VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SEPARACE MLUVČÍCH V ČASOVÉ DOMÉNĚ

TIME DOMAIN AUDIO SEPARATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JIŘÍ PEŠKA**
AUTHOR

VEDOUCÍ PRÁCE **ing. KATEŘINA ŽMOLÍKOVÁ,**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Peška Jiří**
Program: Informační technologie
Název: **Separace mluvčích v časové doméně pomocí neuronové sítě**
Time-Domain Neural Network Based Speaker Separation
Kategorie: Zpracování řeči a přirozeného jazyka

Zadání:

1. Seznamte se s problémem separace mluvčích pomocí neuronových sítí.
2. Seznamte se s metodou TasNet pro jednokanálovou separaci signálu v časové doméně.
3. Implementujte danou metodu s využitím vhodného toolkitu (např. PyTorch, Keras).
4. Otestujte systém na vhodném datasetu. Zaměřte se na vyhodnocení vlivu velikosti sítě na přesnost.
5. Navrhněte a diskutujte možné zlepšení použité metody.

Literatura:

- Luo, Yi, and Nima Mesgarani. "Conv-TasNet: Surpassing Ideal Time-Frequency Magnitude Masking for Speech Separation." *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 27.8 (2019): 1256-1266.
- dle doporučení vedoucího

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Žmolíková Kateřina, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 14. května 2020
Datum schválení: 5. listopadu 2019

Abstrakt

Práce se zabývá využitím konvolučních neuronových sítí pro automatickou separaci mluvčích v akustickém prostředí. Cílem je implementovat neuronovou síť podle architektury TasNet za použití frameworku PyTorch, natrénovat síť s různými hodnotami hyperparametrů a porovnat kvalitu separací vzhledem k velikosti sítě.

Architektura oproti dosavadním metodám, které převáděly vstupní směs do časově–frekvenční reprezentace, používá konvoluční autoenkodér, který vstupní směs převádí do nezáporné reprezentace, která je optimalizovaná pro extrakci jednotlivých mluvčích. Samotné separace je docíleno aplikací masek, které jsou odhadnuty v separačním modulu. Modul tvoří opakující se posloupnost konvolučních bloků se zvyšující se dilatací, která napomáhá k modelování časových závislostí ve zpracovávané směsi.

K vyhodnocení přesnosti bylo použita metrika Signal to Distortion Ratio (SDR), která určuje poměr zastoupení šumu a zvuku v nahrávce. Vyhodnocování proběhlo na množině směsí dvou mluvčích, která byla vygenerována náhodnou kombinací mluvčích z Wall Street Journal datasetu (WSJ0). Natrénováním několika modelů s různými hodnotami hyperparametrů bylo možno pozorovat závislost mezi velikostí sítě a hodnotou SDR. Zatímco menší síť dosahovala, po X epochách trénování, přesnosti XY, větší síť dosahovala až XX.

[[Doplnit SDR presnost do odstavce vyse. Jeste 4. cast?]]

Abstract

[[Prelozit do anglictiny CZ abstrakt]] Thesis is about usage of convolutional neural networks for automational speech separation in acoustic environment. The goal is to implement neural network by following TasNet architecture in pytorch framework, train this network with various values of hyper parameters and to compare the quality of separation based on the size of the network.

In contrast to older architectures, that transformed input mixture into time–frequency representation, this architecture use convolutional autoencoder, which transforms input mixture into non–negative representation optimized for speaker extraction. Separation is achieved by applying the masks, which are estimated in the separation module. This module consists of stacked convolutional blocks with increasing dilation, which helps with modelling of long–term time dependencies in the processed speech.

For evaluation of preci. Ye.

Klíčová slova

neuronové sítě, zpracování řeči, konvoluční neuronová síť, autoenkodér, separace mluvčích, strojové učení, tasnet, feed forward, hluboké učení

Keywords

artificial neural networks, speech processing, convolutional neural networks, autoencoder, speech separation, machine learning, tasnet, feed forward, deep learning

Citace

PEŠKA, Jiří. *Separace mluvčích v časové doméně*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce ing. Kateřina Žmolíková,

Separace mluvčích v časové doméně

Prohlášení

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

.....
Jiří Peška
3. července 2020

Poděkování

V této sekci je možno uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc (externí zadavatel, konzultant apod.).

