc.

Liberec 2018

Technická univerzita v Liberci
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: 2017/2018

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jonáš Malena**
Osobní číslo:	**M15000042**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Generování harmonického doprovodu melodie pomocí neuronových sítí**
Zadávající katedra:	**Ústav informačních technologií a elektroniky**

Zásady pro vypracování:

1. Seznamte se s problematikou moderních neuronových sítí, zejména tzv. hlubokých a rekurentních, a s jejich využitím pro automatické generování hudby.
2. Vytvořte co nejrozsáhlejší soubor trénovacích, vývojových a testovacích dat, která budou obsahovat digitálně zapsané jednohlasé melodie různých žánrů a k nim přiřazený harmonický doprovod (sled akordů).
3. Navrhněte a implementujte programy, které budou schopny se naučit přiřadit harmonický doprovod (akordy) k dané melodii. Vyzkoušejte a porovnejte více přístupů, zejména však využití rekurentních neuronových sítí.
4. Výsledný program bude schopen po zadání melodie (a případně žánru) vygenerovat a přehrát melodii i s harmonickým doprovodem.
5. Na nezávislých testovacích datech vyhodnoťte shodu mezi vygenerovaným a referenčním doprovodem.

Rozsah grafických prací: **Dle potřeby dokumentace**

Rozsah pracovní zprávy: **cca 30-40 stran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] **Mařík V., Štěpánková O., Lažanský J. a kol.: Umělá inteligence (1), Academia, Praha, 1993.**

[2] **Bishop Ch.: Pattern Recognition and Machine Learning. Springer, 2006.**

[3] **Online kurz "Machine Learning", dostupný na https://www.coursera.org/learn/machine-learning**

[4] **A Recurrent Neural Network Music Generation Tutorial dostupný na https://magenta.tensorflow.org/2016/06/10/recurrent-neural-network-generation-tutorial**

Vedoucí bakalářské práce: **prof. Ing. Jan Nouza, CSc.**
Ústav informačních technologií a elektroniky

Datum zadání bakalářské práce: **19. října 2017**

Termín odevzdání bakalářské práce: **14. května 2018**

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

L.S.

prof. Ing. Ondřej Novák, CSc.
vedoucí ústavu

V Liberci dne 19. října 2017

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

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

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

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 14. 5. 2018

Podpis: Balan

Abstrakt

Práce se zabývá návrhem, učením a vyhodnocením rozdílných modelů, jejichž účelem bylo pro poskytnutou sekvenci tónů (melodii), vygenerovat harmonický doprovod, tvořený akordy.

Učení modelu bylo podmíněno tvorbou rozsáhlého datového souboru, složeného z melodických a harmonických dvojic. Ten byl vytvořený zpracováním velkého množství MIDI souborů, do formátu vhodného k reprezentaci neuronovými sítěmi.

Oba navržené modely byly založeny na hlubokých rekurentních neuronových sítích. První model byl diskriminativní logistický klasifikátor. Druhý, generativní model, byl založený na variačním autoenkodéru.

Dále je popsáno vyhodnocení obou modelů na nezávislých datech. To bylo prováděno pomocí testování shody mezi vygenerovanými a referenčními daty. Dále pak porovnáním vlastností melodie a vygenerovaného doprovodu.

Abstract

The aim of this thesis was to design, learn a evaluate distinct models whose purpose was to generate harmonic accompaniment consisting of chords for given melody.

The learning was conditioned by the creation of a large dataset consisting of melodic and harmonic pairs. The dataset was created by processing a large ammount of MIDI files, into the format which is more natural for neural networks.

Both models were based on deep recurrent neural networks. First model was a discriminative logistic classifier. Second model was generative and based on a variational autoencoder.

The evaluation of both models was done on independent data. It was performed using measuring similarity between the generated and the reference data. Another accuracy metric consisted of measuring properties between melody and generated data.

Obsah

1 Úvod **9**

1.1 Melodie a harmonie . . . 9

1.2 Stručná historie počítačové hudby . . . 9

1.3 Stanovené cíle . . . 10

1.4 Neuronové sítě . . . 11

1.4.1 Diskriminativní a generativní modely . . . 12

2 Související architektury neuronových sítí **13**

2.1 RNN . . . 13

2.1.1 Zpětná propagace v čase . . . 14

2.1.2 Problém mizejícího a explodujícího gradientu . . . 15

2.1.3 Hluboké rekurentní sítě . . . 15

2.2 LSTM . . . 15

2.2.1 Peephole LSTM . . . 16

2.3 VAE . . . 16

3 Datová sada **18**

3.1 Formát záznamů . . . 18

3.2 Zdrojová data . . . 19

3.3 Popis formátu MIDI . . . 20

3.3.1 Popis hlavičky . . . 20

3.3.2 Popis stop . . . 21

3.4 Transformace MIDI stopy na maticový formát . . . 22

3.4.1 Přeškálování časových kroků MIDI stopy . . . 22

3.4.2 Převod na maticový formát . . . 22

3.5 Postprocessing . . . 23
3.5.1 Eliminace repetitivních matic . . . 23
3.5.2 Dekompozice a párování . . . 24
3.5.3 Fragmentace partitur . . . 24
3.5.4 Transpozice partitur . . . 25
3.6 Statistiky . . . 26
3.6.1 Rozdělení příznaků . . . 26
3.6.2 Poměr tříd příznaků . . . 27
3.6.3 Rozdělení akordů dle četnosti tónů . . . 28
3.7 Formát datové sady . . . 28

4 Model **29**
4.1 Problém nerovnováhy tříd . . . 29
4.2 Regularizace . . . 30
4.3 Adagrad . . . 30
4.4 Diskriminativní model . . . 31
4.5 Generativní model . . . 31
4.6 Ostatní modely . . . 32
4.7 Výsledná implementace . . . 34

5 Vyhodnocení modelů **36**
5.1 Hyperparametry teploty . . . 36
5.2 Vyhodnocení přesnosti . . . 36
5.3 Vyhodnocení eufonie . . . 38
5.4 Ukázka vygenerovaného doprovodu . . . 38

6 Závěr **40**

Seznam obrázků

2.1 RNN paměťová buňka a její rozvinutí v čase 14
2.2 Autoenkodér, složeného ze dvou dopředných sítí 17

3.1 Ukázka kódování tónů a ligatur 19
3.2 Přeškálování časových kroků MIDI stopy 23
3.3 Vzorkování přeškálované MIDI stopy 23
3.4 Dekompozice matice 25
3.5 Rozdělení příznaků melodie trénovací sady 26
3.6 Rozdělení příznaků harmonie trénovací sady 27
3.7 Rozdělení akordů dle četnosti tónů 28

4.1 Schéma diskriminativního modelu 32
4.2 Schéma generativního modelu 33
4.3 Průběh účelové funkce obou modelů 33

5.1 Srovnání doprovodu s odlišnými hyperparametry teploty 37
5.2 Graf vyhodnocení přesnosti 37
5.3 Graf vyhodnocení eufonie 38
5.4 Harmonie vygenerovaná diskriminativním modelem 39
5.5 Harmonie vygenerovaná generativním modelem 39

1 Úvod

1.1 Melodie a harmonie

Melodie a harmonie jsou spolu s rytmem základním prvkem hudby. Melodie je organizovaná sekvence tónů, odlišujících se výškou a délkou a zpravidla tvoří myšlenku hudebního díla. Výška tónu je úměrná její frekvenci. Například tón o frekvenci 440 Hz vnímá lidský mozek jako tón vyšší, než tón o frekvenci 196 Hz. Dva tóny tvoří oktavu, pokud je poměr jejich frekvencí 1 : 2 (například 440 Hz a 880 Hz). Podstata oktáv tkví v tom, že lidský mozek vyhodnocuje tóny o tomto poměru jako tóny velmi podobné. Důvodem je, že lidský mozek vnímá výšku tónu logaritmicky.

Hudba, běžná v našem prostředí, je založena na vybrané podmnožině frekvencí, též nazývána jako chromatická stupnice. Ta rozděluje oktávu do dvanácti tónů, kde je každý tón identifikován cyklickou posloupností písmen. Ta bývá doplněna číslem, udávajícím oktávu (například C_4, $F_4^{\#}$ nebo A_5). Nejmenší vzdálenost mezi takto vybranými sousedními tóny se nazývá půltón. Vzdálenost dvou půltónů se pak nazývá jako celý tón.

Z chromatické stupnice dále vychází stupnice diatonická. Ta používá pro jednu oktávu tónů sedm o jasně definovaných vzdálenostech mezi tóny (konkrétně pět celých tónů a dva půltóny). Všechny možné kombinace vzdáleností tvoří dohromady sedm modů. Jedním z modů je pak i stupnice durová (jónský modus) a mollová (aiolský modus). Další hojně používanou stupnicí bývá například stupnice pentatonická, vycházející z durové a mollové stupnice. Stupnic existuje však mnohem více a každá disponuje jinými kvalitami.

Harmonie je sekvence více tónů, současně znějících v jeden časový okamžik. Účelem harmonie bývá doplňovat melodii. Souzvuk o více jak dvou tónech, splňující určitá pravidla, se nazývá akord.

1.2 Stručná historie počítačové hudby

Historie počítačové hudby sahá až do roku 1951, kdy Alan Turing vytvořil první počítačem generovanou hudební nahrávku. Ta se skládala ze tří skladeb – hymny Spojeného království „God Save the Queen“, „Baa Baa Black Sheep“ a úryvku

písně „In the Mood“. O šest let později vznikl v Bellových laboratořích první šířeji používaný program, sloužící k počítačovému komponování hudby. Nazýval se MUSIC a jeho autorem byl Maxem Mathewsem.

Revoluce přišla až na počátku 80. let, s rozvojem prvních osobních počítačů, konkrétně s pomocí počítačů Commodore 64 a Atari ST, disponující zvukovým čipem. Ty přinesly možnost počítačově generované hudby mezi širokou populaci. Zhruba v této době též vznikl protokol MIDI, umožňující komunikaci s vnějším hardwarem (například syntetizátory) a přetrval jako standard až dodnes.

Všechna doposud popsaná hudba byla však komponována výhradně člověkem. První hudba ryze počítačově generovaná, vyžadující minimální zásah člověka, byla tvořena Markovovými řetězci. Markovův řetězec je jednoduchý stochastický stavový automat, kde každá přechodová hrana disponuje pravděpodobností pro přechodu do stejného či odlišného stavu. Tento postup například využil Iannis Xenakis k vytvoření díla nazvaného „Analogique“.

Zlom přišel až s postupnou evolucí v oblasti neuronových sítí, převážně se sítěmi rekurentními. První pokus o generování hudby pomocí rekurentních sítí se datuje do roku 1988 [1]. Dalším mezníkem bylo například generování bluesových skladeb, pomocí vylepšených LSTM sítí [2]. V současné době se na generování hudby pomocí neuronových sítí zaměřuje kupříkladu tým Google Magenta nebo IBM (Watson Beat).

1.3 Stanovené cíle

Jelikož se jednalo o velmi komplexní téma, bylo vymezeno několik požadavků, které by měl výsledný model splňovat.

- Vygenerovaný doprovod by měl být výhradně polyfonní. V jednom časovém okamžiku by tedy po většinu času mělo znít zároveň několik tónů.
- Doprovod by měl být vzhledem k melodii pro posluchače eufonický. Snahou tedy bylo, aby vygenerovaná harmonii alespoň odpovídala tónice a stupnici melodie. Omezil jsem se pouze na diatonickou stupnici, respektive její durovou a mollovou podmnožinu, případně na pentatonickou stupnici z ní vycházející.
- Rytmus vygenerovaného doprovodu by měl víceméně korelovat s rytmem melodie.

Výsledný doprovod naopak nemusel dodržovat běžné hudební formy. Tím je myšleno to, že model nemusel v rámci jedné skladby generovat vždy identický doprovod pro opakující se melodické fragmenty. Ty bývají běžné například v rámci slok, refrénů a podobně. Důvodem bylo, že je komplikované naučit neuronové sítě vnímat hudbu jako celek. S obdobným problémem se potýkají modely, generující text přirozeného jazyka. Ty často dokážou věrně napodobit syntaktickou strukturu textu, přesto ale

selhávají v oblasti sémantiky a věty často jako celek postrádají smysl. Přesto ale byla snaha tuto vlastnost do výsledného modelu zahrnout.