Obsah

Kapitola 1

Úvod

Zpracování řeči hraje v dnešní době důležitou roli v mnoha rozličných oborech. Mezi jeden z hlavních úkolů bezesporu patří separace zdrojů v zaznamenaném signálu, který může být složen ze signálů N mluvčích, ale i nechtěného hluku okolí. Vyřešení problému je předpoklad k dalším úkonům jako identifikace konkrétního mluvčího nebo třeba přepis konverzace na text. Se stále se zrychlujícím vývojem počítačů a s jejich zvyšujícím se výkonem se do popředí dostávají metody zpracování řeči založené na neuronových sítích, které v mnoha ohledech předčily ostatní algoritmy strojového učení.

Separace mluvčích v časové doméně dosahuje mimořádných výsledků v porovnání s dosavadními metodami založenými na převodu signálu z časové domény do frekvenční domény pomocí algoritmu STFT (Short–Time Fourier Transform). Taková reprezentace signálu není optimalizovaná pro separaci řeči a nemusí pro tento úkol být optimální. V architektuře, která je navržena v referenční studii s názvem **TasNet: Surpassing Ideal Time-Frequency Masking for Speech Separation** [12], je vstupní signál převeden do nezáporné reprezentace, která je optimální pro extrakci jednotlivých mluvčích. Silnou stránkou systému je hluboká architektura sítě, která lépe modeluje dlouhodobé závislosti v signálu. Zároveň se ale musí vypořádat s problémy, které hluboké neuronové sítě mohou přinášet.

Úkolem této práce je nastudovat si problematiku neuronových sítí a jejich základní principy, seznámit se problémem separace mluvčích pomocí neuronových sítí a následně implementovat síť podle architektury TasNet pro separaci mluvčích v časové doméně, která byla navržena a popsána ve studii [12]. Poté tuto neuronovou síť natrénovat s různými kombinacemi hodnot hyperparametrů, které ovlivňují velikost sítě a její vlastnosti, a nakonec porovnat přesnost a kvalitu separace mezi jednotlivými, různě velkými sitěmi a s výsledky, kterých bylo dosaženo ve studii. Přesnost a kvalita separace bude vypočítána pomocí metrik určených k hodnocení kvality řečového signálu. Sítě budou testovány a vyhodnocovány na testovací množině jednokanálových směsí dvou mluvčích.

Text je rozdělen na několik kapitol, které postupně pokrývají různé logické celky. V kapitole 2 je popsán problém separace mluvčích a dosavadní přístupy k jeho řešení. Kapitola 3 se zabývá základními vlastnostmi a principi neuronových sítí. Kapitola 4 je věnována architektuře TasNet, kterou se moje práce zabývá. Dále kapitola 5 popisuje implementaci sítě a pomocných nástrojů a nakonec v kapitole 6 jsou experimenty a vyhodnocování přesnosti sítě.

Kapitola 2

Separace mluvčích

Živočichové mají vrozenou schopnost zaměřit se na jeden konkrétní zvuk, zatímco všechny ostatní dokážou potlačit. Naučit ale této dovednosti počítač se ukázalo jako obtížný úkol, který je překážkou ASR (*automatic speech recognition*) systémům v rozsáhlejším nasazení do běžného života. Tento problém se nazývá koktejl párty. Vyřešení tohoto problému, tedy separování zdrojů, umožňuje širokou aplikaci v oblasti zpracování řeči, jako přepis konverzací na text, ovládání počítače hlasem, používání hlasových asistentů či identifikace konkrétního mluvčího v prostředí s mnoha současně mluvícími lidmi.