1.4 Neuronové sítě

Tvorba modelu, založeného na neuronových sítích, sestává z několika kroků. Ty budou v této sekci popsány pouze velmi zběžně. Prvním a naprosto zásadním krokem, je získání dostatečně velkého množství dat, souvisejících s modelovanou úlohou. Více se touto problematikou zabývám v kapitole 3.

Druhým krokem je vytvoření modelu. Existuje nepřeberné množství architektur neuronových sítí, kdy každá architektura je vhodná pro jiný druh úloh. Jmenovitě například konvoluční sítě, používané výhradně v oblasti strojového vidění (eventuálně v celé oblasti zpracování signálů), nebo rekurentní sítě, o kterých pojednává kapitola 2 a 4.

Posledním krokem je trénování modelu. Tato fáze obecně bývá velmi zdlouhavá. Jednak proto, že pro některé úlohy je trénování náročné na výpočetní výkon. Za druhé proto, že model a optimalizační proces obsahuje mnoho stupňů volnosti (též nazývané jako hyperparametry). Hyperparametrem se rozumí například velikost sítě nebo velikost optimalizačního kroku. Ty obvykle není možné odhadnout z trénovacích dat a je potřeba je k zdárnému natrénování modelu vhodně nastavit, často jen na základě intuice. K trénování modelu je potřeba specifikovat účelovou funkci a použít jeden z mnoho optimalizačních algoritmů:

Účelová funkce $J(\theta)$ (anglicky „objective function“) slouží jako metrika optimalizačního procesu. Ta udává, jak moc se výstup, predikovaný modelem, odlišuje od výstupu referenčního. Cílem je, aby rozdělení modelu a rozdělení trénovacích dat bylo co nejpodobnější. Například funkce křížové binární entropie, jenž je často používána jako metrika binárního klasifikátoru, je definována následovně:

$$J(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i) \tag{1.1}$$

kde y je referenční hodnota a $\hat{y}$ hodnota predikovaná modelem. Tato funkce, respektive její drobná modifikace, bude dále používána v této práci.

Optimalizační proces je proces, jehož cílem je najít $\theta = \arg\min_{\theta} J(\theta)$, neboli parametry modelu θ, minimalizující účelovou funkci J. Nejzákladnějším optimalizačním algoritmem, používaným pro optimalizaci neuronových sítí, je algoritmus Stochastic Gradient Descent (zkráceně SGD) [3], definovaný následovně:

$$\theta_{t+1} = \theta_t - \alpha \frac{\partial}{\partial \theta_t} J(\theta) \tag{1.2}$$

kde α je hyperparametr velikosti kroku. Jedná se o iterativní algoritmus, hledající

minimum účelové funkce $J(\theta)$, na základě její derivace. Není však zaručeno, že nalezené minimum bude globální, nebo že algoritmus bude vůbec k minimu konvergovat. Druhý případ obecně nastává při použití příliš velké hodnoty α.

1.4.1 Diskriminativní a generativní modely

Cílem metody učení s učitelem je natrénování f funkce, která bude přiřazovat příznakům X správnou hodnotu y, tedy $f(X) = y$. Existují dva hlavní přístupy k odhadnutí funkce f:

Diskriminativní přístup se učí podmíněnou pravděpodobnost $P(y|X)$. Obecně bývá náchylný k přeučení. Tento přístup dominuje v oblasti učení s učitelem. Zahrnuje například SVM nebo kNN modely.

Generativní přístup se učí sdruženost pravděpodobnost $P(y, X)$. Jeho cílem je odhadnout rozdělení příznaků X a hodnot y. Tento přístup dominuje v oblasti učení bez učitele. Zahrnuje například Naivní Bayesův klasifikátor nebo GAN modely.

Pokud je k dispozici dostatečné množství trénovacích dat, funguje diskriminativní přístup často lépe v oblasti logistické regrese. Oproti tomu generativní přístup, jelikož explicitně modeluje rozdělení dat, umožňuje generovat umělá data, podobná datům trénovácích.

Těchto odlišných vlastností bylo využito k tvorbě dvou rozdílných modelů. První model byl založený na diskriminativním přístupu a logistické regresi. Druhý model byl generativní a založený na variačním autoenkodéru.

2 Související architektury neuronových sítí

Tato kapitola pojednává o několika architekturách neuronových sítí, které souvisely nebo byly použity v této práci.

2.1 RNN

Rekurentní neuronové sítě (zkráceně RNN) a ostatní sekvenční modely, se používají především v případě, kdy vstupní nebo očekávaná výstupní data jsou časově závislá. Příkladem může být mezijazyčný překlad nebo předpověď či klasifikace časových řad (přepis zvukových nahrávek, detekce anomálií v EKG apod.). Ačkoli by mohlo být možné modelovat tyto úlohy pomocí dopředných neuronových sítí, byla by jejich reprezentace velice neefektivní. Prvním problémem je počet parametrů, který by rostl s každým dalším časovým krokem. Oproti tomu rekurentní neuronové sítě umožňují parametry modelu mezi jednotlivými časovými kroky sdílet. Druhým problémem je, že počet vstupů a výstupů dopředné neuronové sítě je fixní. Rekurentní neuronové sítě na druhou stranu umožňují reprezentovat libovolný počet časových kroků v rámci jednoho modelu.

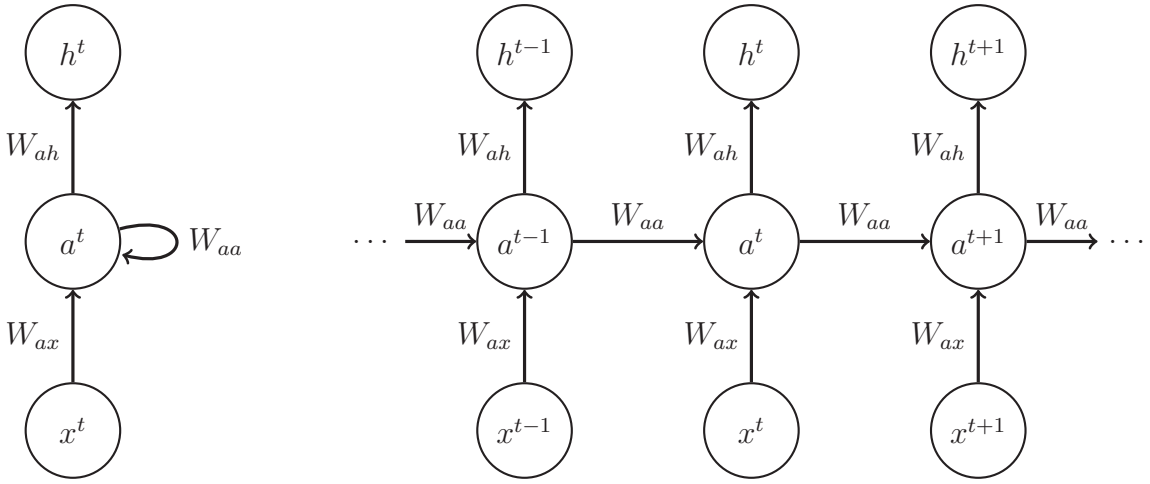
Rekurentní neuronová síť je tvořena několika paměťovými buňkami (ta zhruba odpovídá neuronu v dopředné neuronové síti). Je tvořena (obrázek 2.1) třemi váhami W_{ax}, W_{aa}, W_{ah} (a odpovídajícími biasy b pro každou váhu) a vnitřním stavem a. Vnitřní stav umožňuje buňce uchovávat dlouhodobé informace. Váhy jsou mezi jednotlivými časovými kroky neměnné, kdežto vnitřní stav bývá zpravidla každý časový krok modifikován. Jeho hodnota v čase t je definován jako:

$$a^t = g(W_{aa}a^{t-1} + W_{ax}x^t + b_a) \tag{2.1}$$

kde g je aktivační funkce (nejčastěji tanh nebo ReLU [4]) a výchozí stav bývá běžně $a^0 = \vec{0}$. Výstup paměťové buňky h v čase t je následující:

$$h^t = g(W_{ha}a^t + b_h) \tag{2.2}$$

kde g bývá σ aktivační funkce.



Obrázek 2.1: RNN paměťová buňka a její rozvinutí v čase

2.1.1 Zpětná propagace v čase

Zpětná propagace v čase (anglicky „backpropagation through time“) je algoritmus, sloužící k výpočtu gradientu parametrů vah W a b, vzhledem k účelové funkci $J(\theta)$. Velikost gradientu je úměrná rozdílu (chybě) predikované hodnoty $\hat{y}$ a očekávané hodnoty y. Vypočtené gradienty jsou poté použity během optimalizačního procesu, kde určují, jak moc a jakým směrem (určené znaménkem) má být hodnota váhy upravena. Výpočet je obdobný jako u dopředných sítí, s tím rozdílem, že jednotlivé gradienty vah W (a vah b) se pro všechny časové kroky sčítají:

$$\frac{\partial J}{\partial W} = \sum_{t=1}^{T} \frac{\partial J^t}{\partial W} \tag{2.3}$$

Gradient váhy W_{ha} v časovém kroku t, je s použitím řetízkového pravidla, následující:

$$\frac{\partial J^t}{\partial W_{ha}} = \frac{\partial J^t}{\partial \hat{y}^t} \frac{\partial \hat{y}^t}{\partial z^t} \frac{\partial z^t}{\partial W_{ha}} \tag{2.4}$$

kde $z^t = W_{ha} a^t$. Gradient váhy W_{aa} je v časovém kroku t pak:

$$\frac{\partial J^t}{\partial W_{aa}} = \frac{\partial J^t}{\partial \hat{y}^t} \frac{\partial \hat{y}^t}{\partial a^t} \frac{\partial a^t}{\partial W_{aa}} \tag{2.5}$$

kde však a^t závisí na a^{t-1}, které dále závisí na W_{aa} a tak dále. Vzorec je proto potřeba upravit tak, aby byly explicitně vyjádřeny všechny závislé časové kroky:

$$\frac{\partial J^t}{\partial W_{aa}} = \sum_{k=1}^{t} \frac{\partial J^t}{\partial \hat{y}^t} \frac{\partial \hat{y}^t}{\partial a^t} \frac{\partial a^t}{\partial a^k} \frac{\partial a^k}{\partial W_{aa}} \tag{2.6}$$

Gradient váhy W_{ax} je následující:

$$\frac{\partial J^t}{\partial W_{ax}} = \frac{\partial J^t}{\partial \hat{y}^t} \frac{\partial \hat{y}^t}{\partial a^t} \frac{\partial a^t}{\partial W_{ax}} \tag{2.7}$$

Obdobně se vypočítají gradienty biasů b.

2.1.2 Problém mizejícího a explodujícího gradientu

RNN paměťové buňky se potýkají s problémem mizejícího gradientu (anglicky „vanishing gradient problem“). Ten má za následek neschopnost dlouhodobého uchování informace. Vzniká během zpětné propagace v čase, v případě, kdy je vypočítaný gradient v čase dlouhodobě menší než jedna, následkem čehož může gradient konvergovat k nule. Z tohoto důvodu jsou v praxi RNN paměťové buňky nahrazovány modernějšími LSTM nebo GRU paměťovými buňkami, které jsou navrženy tak, aby problému mizejícího gradientu předcházely.

Opačným chováním je explodující gradient (anglicky „exploding gradient“), kdy je hodnota gradientu dlouhodobně větší než jedna. Běžně se tento problém řeší tak, že pokud gradient překročí určitý práh, je gradient přeškálován. Tato technika se nazývá „gradient clipping“ [5].

2.1.3 Hluboké rekurentní sítě

Paralelou k hlubokým dopředným sítím (neuronové sítě, obsahující více jak dvě vrstvy), jsou hluboké rekurentní sítě. Vnikají též propojením výstupu a vstupů sériovým spojením několika vrstev (v tomto případě několika rekurentních paměťových buněk).