2.1 Koktejl párty efekt

Tento problém se nejlépe dá popsat jako schopnost soustředit se na jeden konkrétní zdroj zvuku z mnoha souběžných zdrojů jako hudba, konkrétní hudební nástroj, řeč dalších lidí či okolní zvuky, a přepínat mezi nimi dynamicky pozornost.

[[vzniklo pro jeho vyreseni hafo altofirmu, ktere dosahovaly nejakych vysledku. jeho vyresenim by se mohlo pouzivat asr na prepis reci an text a pod. Nejlepsich vysledku dosahly hluboke neuronove site.]]

2.2 Metody pro separaci jednokanálových nahrávek

Pro separaci zdrojových signálů z nahrávek směsí existuje mnoho algoritmů založených na zpracování signálů, z nichž asi nejznámější jsou computational auditory scene analysis (*CASA*)**[[ref?]]** a non-negative matrix factorization (*NMF*) **[[ref?]]**, používané pro separaci jednokanálových nahrávek. V posledních letech se do popředí dostaly přístupy založené na hlubokých neuronových sítích, které předčily svou přesností a výkonností dosavadní algoritmy. Pro separaci vícekanálových nahrávek se používají metody *beamforming* nebo state–of–the–art *Multi-channel blind source separation*, které ale jsou nad rámec této práce. Tato kapitola shrnuje některé metody pro řešení separace, které jsou zmíněny v článku [14].

2.2.1 CASA – computational auditory scene analysis

Metoda CASA je založena na procesu v lidském mozku, který separuje zdroje ze směsi mluvčích a simuluje vysokoúrovňové chování lidského sluchu. Pro separaci jsou většinou manuálně navrhnuta segmentační pravidla pro operování nad nízkoúrovňovými příznaky k odhadu časově frekvenční (T–F) masky, která izoluje komponenty signálu jednotlivých

mluvčích a následně je použita pro rekonstrukci všech zdrojů. **[[Hodit sem teda ty zdroje jako v 3.1 pro ASA, CASA a pod? jak jsou tam asi 3 po sobe?]]**

Přestože tato metoda byla i nadále rozvíjena, tak má mnoho nevýhod jako špatnou generalizaci v důsledku manuálního vytváření pravidel, nemožnost automatického učení z dat nebo nemožnost použití na separaci jiných zdrojů než je lidská řeč, které značně omezují její použití v mnoha reálných případech.

2.2.2 NMF – non-negative matrix factorization

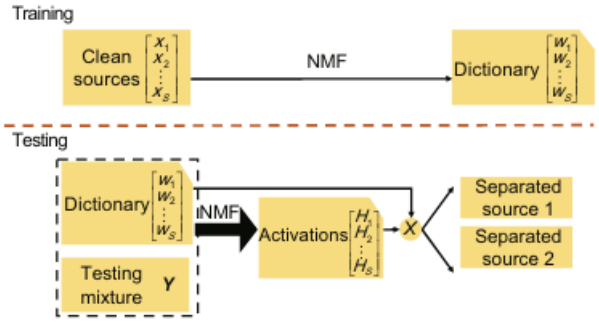
Nezáporná maticová faktorizace (*NMF*) **[[[(Lee and Seung, 2001)]]]** patří mezi daty řízené metody a je založena na předpokladu, že struktura spektrogramu může být reprezentována malým počtem bází.

Pro NMF platí

$$Y = \sum sW_sH_s \tag{2.1}$$

kde každý zdroj s je modelován aproximací nízkého řádu nezápornými maticemi W_s, která reprezentuje slovník a H_s, která reprezentuje aktivační funkci. Ty jsou sečteny a dohromady tvoří výslednou směs Y.

Metoda prochází trénovací a testovací fází. V trénovací fázi je každý zdroj dekomponován a mapován na množinu bází a aktivací a tím je zformován slovník W pro tento zdroj. V testovací fázi jsou naučené slovníky jednotlivých zdrojů s spojeny do jednoho, který dále není upravován. Následuje optimalizace aktivační matice H pro každý zdroj. Každý zdroj je následně ze směsi rekonstruován s použitím výsledných bází a aktivací.