Ty, stejně jako u dopředných sítí, zvyšují vyjadřovací schopnosti sítě. Z důvodu sériového zapojení však nelze paralelně počítat všechny vrstvy v jeden okamžik, jelikož vstup následující vrstvy závisí na výstupu vrstvy předchozí. Přímým důsledkem jejich použití je pak zpomalené trénování sítě.

2.2 LSTM

Long short-term memory (zkráceně LSTM) [6] je variací obyčejné RNN paměťové buňky. Na rozdíl od RNN efektivně řeší problém mizejícího gradientu a disponuje většími vyjadřovacími schopnostmi. Architekturu RNN rozšiřuje o trojici bran:

Aktualizační brána (anglicky „update gate“) na základě předchozího výstupu h^{t-1} a aktuálního vstupu x^t rozhoduje, která data z x^t použít. Je definována násle-

dovně:

$$u^t = g(W_{uh}h^{t-1} + W_{ux}x^t + b_u) \tag{2.8}$$

kde g bývá σ aktivační funkce.

Resetovací brána (anglicky „reset gate") na základě předchozího výstupu h^{t-1} a aktuálního vstupu x^t určuje, která data „zapomenout" z předchozí výstupní hodnoty y^{t-1}. Je definována následovně:

$$r^t = g(W_{rh}h^{t-1} + W_{rx}x^t + b_r) \tag{2.9}$$

kde g bývá σ aktivační funkce.

Výstupní brána (anglicky „output gate") na základě předchozího výstupu h^{t-1} a aktuálního vstupu x^t rozhoduje, jaká data budou výstupem paměťové buňky. Je definována následovně:

$$o^t = g(W_{oh}h^{t-1} + W_{ox}x^t + b_o) \tag{2.10}$$

kde g bývá σ aktivační funkce. Výstupní hodnota h^t v čase t je poté odvozena z výše popsaných bran a vnitřním stavu a^t:

$$\begin{aligned} \tilde{a}^t &= \tanh(W_{ch}h^{t-1} + W_{cx}x^t + b_c) \\ a^t &= u^t \odot \tilde{a}^t + f^t \odot a^{t-1} \\ h^t &= o^t \odot g(a^t) \end{aligned} \tag{2.11}$$

kde $\odot$ je Hadamardův součin a g je σ aktivační funkce nebo funkce identity $f(x) = x$. Výpočet gradientu je pak v principu stejný jako u RNN.

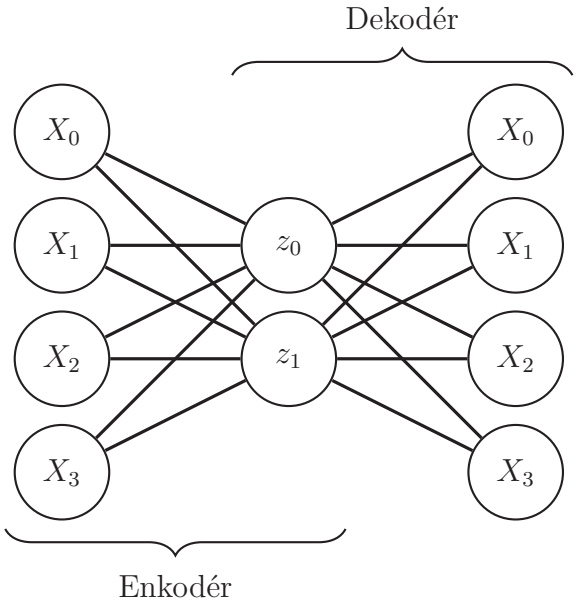
2.2.1 Peephole LSTM

Peephole LSTM [7] je jednou z modifikací obyčejné LSTM paměťové buňky. Jedinou změnou je úprava všech bran tak, aby nezávisely na výstupní hodnotě h^t, nýbrž na vnitřním stavu a^t. Tato změna by měla vést k preciznějšímu časování událostí. Modifikovaná výstupní brána pak bude vypadat následovně:

$$o^t = g(W_{oa}a^{t-1} + W_{ox}x^t + b_o) \tag{2.12}$$

2.3 VAE

Variační autoenkodér (zkráceně VAE) [8] je generativní architektura, rozšiřující autoenkodér o možnost generování nových dat. Takto generovaná data by v ideálním



Obrázek 2.2: Autoenkodér, složeného ze dvou dopředných sítí

případě měla pocházet z podobného rozdělení, jako pocházela data trénovací.

Autoenkodér se skládá z enkodéru a dekodéru (obrázek 2.2). Enkodér transformuje vstupní data X na reprezentaci o nižší dimenzi z. Úlohou dekodéru je pak data z převést do reprezentace původní. Autoekodér se tedy snaží redukovat počet dimenzí, potřebných k reprezentaci dat, přičemž reprezentace nová by měla být podobná reprezentaci původní. Množství ztracené informace se měří pomocí účelové funkce a je snahou ji minimalizovat. V tomto smyslu se chová autoenkodér jako ztrátová komprese.

Nové příznaky vytvořené enkodérem, se nazývají latentní příznaky. Ty často bývají obecnější, než příznaky původní, jelikož k co nejpřesnější reprodukci o nižší dimenzi je potřeba zachytit co nejdůležitější příznaky vstupních dat X. Po natrénování sítě se dekodér zahodí a výstupem jsou pouze latentní příznaky.

Variační autoenkodér, na rozdíl od autoenkodéru, se učí parametry rozdělení (typicky pocházející z normálního rozdělení) trénovacích dat. Ty jsou, obdobně jako u autoenkodéru, reprezentovány latentními příznaky z. Dalším rozdílem je, že u variační autoenkodéru se zahazuje enkodér a nová data vznikají na základě poskytnutých hodnot z. Účelová funkce variačního autoenkodéru je následující:

$$J(\theta)_{\text{VAE}} = -\frac{1}{N}\sum_{i=1}^{N}(J(\theta) - \frac{1}{2}\sum_{j=1}^{J}(1 + \log(\sigma_i^{(j)})^2 - (\mu_i^{(j)})^2 - (\sigma_i^{(j)})^2) \tag{2.13}$$

kde $J(\theta)$ je metrika (například funkce křížové entropie) určující, jak moc se podařilo trénovací data rekonstruovat. Proměnné σ a μ jsou poté parametry latentního prostoru z.

3 Datová sada

Tato kapitola se zabývá tvorbou datové sady. Datová sada se skládala z partitur, tvořenými melodickou a harmonickou složkou. Jsou popsány operace, použité ke zpracování datové sady ze surových dat, a výsledné statistiky vzniklé datové sady.

Velikost a kvalita datové sady bývá při trénování neuronových sítí rozhodující. Exaktní množství není nijak specifikováno a liší se úlohu od úlohy. Například MNIST dataset [9], používaný k trénování klasifikátoru jednociferných číslic, čítá 60 tisíc černobílých obrázků o rozměru 28x28 pixelů. Oproti tomu One Billion Word Benchmark [10], často používaný jako benchmark lingvistických modelů, obsahuje více jak miliardu trénovacích n-gramů anglického jazyka. Datová sada bývá rozdělena do tří množin:

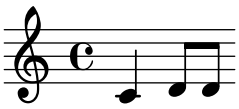
Trénovací množina obsahuje záznamy, použité k trénování modelu. Běžně sestává z více jak 90 % všech dat datové sady. Měla by obsahovat záznamy podobné těm, s kterými bude model při reálném provozu konfrontován.

Validační množina a **testovací množina** by měly pocházet ze stejného rozdělení, jako pocházela množina trénovací. Podstatou validační množiny je vyhodnocení přesnosti modelu během trénování, kdežto množina testovací slouží k vyhodnocení přesnosti finálního modelu, po ukončení trénování. Smyslem tohoto rozčlenění je vyloučení předpojatosti modelu vůči trénovacích datům. Velikost obou množin se pohybuje v řadu jednotek tisíců až desetitisíců záznamů (běžně pouze zlomek celkové velikosti datové sady). Všechny popsané množiny, včetně množiny trénovací, by mezi sebou měly být disjunktní.

Vytvořená datová sada se skládala přibližně z 50 tisíc vzorků trénovacích, přibližně 5 tisíc vzorků validačních a 5 tisíc vzorků testovacích. Důvod jejího vzniku byla absence adekvátního a již hotového řešení.

3.1 Formát záznamů

V této části bude popisován formát, použitý k reprezentaci vstupů a výstupů neuronové sítě a záznamů datové sady. Motivací pro vytvoření popisovaného formátu, byla snaha o vytvoření reprezentace, která by byla přirozeně vhodná ke zpracování neuronovou sítí. Výsledkem byla podmnožina notového zápisu, uložena pomocí matice.



$$\begin{array}{cccc|cc} C_4 & C_4\# & D_4 & C_4 & C_4\# & D_4 \\ \left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{array}\right) \end{array}$$

Obrázek 3.1: Ukázka kódování tónů a ligatur

Esenciální vlastností tohoto formátu byla absence stavu. Každý časový krok explicitně vyjadřoval všechny aktuálně znějící tóny. Byly uchovávány pouze dva druhy příznaků – příznaky aktuálně znějících tónů a příznaky ligatur. Příznaky ligatur určovaly, zda měl být tón mezi dvěma časovými kroky hrán přerušovaně.

Každá buňka matice mohla nabývat hodnot pouze jedné ze dvou tříd, které byly reprezentovány hodnotami 0 nebo 1. Třída nula značila absenci (tónu nebo ligatury), třída 1 naopak její prezenci.

Jeden řádek matice odpovídal jednomu časovému kroku a jeden časový krok se shodoval s délkou jedné čtvrťové noty. Použití takto hrubé škály mělo příznivý dopad na celkový počet parametrů modelu. Na obrázku 3.1 je ukázková reprezentace krátké melodie.

Bylo více příznaků, které by bylo možné do tohoto formátu zahrnout. Ty však byly, kvůli přímočarosti, vynechány.

3.2 Zdrojová data

Zdrojová data sloužila jako surová data, která byla dále upravována do výsledných záznamů. Cílem bylo najít zdroj dat, složený z multi-instrumentálních skladeb, ideálně distribuovaný ve strojově zpracovatelné podobě. Z volně dostupných dat byly zvažovány následující:

The Largest MIDI Collection on the Internet je kompilace MIDI souborů, poskládaná z mnoha různých zdrojů. Obsahovala ∼130000 unikátních souborů, výhradně populární hudby. Duplikáty byly odstraněny porovnáváním MD5 kontrolního součtu jednotlivých skladeb. Tato velikost by měla být už sama o sobě dostatečná, k vytvoření rozmanité datové sady.

The Lakh MIDI Dataset [11], který ve verzi 0.1 obsahoval 176581 MIDI souborů, tvořených výhradně populární hudbou. Z toho 45129 souborů obsahovalo metadata (text, žánr nebo tónika skladeb) prostřednictvím Million Song Dataset

[12]. Metadata zahrnovala například žánr, text nebo tóniku skladby. Stejně jako *The Largest MIDI Collection on the Internet* byl už sám o sobě dostatečně rozsáhlý. Je nabízený pod licencí *CC-BY 4.0*.

Yamaha Disklavier World Dataset je MIDI katalog vážné hudby, jenž se svou rozsáhlostí pohybuje v řádu desetitisíců MIDI souborů. Je tedy výrazně menší, než předchozí popsané soubory dat. Jeho použití by bylo vhodné v případě, kdy by bylo kombinováno s dalším zdrojem dat (pocházejím ze stejného rozdělení).

Nottingham Database, objevený až ke konci práce. Na rozdíl od popsaných předchozích datových sad se neskládá z MIDI souborů, nýbrž ze skladeb zapsaných v ABC notaci, která od druhé verze umožňuje ukládat vícestopé skladby. Skládá se z více jak tisíce folkových skladeb. Díky povaze této notace by mělo být snazší oddělit melodickou a harmonickou složku jednotlivých stop. Nevýhodou je však jeho velikost, která je nedostačující a stejně jako u *Yamaha Disklavier World Dataset*, by bylo vhodné sadu kombinovat se sadou jinou.

Z popsaných souborů dat byl zvolen *The Lakh MIDI Dataset*, jednak pro svou licenční politiku. Druhým důvodem bylo, že již samotný byl dostatečně rozsáhlý k tvorbě pestré datové sady. Důsledkem této volby bylo, že výsledný model byl omezen výhradně na generování doprovodu, disponující aspekty populární hudby.

Posledním důvodem byla striktní a jednoznačná notace, vyplývající z formátu MIDI. V počátcích práce připadalo v úvahu též parsování tabulatur. Od toho však bylo opuštěno, vzhledem k absenci jejich standardizované reprezentace.

3.3 Popis formátu MIDI

Standard MIDI file je binární souborový formát, definovaný standardem *General MIDI*[1]. Soubory, vyhovující tomuto standardu, bývají ukládány pod příponou `.mid`.

MIDI soubory jsou podmnožinou notového zápisu, využívaného k úsporné reprezentaci hudby. Hudba je v tomto případě uložena pouze jako sled událostí (například stisk nebo uvolnění klávesy). Každý soubor se skládá z dvou hlavních úseků – hlavičky a množiny stop.

3.3.1 Popis hlavičky

Hlavička je vždy úplně prvním úsekem souboru. Začíná magickou čtyřbytovou konstantou `MThd`. Ta je následována informací o celkové velikosti úseku, informací o typu souboru, dále informací o celkovém množství stop a nakonec hodnotou `PPQ`. MIDI soubory se dělí do tří typů:

Soubor typu 0 obsahuje všechny nástroje v jedné stopě. Je proto pro tuto práci nepoužitelný.

[1] http://oktopus.hu/uploaded/Tudastar/MIDI%201.0%20Detailed%20Specification.pdf

Soubor typu 1 ukládá každý nástroj do nové stopy a všechny stopy jsou přehrávány souběžně. To umožňuje snazší manipulaci s jednotlivými stopami. Bylo pracováno pouze s tímto typem souboru.

Soubor typu 2 odpovídá prvnímu typu, s tím rozdílem, že stopy jsou přehrávány a ukládány nezávisle na sobě. Bývají proto využívány například k úspornému ukládání opakujících se rytmických částí.

V případě souborů prvního a druhého typu nemusí vždy platit, že jedna stopa odpovídá jednomu nástroji. Stopy mnohdy bývají využívány i k ukládání metainformací o celé skladbě.

Hodnota `PPQ` udává, kolik časových kroků připadá na jednu čtvrťovou notu. Jeden časový krok je nejmenší časovou jednotkou, kterou MIDI soubory disponují. To znamená, že pokud $\texttt{PPQ} = 240$, je čtvrťová nota složena z 240 časových kroků.

3.3.2 Popis stop

Stopa následuje vždy po bloku hlavičky nebo jiné stopě. Začíná magickou čtyřbytovou konstantou `MTrk`, následovanou informací o celkové velikosti úseku a sekvenčním seznamem událostí. Události se dělí na **kanálové události**, související se zvukovým výstupem a řídící **meta události**. Každá událost musí obsahovat informaci o časovém kroku, ve kterém má být zpracována. Události, relevantní pro tuto práci, byly následující:

`End of track` je meta událost, která nemá žádné parametry. Je vždy úplně poslední událostí každé stopy.

`Note on` a **`Note off`** jsou kanálové události, vyžadující dva parametry. Prvním parametrem je jednobajtové číslo tónu, kdy minimální hodnota 0 odpovídá tónu C_{-1} a maximální hodnota 127 odpovídá tónu G_9. Druhý parametr udává, jakou intenzitou má být tón přehrán.

`Note on` událost určuje počátek noty a událost `Note off` její konec. Dohromady obě tyto události tvoří interval, po kterou bude daný tón přehráván. Výjimkou jsou některé MIDI soubory, které nahrazují událost `Note off`, událostí `Note on` s nulovou intenzitou.

`Program change` je meta událost, popisující, jaký nástroj má být použitý k přehrávání budoucích kanálových událostí. Obsahuje jednu jednobajtovou informaci o typu nástroje, kdy například hodnota v rozmezí od 0 až po 7 odpovídá klavírům o odlišných barvách tónu. Tato událost je užitečná k filtrování nástrojů během fáze předzpracování, jelikož některé druhy nástrojů by mohli narušovat homogenitu datového souboru. Jedná se však jen o orientační informaci, která nemusí vždy odpovídat skutečnosti, případně nemusí být uvedena vůbec.

3.4 Transformace MIDI stopy na maticový formát

Tato sekce popisuje způsob, jakým byly převedeny stopy surových MIDI souborů, na maticovou reprezentaci, použitou ve výsledné datové sadě.

3.4.1 Přeškálování časových kroků MIDI stopy

MIDI soubory obecně používají příliš jemnou škálou časových kroků, která není kompatibilní s formátem, popsaným v 3.1. Škála maticového formátu odpovídala `PPQ` = 1. Tato hodnota se však u MIDI souborů běžně pohybuje v rozmezí od `PPQ` = 96 po `PPQ` = 960. Aby bylo možné stopu do maticového formátu převést, bylo potřeba MIDI soubory přeškálovat.

Cílová škála MIDI souboru měla být `PPQ` = 1. K tomu bylo potřeba projít časové kroky všechny události a vydělit je původní hodnotou `PPQ`. Následkem toho mohlo dojít k dvěma nejednoznačným situacím, které byly řešeny následovně:

Situace první nastala, pokud časový krok nebyl dělitelný beze zbytku. Řešením bylo časový krok zaokrouhlit k nejbližšímu celému číslu.

Situace druhá, občas podmíněná první situací, nastala, pokud byly dvě události, dříve se nacházející v odlišných časových krocích, přiřazeny do stejného časového kroku.

Konkrétně se tento problém týkal pouze dvojic událostí `Note on` a `Note off`. Pokud se tyto události nacházely zároveň v jednom časovém kroku, bylo nejednoznačné, zda má tón skončit, nebo pokračovat dále. Tyto události byly nakonec interpretovány jako dva nezávislé tóny.

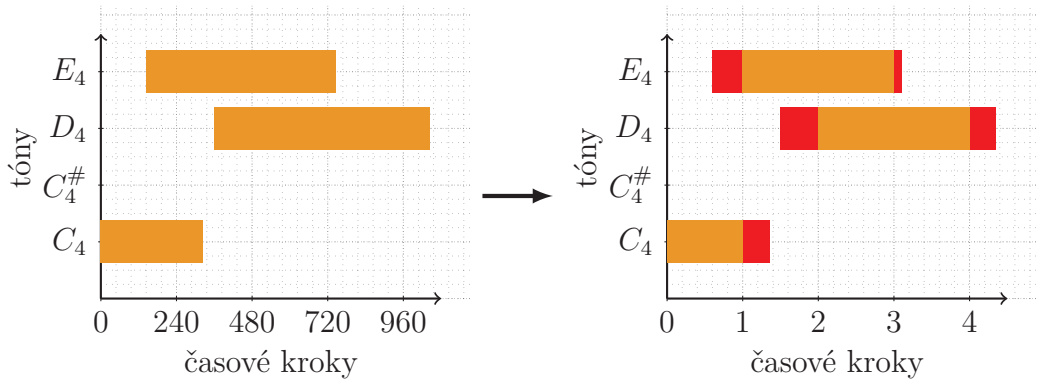
Přeškálování stopy bylo ztrátovou operací a nová stopa se mohla lišit od stopy původní. Tento jev byl způsoben zaokrouhlováním časových kroků. Následkem mohla být ztráta některých informací, případně drobné časové odchylky. To kupříkladu při `PPQ` = 480 a tempu `BPM` = 120 odpovídalo horní hranici nepřesnosti 0,25 sekundy. Při běžném poslechu však byly tyto rozdíly jen obtížně identifikovatelné.

Tato operace je znázorněna na obrázku 3.2, kde původní MIDI stopa odpovídala `PPQ` = 240. Oranžové intervaly znázorňují dvojici událostí `Note on` a `Note off`. Červenou barvou jsou zobrazeny chyby, vzniklé zaokrouhlováním.

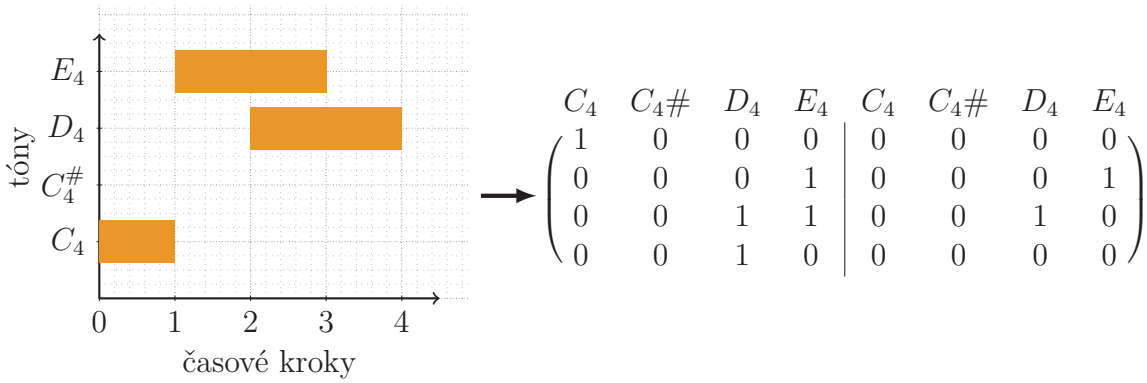
3.4.2 Převod na maticový formát

Přeškálovaná MIDI stopa byla následně převedena na maticový formát. To bylo provedeno procedurou obdobnou vzorkování, s krokem jednoho časového kroku. Cílem bylo získat aktuálně znějící tóny každého časového kroku.

Matice ligatur vznikala tak, že pokud tón hrál v předchozím časovém kroku a zároveň nebyl přerušen, byl mu nastaven příznak ligatury. Výjimkou byl poslední



Obrázek 3.2: Přeškálování časových kroků MIDI stopy



Obrázek 3.3: Vzorkování přeškálované MIDI stopy

časový krok intervalu tónu, pro který nebyl příznak ligatury nastaven. Tato operace byla bezztrátová a je ilustrována na obrázku 3.3.

3.5 Postprocessing

Tato sekce pojednává o formě postprocessigu, který byl aplikován na záznamy převedené do maticového formátu. Výsledkem byly partitury, tvořené melodickou a harmonickou složkou o fixní délce a omezeném tónovém rozsahu. Ty byly následně uloženy do datové sady.

3.5.1 Eliminace repetitivních matic

Populární hudba se skládá výhradně ze stále se opakujících melodických nebo harmonických částí. Aby bylo zamezeno jevu, kdy by výsledný model inklinoval k jedné harmonii z důvodu, že jí byl po dobu trénování více vystaven, byla snaha tyto opakující se fragmenty odstranit.

Jako metrika, pro určení podobnosti dvou záznamů, byla použita kosinová podobnost (3.1). Ta udává podobnost na oboru hodnot $[0, 1]$, kde 1 značí nejvyšší podobnost a naopak. Jako vstup byl použit dvanáctiprvkový vektor frekvencí tónů, bez informací o oktávě. To umožnilo fragment identifikovat nezávisle na tom, zda se počátky obou porovnávaných záznamů shodovaly.

Záznam A byl zahozen, pokud jeho vektor frekvencí tónů $\vec{a}$ splňoval $s(\vec{a}, \vec{b}) > \epsilon$. Vektor $\vec{b}$ je vektor již zahrnutého záznamu. Hodnota hranice byla empiricky stanovena na $\epsilon = 0.95$.

$$s(\vec{a}, \vec{b}) = \cos(\theta) = \frac{\vec{a} \cdot \vec{b}}{\|a\|\|b\|} \tag{3.1}$$

3.5.2 Dekompozice a párování

Cílem této operace bylo oddělit melodickou a harmonickou složku z maticového formátu. K identifikaci melodické, respektive harmonické, složky nástroje byla použita vektorová $l1$ norma (3.2), aplikovaná na $\vec{a}_t$, představující časový krok t matice tónů. Pokud $\|\vec{a_t}\| \leq 1$, byl časový krok vektoru $\vec{a}_t$ vyhodnocen jako melodický. Pokud $\|\vec{a_t}\| > 1$, tak jako harmonický.

$$\|x\| = \sum_{i=1}^{n} |x_i| \tag{3.2}$$

Z takto identifikovaných časových kroků byla poté vytvořena výhradně melodická a výhradně harmonická matice. Časový krok, které příslušely do druhé matice, byly nahrazeny nulovým vektorem. Tato operace je znázorněna na obrázku 3.4.

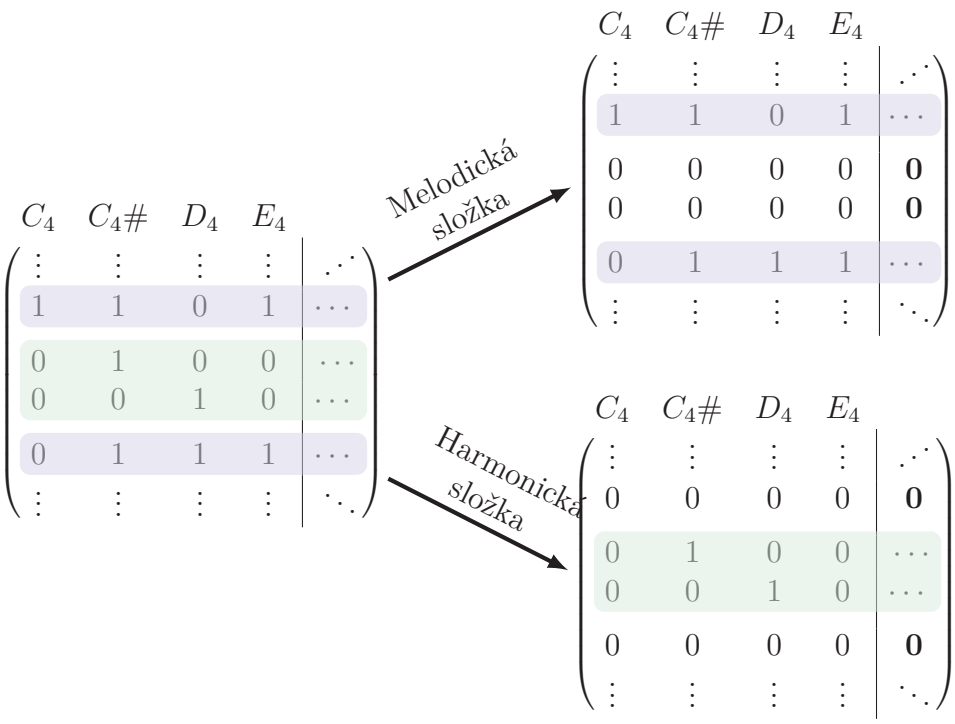
Melodické a harmonické složky stop jedné skladby byly následně vzájemně párovány do partitur. Pro n stop v jedné skladbě tedy vzniklo V_2^n různých partitur.

3.5.3 Fragmentace partitur

Cílem fragmentace bylo odstranění prázdných úseků, které se přirozeně nacházely v melodických a harmonický složkách partitury. Prázdným časovým úsekem je myšlen úsek, kdy zároveň pro obě ze složek nezněl žádný tón.

Nalezené prázdné sekvence časových kroků delších, než byla stanovená hranice k, poté sloužily jako oddělovače partitur nových, derivovaných z partitury výchozí. Výsledkem fragmentace tedy mohlo být 0 až n partitur nových, každá obsahující prázdné sekvence o nejvýše $k - 1$ časových krocích. Výsledná datová sada byla vytvořena s použitím $k = 8$, odpovídající pomlce o délce dvoucelé noty.

Z nově derivovaných dvojic poté byly odstraněny dvojice, jenž byly kratší než T časových kroků. Dvojice delší, než T časových kroků byly rozděleny do $n = \text{floor}(\frac{l}{T})$



Obrázek 3.4: Dekompozice matice

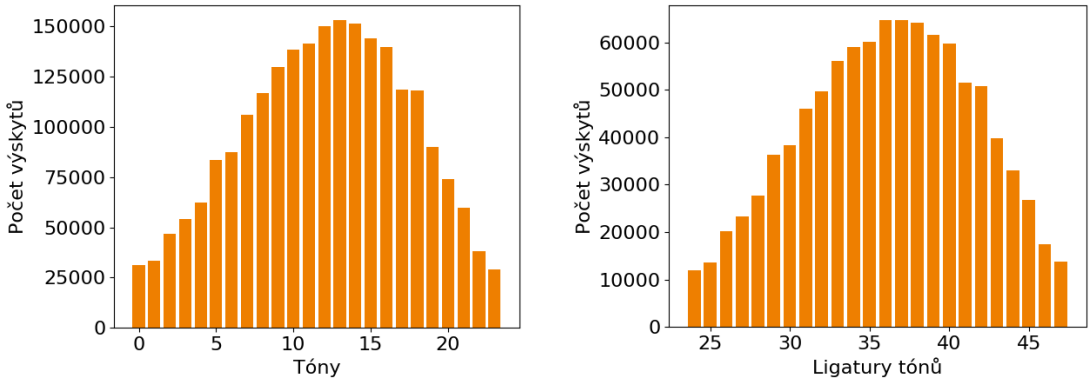
dvojic nových (l je počet časových kroků jedné dvojice). Pokud nebylo možné dvojice rozdělit beze zbytku, byly nadbytečné časové kroky zahozeny. Datová sada byla vytvořena s použitím $T = 64$.

3.5.4 Transpozice partitur

Transpozicí se rozumí převedení celé skladby (nebo její části) do jiné tóniny, přičemž jsou zachovány intervaly mezi jednotlivými tóny. Prvním důvodem pro transponování záznamů byla snaha o rovnoměrnější pokrytí tónů datové sady. Výsledný model by v opačném případě mohl selhávat na tónech, které v ní nebyly dostatečně zastoupeny. Druhým důvodem bylo umělé rozšíření datové sady o nové záznamy.

Tato technika se v rámci strojového učení nazývá augmentace dat. Její podstatou je odvození nových záznamů, ze záznamů již existujících. Záměrem je přidání nových informací do datové sady a tím zvýšení její variability. Důvodem pro použití může být například případ, kdy není možné získat dostatečné množství dat, potřebných k úspěšnému natrénování modelu. Druhým případem může být (výše popsaná) snaha o zvýšení robustnosti výsledného modelu. Například v oblasti počítačového vidění se provádí jednoduché transformace (pootočení, zrcadlení, úprava barevného nádechu apod.) trénovacích snímků.

V této fázi byl též omezen rozsah tónů na dvě oktávy. Bylo využito vlastnosti



(a) Rozdělení tónů

(b) Rozdělení ligatur tónů

Obrázek 3.5: Rozdělení příznaků melodie trénovací sady

transpozice kdy pokud jsou transponovány všechny složky o stejný rozdíl, disponuje výsledek stejnými zvukovými kvalitami. Přes zvolený interval poté byla partitura postupně transponována s krokem 4 a náhodným počátkem v rozmezí $[0, 3]$. Vzniklo tak 0 až n partitur nových, derivovaných z partitury původní. Pokud byl tónový rozsah partitury větší, než dvě oktávy, byla partitura zahozena.

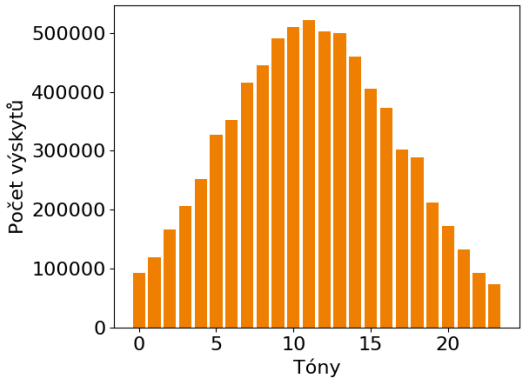
Bývá běžné data nahodile augmentovat před každou epochu trénování. Tento přístup avšak nebyl použit, jelikož trénování modelu bylo podmíněny statistikami jednotlivých příznaků datové sady.

3.6 Statistiky

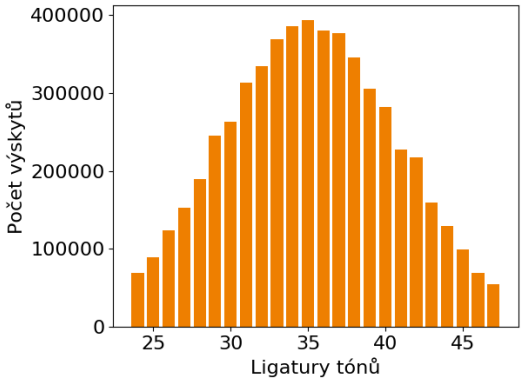
Statistiky jsou omezeny pouze na trénovací sadu. Jednak proto, že by statistiky měly být obdobné (trénovací, validační a testovací množina pocházela z podobných zdrojových dat). Dále proto, že statistiky validační a testovací sady nebyly nezbytné pro další trénování modelu.

3.6.1 Rozdělení příznaků

Rozdělení příznaků udává absolutní počet třídy 1, reprezentující prezenci tónu nebo ligatury, pro každý příznak datové sady. Na obrázku 3.5 je zobrazeno rozdělení tříd pro melodické složky. Obsahovalo o ∼2,32 % více tónů, než ligatur. Obrázek 3.6 zobrazuje rozdělení tříd pro harmonické složky. Obsahovalo o ∼1,33 % více tónů, než ligatur. Z toho vyplývá, že noty harmonie by měly být obecně delší (měli menší poměr tónů a ligatur). Z obrázků je patrné, že rozdělení tónů a ligatur se řídilo normálním rozdělením. To mohlo být způsobeno nebo zkresleno transpozicí.



(a) Rozdělení tónů



(b) Rozdělení ligatur tónů

Obrázek 3.6: Rozdělení příznaků harmonie trénovací sady

3.6.2 Poměr tříd příznaků

Každý příznak mohl nabývat jedné ze dvou tříd. Výsledné hodnoty reprezentují poměr mezi třídou absence a třídou prezence. Výsledky > 1 značily, že v příznaku dominovala třída absence a naopak.

V tabulce 3.1 a 3.2 jsou absolutní výsledky pro každou třídu. Třída $0 \leq k < 24$ jsou příznaky tónů a $24 \leq k < 48$ příznaky ligatur. Z dat vyplývá, že třídy jsou výrazně nevyvážené, přičemž pro každý příznak výrazně převažuje třída, reprezentující absenci.

Výsledky byly dále použity jako konstanty pro penalizaci příznaků v účelové funkci během trénování (viz kapitola 4).

k	0	1	2	3	4	5	6	7	8	9	10	11
	42,3	32,2	22,3	17,8	14,4	10,7	9,9	8,2	7,6	6,8	6,6	6,4
k	12	13	14	15	16	17	18	19	20	21	22	23
	6,6	6,8	7,6	8,8	9,8	12,5	13,3	18,3	23,3	30,9	45,8	56,7

Tabulka 3.1: Poměr tříd příznaků tónů v trénovací sadě

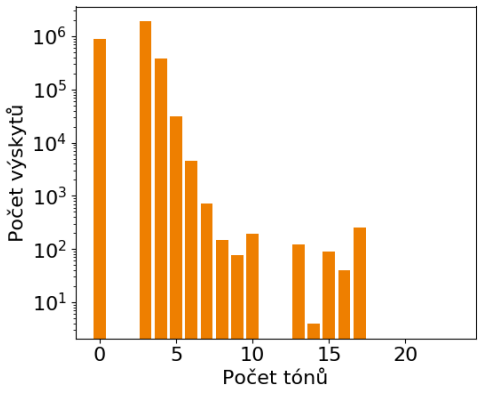
k	24	25	26	27	28	29	30	31	32	33	34	35
	62,6	47,9	33,2	26,6	21,5	16,1	14,9	12,3	11,5	10,3	9,9	9,7
k	36	37	38	39	40	41	42	43	44	45	46	47
	10,0	10,4	11,4	13,2	14,6	18,5	19,8	27,5	35,6	47,2	71,8	90,4

Tabulka 3.2: Poměr tříd příznaků ligatur v trénovací sadě

3.6.3 Rozdělení akordů dle četnosti tónů

Na obrázku 3.7 (logaritmická škála) jsou zobrazeny četnosti tónů, jenž obsahovaly akordy trénovací sady. V trénovací sadě byly nejvíce zastoupeny akordy složené ze tří tónů. S podobnou četností byly zastoupeny i pomlky (akord o nula tónech).

Trénovací sada dále obsahovala zanedbatelné množství akordů o sedmi a více tónech. To mohlo být kupříkladu způsobeno přeškálováním, popsaným v 3.4.1, případně poškozenými zdrojovými MIDI soubory.



Obrázek 3.7: Rozdělení akordů dle četnosti tónů

3.7 Formát datové sady

K uložení datové byl použit binární datový formát `TFRecords`, založený na knihovně `protobuf`. Důvodem pro zvolení tohoto formátu byla nativní podpora ve frameworku `TensorFlow`.

Ve výsledku vzniklo osm souborů pro trénovací sadu a po jednom pro sadu validační a testovací, kdy jeden soubor obsahoval ∼6000 záznamů. Průměrná velikost souboru byla 10 MB. Data byla komprimována algoritmem Gzip.

4 Model

Pro účely generování hudebního doprovodu byly vytvořena dva modely, založené na rozdílných přístupech. Pro implementaci byl zvolen framework TensorFlow [13], disponující automatickým výpočtem gradientu a umožňující provádět výpočty na grafické kartě, pomocí technologie CUDA.

4.1 Problém nerovnováhy tříd

Problém nerovnováhy tříd (anglicky „class imbalance problem“) je jev, kdy četnost jedné třídy dat v trénovací datové sadě výrazně převyšuje třídu jinou. Neuronové sítě (a některé další algoritmy strojového učení) jsou implicitně navrženy tak, že dosahují optimálních výsledků, pokud jsou třídy zastoupeny víceméně rovnoměrně.

Především diskriminativní model, založený na logistické regresi, měl tendence konstantně predikovat prázdné sekvence. Příčinou tohoto chování byla účelová funkce binární křížové entropie, jejíž minimem, vzhledem k nerovnováze tříd, byl model, predikující samé nuly. Jako dočasné řešení posloužilo použití velmi malého optimalizační kroku a zastavení trénování modelu ještě předtím, než bylo tohoto minima dosaženo. Získané výsledky byly avšak podprůměrné o od tohoto postupu bylo opuštěno.

Jednou z technik řešení tohoto problému může být vygenerování umělých dat minoritní třídy, případně zanedbání některý dat třídy majoritní tak, aby se dosáhlo rovnováhy. Nicméně ani jeden z popsaných přístupů nebylo možné v tomto případě použít. Z reprezentace hudebního záznamu, popsané v sekci 3.1, nebylo možné opomenout některý z příznaků. Obdobně generování matic, které by obsahovaly rovnoměrně rozdělené třídy pro všechny příznaky, nedávalo smysl.

Použité řešení zahrnovalo modifikaci účelové funkce tak, aby více penalizovala falešně negativní předpovědi. Jako penalizační konstanty byly použity hodnoty ze statistiky popsané v sekci 3.6.2, kdy byl každý příznak penalizován odlišnou konstantou, vzhledem k absolutnímu zastoupení třídy daného příznaku. Modifikovaná účelová funkce J, vycházející z binární křížové entropie, vypadala následovně:

$$J(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N} C y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i) \tag{4.1}$$

kde C byl vektor penalizačních konstant pro každý příznak.

4.2 Regularizace

Předmětem regularizace je zamezení přeučení. Přeučení je jev, kdy se model až příliš přizpůsobil datům z trénovací sady, čímž se redukovaly jeho generalizační schopnosti. Přeučení může indikovat například veliký rozdíl přesnosti na testovací a validační sadě, oproti sadě trénovací. Může též vzniknout použitím velkého množství parametrů modelu. Přesto se však nedoporučuje počet parametrů ihned snižovat (a tím omezovat vyjadřovací schopnost modelu), nýbrž použít nějakou jinou formu regularizace.

V kontextu této úlohy problém přeučení vznikal v moment, kdy model přestal generovat výhradně originální sekvence, ale místo toho začal kopírovat sekvence ze sady trénovací. Problém přeučení avšak nebyl z hlediska této úlohy tolik problematický, jako by mohl být v úlohách jiných. Důvodem je, že opakující se motivy jsou napříč hudebními díly populární hudby běžné [1].

První použitou regularizační technikou, byla technika Dropout, respektive její upravená varianta pro rekurentní sítě [14]. Myšlenkou této techniky je opomenutí některých vstupních, výstupních nebo skrytých neuronů (případně buněk, v případě rekurentních sítí) během dopředného a zpětného průchodu. Ty bývají při každé iterace vybírány náhodně, na základě předem stanoveného výsledného procentuálního zastoupení.

Druhou použitou regularizační technikou bylo předčasné zastavení trénování (anglicky „early stopping“). Jako výsledný model byl vybrán ten, jenž dosahoval nejvyšší přesnosti na validační sadě. Během trénovací fáze tedy byla ukládána historie několika posledních modelů, přičemž model byl ukládán po každé epoše trénování.

4.3 Adagrad

K trénování obou modelů byl použit algoritmus Adagrad [15], který je definovaný následovně:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{G_t + \epsilon}} \odot \frac{\partial}{\partial \theta_t} J(\theta) \tag{4.2}$$

kde $\odot$ je Hadamardův součin, α je počáteční velikost kroku a ϵ je velmi malý člen, sloužící k eliminace dělení nulou. Adagrad rozšiřuje algoritmus SGD o adaptivní velikost kroku α, závisejícího na parametru θ. Hyperparametr α je tudíž pro každou dimenzi odlišný. Dále algoritmus automaticky volí velkou hodnotu α pro méně časté

[1] https://www.nature.com/articles/srep00521.pdf

příznaky a malou hodnotu pro více časté příznaky. Díky této vlastnosti se jeho použití osvědčilo na nevyvážených datech.

4.4 Diskriminativní model

Diskriminativní model generoval harmonii časového kroku t na základě aktuálního tónu melodie a skrytého stavu, odvozeného ze všech předchozích časových kroků melodie, neboli:

$$p(y) = \prod_{t=1}^{T} p(y_t|X_t, \ldots, X_1) \tag{4.3}$$

kde X je poskytnutá melodická sekvence, T celkový počet časových kroků melodie a y vygenerovaná harmonie.

Model byl založen na hluboké rekurentní neuronové sítí o dvou vrstvách, kdy se každá vrstva skládala z 128 Peephole LSTM paměťových buněk. Výstup paměťových buněk v každém kroku byl dále transformován dopřednou neuronovou sítí na rozměr výsledné harmonické matice. Takto transformovaný výstup byl poté přeškálován funkcí σ na interval hodnot v rozsahu $[0, 1]$. Jako účelová funkce byla použita modifikovaná verze binární křížové entropie, popsaná v 4.1.

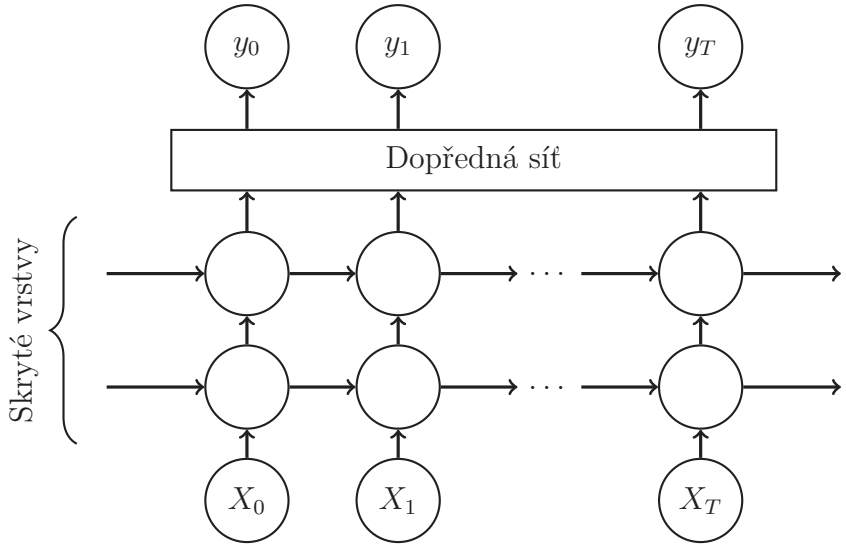
Během trénování byla na výstupní hodnoty LSTM paměťových buněk aplikována regularizace pomocí techniky Dropout, která měla zabránit přeučení. Pravděpodobnost zachování výstupu obou vrstev byla empiricky stanovena na 70 %.

Model byl trénován po dobu 100 epoch. K optimalizaci modelu byl použit algoritmus Adagrad, s krokem $0,025$. V jedné trénovací iteraci byl prováděn paralelní výpočet s 128 trénovacími vzorky, které byly před každou epochou náhodně promíchány. Graf průběh učení je na obrázku 4.3a. Schéma modelu je na obrázku 4.1.

4.5 Generativní model

Generativní model byl založený na variačním autoenkodéru, tvořený dvěma rekurentními sítěmi. Architektura modelu byla výrazně inspirována článkem „Generating Sentences from a Continuous Space„ [16], kde byla tato architektura použita ke generování krátkých vět. První síť, enkodér, byla tvořena dvěma vrstvami o 128 Peephole LSTM paměťových buňkách. Druhá síť, dekodér, disponovala identickými parametry.

Generativní model, navíc oproti modelu diskriminativnímu, generoval harmonii na základě globální latentní reprezentace melodie. To probíhalo tak, že vstupní melodie byla enkodérem převedena na latentní reprezentaci z, tvořenou 11 hodnotami, pocházejích z normálního rozdělení. Tato latentní reprezentace byla následně použita



Obrázek 4.1: Schéma diskriminativního modelu

jako počáteční stav obou rekurentních vrstev dekodéru.

Po natrénování byl enkodér zahozen. Generování harmonie probíhalo tak, že počáteční stav dekodéru byl nastaven na náhodnou hodnotu $z \sim \mathcal{N}(0, 1)$ a síti byla jako vstup poskytnuta celá melodie. Přítomnost latentní reprezentace z umožňovala pro jednu melodii generovat libovolný počet harmonií.

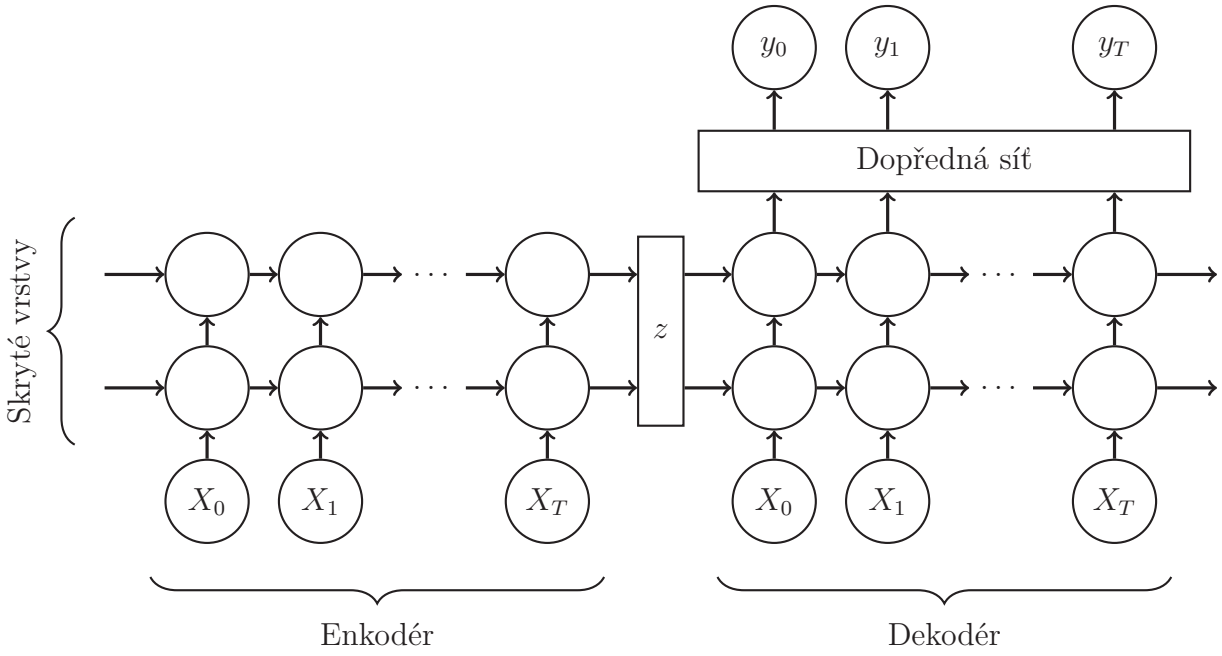
Optimalizace probíhala téměř identicky, jako u diskriminativního modelu. Rozdílná byla pouze velikost kroku, kdy byla použita nižší hodnota $0,01$. Jako účelová funkce byla použita funkce $J(\theta)_{\text{VAE}}$ variačního autoenkodéru, popsaná v 2.3. Jako účelová funkce, indikující, jak moc se podařilo trénovací data rekonstruovat, pak funkce popsaná v 4.1.

Graf průběh učení je na obrázku 4.3b. Nebyla použita žádná forma regularizace, vyjma předčasného zastavení. Schéma modelu je na obrázku 4.2. Model byl trénován 100 epoch. Jako výsledný model byl však použit model ze 40 epochy, vzhledem ke klesající přesnosti následujících epoch na validační sadě.

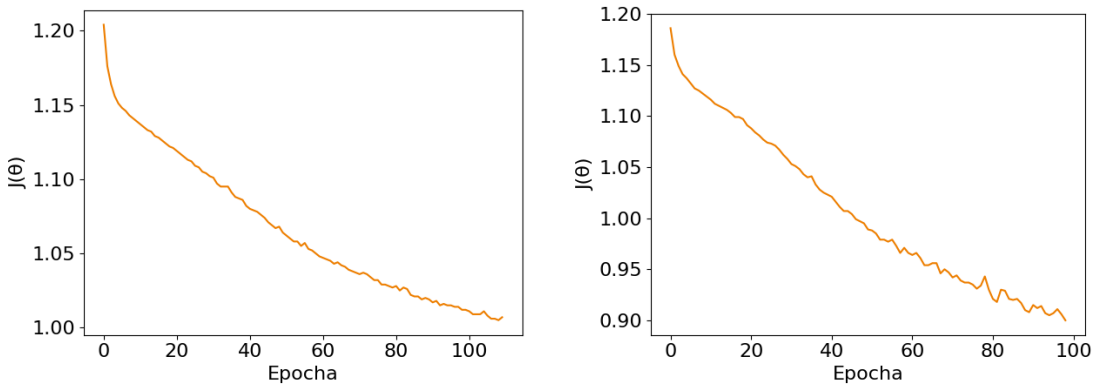
4.6 Ostatní modely

Dále byl vytvořený model, založený na generativní kompetitivní síť (zkráceně GAN) [17]. Ten se však nepodařilo zprovoznit.

Trénování generativních kompetitivních sítí, založených na sítích rekurentních, je obecně velmi komplikované [18]. V odkazovaném článku byly k trénování modelu například použity metody, založené na graduálním zvyšování trénovací sekvence (curriculum learning [19]). Trénování modelu, generující text přirozeného jazyka o délce 64 znaků, bylo navíc údajně značně zdlouhavé (přibližně 12 dní na nespecifikovaném hardwaru).



Obrázek 4.2: Schéma generativního modelu



(a) Diskriminativní model

(b) Generativní model

Obrázek 4.3: Průběh účelové funkce obou modelů

Generativní kompetitivní sít jsou, oproti variačnímu autoenkodéru, omezeny pouze na spojité vstupní hodnoty [2]. Techniky, založené na předtrénovaném modelu, zkoušeny nebyly.

4.7 Výsledná implementace

Výsledkem práce byl také jednoduchý program, ovládaný z příkazové řádky. Ten jako vstup přijímá soubor obsahující sekvenci not, zapsaných v notaci vytvořené výhradně pro tento účel. Výstupem programu je MIDI soubor s vygenerovaným doprovodem. Délka sekvence není nijak omezena.

Použitá gramatika je přímočará a skládá se pouze z názvů not, oddělených mezerou, a několika modifikátorů. Validní názvy not jsou C, D, E, F, G, A, B, kdy jedna nota odpovídá notě čtvrťové. Notu lze snížit o půltón přidáním znaku b (ASCII kód 98) za notu, nebo naopak notu o půltón zvýšit, přidáním znaku # (ASCII kód 35). Notu lze o oktávu zvýšit znakem $'$ (ASCII kód 39), bezprostředně následovaným za notou. Z důvodu omezení modelu pouze na dvě oktávy však nelze tento modifikátor řetězit. Posledním modifikátorem je znak $\sim$ (ASCII kód 126), přidaný bezprostředně před notou. Ten zajistí, že nota bude hrána bez přerušení v dalším časovém kroku. Ukázková notace je vyobrazena ve výpisu 4.1.

```
Ab B E' ~F# ~F# ~F# ~F# E B Db ~Db' ~Db' ~Db' ~Db'
```

Výpis 4.1: Příklad melodie zapsané v popsané notaci

Program disponuje několika volitelnými parametry a přepínači, které modifikují výsledný MIDI soubor:

`-o`, **`--output`** nastavuje název výsledného MIDI souboru. Pokud není uvedeno jinak, bude název vygenerovaného soubor `output.mid`.

`--seed`, kdy v případě použití tohoto parametru bude harmonie tvořena pomocí generativního modelu, namísto výchozího diskriminativního. Tato hodnota určuje počáteční stav pseudonáhodného generátoru čísel, generující latentní vektor z. Tím je zajištěn determinismus generování. Pro stejnou hodnotu bude vždy vygenerován stejný doprovod.

`--tempo` určuje rychlost výsledné skladby. Výchozí je tempo 120, které odpovídá 500 ms na jednu čtvrťovou notu.

`--temperature-notes` umožňuje specifikovat hyperparametr teploty, popsaný v 5.1. Čím je tato hodnota nižší, z tím více tónů se budou vygenerované akordy skládat. Výchozí hodnota je 1,3.

`--temperature-ligatures`, stejně jako předchozí parametr, umožňuje kontro-

[2] https://www.reddit.com/r/MachineLearning/comments/40ldq6/generative_adversarial_networks_for_text/cyyp0nl/

lovat hyperparametr teploty ligatur. Čím je tato hodnota nižší, tím budou vygenerované akordy delší a naopak. Výchozí hodnota je 1.

`--ignore-melody` a **`--ignore-harmony`** jsou přepínače, poskytující kontrolu nad tím, zda má být do výsledného MIDI souboru zahrnuta poskytnutá melodie a vygenerovaná harmonie. Ve výchozím stavu disponuje výstupní soubor oběma stopami.

`--octave-melody` a **`--octave-harmony`** jsou parametry, dovolující nastavit základní oktávu pro jednotlivé stopy. Například při `--octave-melody=5`, odpovídá C notě C^5 a C' notě C^6. Výchozími hodnotami jsou `--octave-melody=5` a `--octave-harmony=3`.

5 Vyhodnocení modelů

5.1 Hyperparametry teploty

Hyperparametr teploty τ (anglicky „temperature“) [20] umožňoval kontrolovat sebejistotu modelu. Určoval míru, s jakou byl výstupní signál přeškálovan. Pokud $\tau < 1$, byl výstupní signál zesílen, což měla za následek vyšší množství falešně pozitivních předpovědí. Pokud $\tau > 1$, signál byl zeslaben. Při $\tau = 1$ procházel signál nezměněn. Na rozdíl od odkazovaného článku bylo prováděno přeškálování hodnot až po poslední aktivační funkci, následujícím výrazem:

$$\tilde{y} = \min(1, \tau^{-1}\hat{y}) \tag{5.1}$$

kde funkce min zajišťovala, že $\tilde{y}$ zůstane v rozmezí hodnot $[0, 1]$.

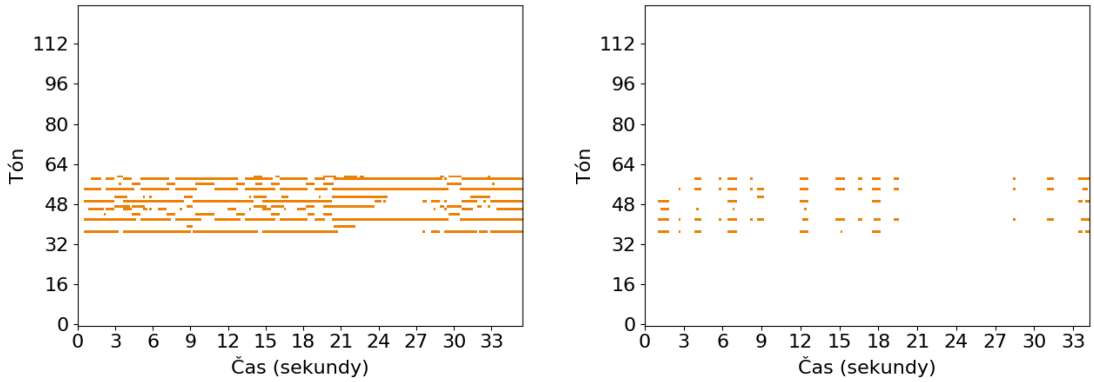
Bez použití tohoto hyperparametru (neboli $\tau = 1$), měl model tendence tvořit dlouhé akordy o mnoha tónech. To se velmi negativně projevovalo na výsledném doprovodu, který tak nebyl v souladu s melodií. Příčinou tohoto chování mohla být penalizace tříd jednotlivých příznaků.

Oba výsledné modely disponovaly dvěma hyperparametry teploty. První hyperparametr τ_α určoval míru teploty tónů. Při vyšší hodnotě model obecně tvořil akordy o méně tónech a naopak. Druhý hyperparametr τ_β reguloval ligatury. Čím byla jeho hodnota nižší, tím byly obecně akordy kratší. Na obrázku 5.1 je porovnán vygenerovaného doprovod na základě různých hodnot hyperparametru teploty.

Optimálních výsledků bylo v dosaženo s použitím hodnot, pohybujících se kolem $\tau_\alpha = 1,3$ a $\tau_\beta = 1$. Tento parametr je ve výsledném programu nastavitelný, přičemž je ve většině případech doporučeno používat $\tau_\alpha > 1$ a $\tau_\beta > 0,8$.

5.2 Vyhodnocení přesnosti

Přesnost modelu byla počítána mezi vygenerovaným doprovodem a referenčním doprovodem z validační a testovací sady. Její hodnota byla stanovena jako poměr mezi



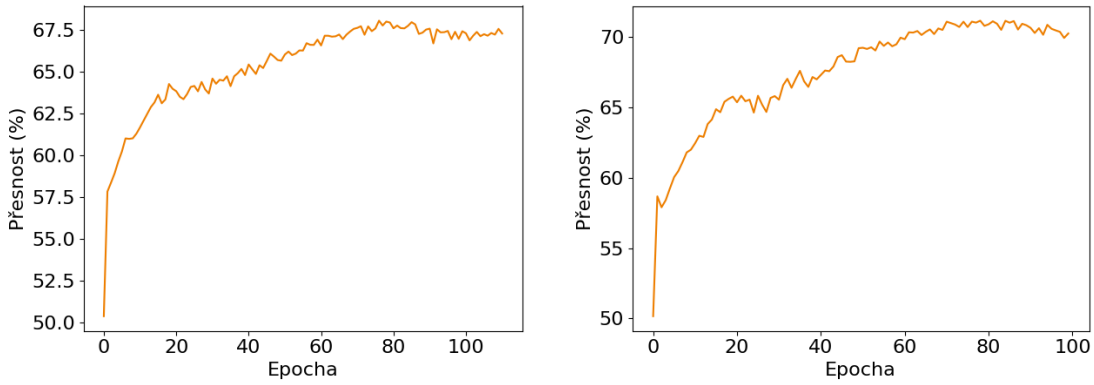
(a) Příliš nízká hodnota (b) Příliš vysoká hodnota

Obrázek 5.1: Srovnání doprovodu s odlišnými hyperparametry teploty

správně klasifikovanými třídami a celkovým počtem dat, neboli:

$$\text{Přesnost} = \frac{\text{Počet správně klasifikovaných}}{\text{Celkový počet}} \tag{5.2}$$

Výsledky modelu pro validační sadu byly počítány během trénování a výsledné hodnoty pro všechny epochy trénování jsou na obrázku 5.2. Přesnost na testovací sadě byla vypočítána až po ukončení trénování. Pro diskriminativní model činila 0,705 % a 0,722 % pro model generativní.



(a) Diskriminativní model (b) Generativní model

Obrázek 5.2: Graf vyhodnocení přesnosti

Tato testovací metrika, vyhodnocující přesnost modelu na základě porovnání predikované hodnoty s referenční hodnotou, nebyla vzhledem ke své povaze zcela relevantní. Důvodem je, že k jedné vygenerované harmonii může náležet více validních melodií a naopak. Je ji potřeba brát s rezervou a měla by mít spíše jen informační charakter.

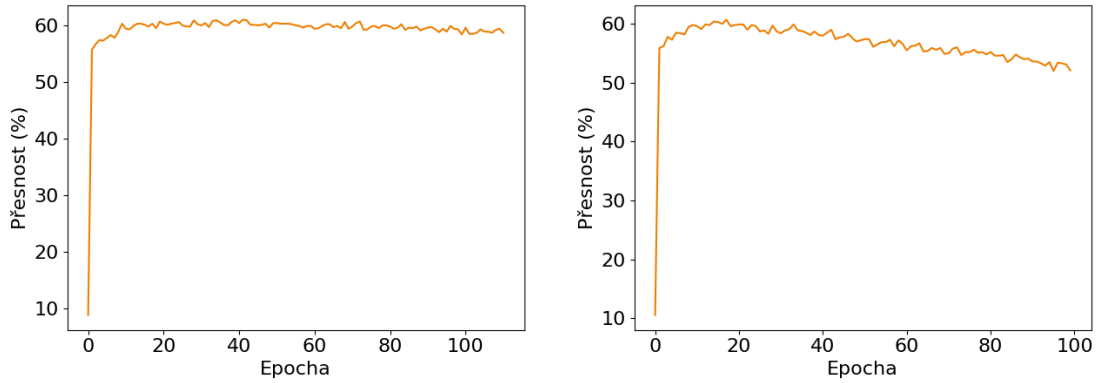
5.3 Vyhodnocení eufonie

Tato metrika se snažila, na rozdíl od metriky předchozí, měřit vlastnosti výsledného doprovodu. Vycházela z předpokladu, že pokud melodie a harmonie spolu dobře zní, je pravděpodobné, že pochází ze stejné tóniny.

K rozeznání stupnic byl použit Krumhansl-Schmuckler algoritmus [21]. Ten odhaduje stupnici na základě porovnávání profilu použitých tónů poskytnutého hudebního vzorku s profily tónin referenčními. Jako výsledná tónina je poté vybrána ta, jenž koreluje nejvíce. Algoritmus je omezený pouze na durovou a mollovou stupnici (což v rozsahu této práce nebyl problém).

Nutno podotknout, že tento test nebral v potaz změnu tóniny v rámci jednoho hudebního vzorku. Jelikož však tento jev není v populární hudbě příliš častý a jelikož validační a testovací sekvence byly krátké, neměly by být výsledky nijak výrazně zkreslené.

Výsledné grafy pro oba natrénované modely, vyhodnocené na validační sadě, jsou na obrázku 5.3. Přesnost na testovací sadě byla vypočítána až po ukončení trénování a její výsledná hodnota byla 0,592 % pro diskriminativní model a 0,59 % pro generativní model.



(a) Diskriminativní model (b) Generativní model

Obrázek 5.3: Graf vyhodnocení eufonie

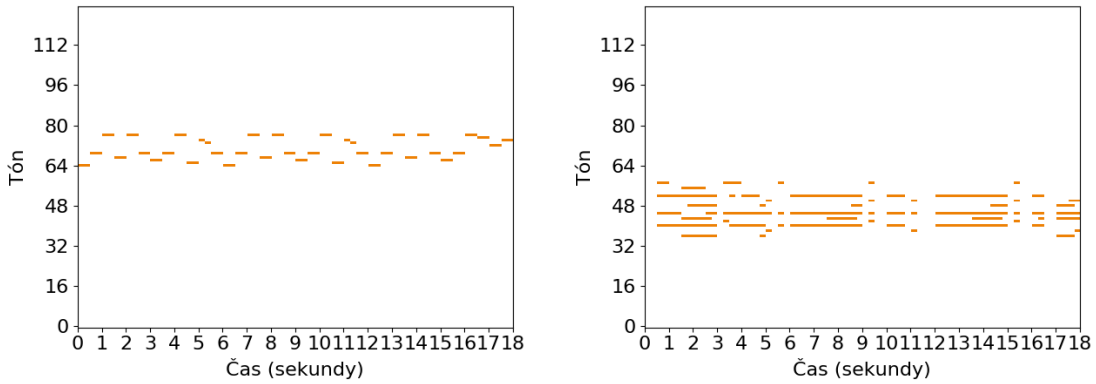
5.4 Ukázka vygenerovaného doprovodu

Na obrázcích 5.4 a 5.5 jsou ukázky vygenerovaných doprovodů obou modelů. Jako vzorová melodie byla použita úvodní melodie skladby „Lucy in the Sky with Diamonds“ od skupiny The Beatles. Z obrázků je patrné, že se síť naučila tvořit souzvuk více tónů o víceméně správném načasování. Špatné načasování akordů se projevovalo tak, že akord nebyl celý zahrán v jeden časový okamžik, nýbrž některé z jeho tónů byly zahrány o (typicky jeden) časový krok později. Tento úkaz byl v některých

případech rušivý a nemusel by být tolik patrný s použitím jemnější škály časového kroku.

Druhým problémem byly redundantní tóny. Jednalo se o tóny, které zněly souběžně s akordem, ale nebyly jeho součástí. Jejich přítomnost lze sledovat i na vzorových obrázcích. Ty bylo však, do jisté míry, možné eliminovat správným nastavením hyperparametrů teploty.

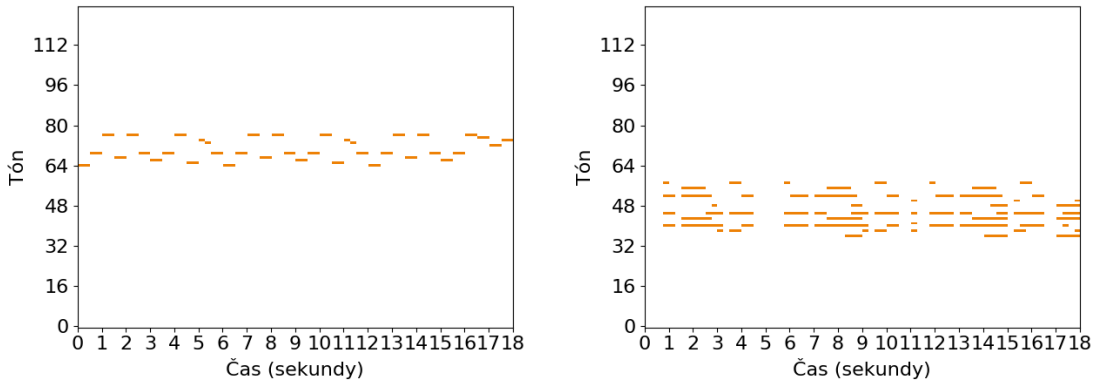
Zajímavý byl rozsah vygenerovaných akordů, který byl často větší než jedna oktáva. Součástí mnoha akordů tak navíc bylo několik stejných tónů, odlišujících se pouze oktávou.



(a) Vstupní melodie

(b) Vygenerovaný doprovod

Obrázek 5.4: Harmonie vygenerovaná diskriminativním modelem



(a) Vstupní melodie

(b) Vygenerovaný doprovod

Obrázek 5.5: Harmonie vygenerovaná generativním modelem

6 Závěr

Práce sestávala ze dvou hlavních částí. První částí bylo vytvoření co nejrozsáhlejší datové sady. Ta byla vytvořena zpracováním velkého množství volně dostupných MIDI souborů. Výsledkem byla trénovací, validační a testovací sada, tvořená dohromady více jak 50 tisíci normalizovanými záznamy. Záznamy se skládaly z partitur – melodických a harmonických sekvencí, kdy melodická sekvence byla tvořena sledem tónů a sekvence harmonická sledem akordů, uložených po jednotlivých tónech.

Druhou částí bylo natrénování modelu, který by k poskytnuté melodii vygeneroval harmonický doprovod. Výsledkem byly dva modely, oba založené na rekurentních sítích (konkrétně na LSTM paměťových buňkách). První model fungoval jako logistický klasifikátor, druhý model odhadoval rozdělení akordů vzhledem k melodii s použitím variačního autoenkodéru.

Specifikum obou modelů bylo, že generovaly doprovod po jednotlivých tónech. Důvodem této volby byla snaha poskytnout modelu větší volnost a učinit tak výsledný doprovod zajímavějším. Následkem toho by měl být model například schopný tvořit akordy, se kterými nepřišel do styku během fáze učení. Tento přístup však vedl k několika problémům.

Jedním z problémů byla například přítomnost redundantních tónů, nezávislých na aktuálně hraném akordu. Druhým problémem bylo špatném načasování některých akordů. Tyto jevy však bylo možné do značné míry omezit správným nastavením parametrů generátoru. Oba modely tedy byly schopny, za určitých podmínek, vygenerovat subjektivně odpovídající doprovod k poskytnuté melodii. Řešením by též mohlo být použití zpětněvazebního učení nebo nějaké formy postprocessingu. Ta však, z časových důvodů, nebyla do této práce zahrnuta.

Výsledkem práce byl i program, ovládány z příkazové řádky. Vstupem je libovolně dlouhá melodie, zapsaná v notaci, navržené speciálně pro tyto účely. Výstupem je pak hudební doprovod, zapsaný ve formátu MIDI. Program dále umožňuje ovládat některé vlastnosti vygenerovaného doprovodu, jakým je například tempo nebo výška tónů. Vstupní melodie a vygenerovaný doprovod je však redukovaný pouze na rozsah dvou oktáv. Důvodem pro toto rozhodnutí byl omezený výpočetní výkon, dostupný pro trénování modelu.

Literatura

[1] P. M. Todd, "A sequential network design for musical applications," in *Connectionist Models Summer School*, pp. 76–84, 1988.

[2] 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.

[3] J. Kiefer and J. Wolfowitz, "Stochastic estimation of the maximum of a regression function," *The Annals of Mathematical Statistics*, pp. 462–466, 1952.

[4] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, (USA), pp. 807–814, Omnipress, 2010.

[5] R. Pascanu, T. Mikolov, and Y. Bengio, "Understanding the exploding gradient problem," *CoRR*, vol. abs/1211.5063, 2012.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[7] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with lstm recurrent networks," *Journal of machine learning research*, vol. 3, no. Aug, pp. 115–143, 2002.

[8] D. P. Kingma and M. Welling, "Auto-encoding variational bayes.," *CoRR*, vol. abs/1312.6114, 2013.

[9] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.

[10] C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson, "One billion word benchmark for measuring progress in statistical language modeling," tech. rep., Google, 2013.

[11] C. Raffel, *Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching*. 2016.

[12] T. Bertin-mahieux, D. P. W. Ellis, B. Whitman, and P. Lamere, "The million song dataset," in *In Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR*, 2011.

[13] 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.

[14] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," in *Advances in neural information processing systems*, pp. 1019–1027, 2016.

[15] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[16] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio, "Generating sentences from a continuous space," *arXiv preprint arXiv:1511.06349*, 2015.

[17] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[18] S. Rajeswar, S. Subramanian, F. Dutil, C. J. Pal, and A. C. Courville, "Adversarial generation of natural language," *CoRR*, vol. abs/1705.10929, 2017.

[19] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, ACM, 2009.

[20] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[21] C. L. Krumhansl, *Cognitive foundations of musical pitch*. New York: Oxford University Press, 1990.