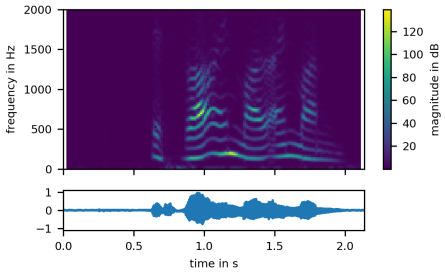
Obrázek 2.1: Znázornění trénovací a testovací fáze. Obrázek byl inspirován [14]

Stejně jako metoda CASA, i tato má mnoho nevýhod, které omezují její použití pro řešení reálných problémů separace.

2.2.3 Hluboké neuronové sítě

Nejlépe si při řešení problému koktejl párty vedou postupy založené na hlubokém učení a neuronových sítích. Metody převáděly směs do časově–frekvenční reprezentace znázorněné na obrázku 2.2 pomocí STFT (*short time fourier transformation*).

[[rozepsat]]



Obrázek 2.2: Časově–frekvenční reprezentace řeči, neboli spektrum

2.3 Vyhodnocovací metriky

Pro vyhodnocení kvality separace se používají různé metriky. Jejich volba závisí na řešeném problému.

2.3.1 Signal–to–noise ratio

Nejzákladnější metrika pro určení kvality separace je signal–to–noise ratio (SNR), která porovnává zastoupení chtěného signálu a šumu v pozadí a je definována jako poměr energie originálního (ideálního) signálu a energie rozdílu mezi originálním a rekonstruovaným signálem.

$$SNR = \frac{E_s}{E_N} \tag{2.2}$$

Výsledný poměr často bývá vyjádřen v decibelech jako

$$SNR(dB) = 10log_{10}\frac{E_s}{E_N} \tag{2.3}$$

kde kladná SNR hodnota udává, že zastoupení signálu je větší, než zastoupení šumu [[**citovat https://labrosa.ee.columbia.edu/ dpwe/pubs/Ellis04-sepeval.pdf**]].

2.3.2 SDR – Source Distortion Ratio

[1] SDR je jedna z nejznámějších metrik pro měření kvality řečového signálu. [[**rozepsat**]]

2.3.3 PESQ – Perceptual evaluation of speech qualit

[2]

Procentuální vyhodnocení kvality řeči (PESQ) [15] je standardizovaná metrika pro vyhodnocování kvality řeči. Jeho hodnota se pohybuje od -0.5 do 4.5 s tím, že vyšší hodnota znamená lepší procentuální kvalitu [10].

[[**rozepsat**]]

[1] https://github.com/sigsep/bsseval
[2] https://github.com/ludlows/python-pesq

2.3.4 STOI

[3] Bylo navrženo mnoho konvenčních přístupů a metod pro řešení problému koktejl párty a s ním spojené separace mluvčích, ale většina z nich nedosahuje dostatečné přesnosti nebo výkonu k použití v reálných situacích. Přístupy založené na hlubokém učení a neuronových sítích předčily dosavadní techniky jak v přesnosti, tak ve výkonu, což umožňuje jejich nasazení v běžném životě.

[[**rozepsat**]]

[3] https://github.com/mpariente/pystoi

Kapitola 3

Neuronové sítě

V dnešní době zažívají neuronové sítě díky výkonosti počítačů velký rozmach. Jejich využití prostupuje skrze mnohé vědní obory a dokáží řešit celou řadu problémů, ve kterých dosahují výborných výsledků, které zdaleka předčily dosavadní postupy.

Neuronové sítě (*artificial neural networks*) jsou výpočetní model, který je inspirovaný strukturou lidského mozku, ve kterém je obrovské množství propojených a komunikujících neuronů. Ty se skládají ze vstupních dendridů, výstupních axonů a samotného těla neuronu. Na základě vnitřního potenciálu a vstupních hodnot je po přesažení prahové hodnoty neuron vybuzen a je vyslán signál na výstupní axon. Signál je nakonec předán dalším neuronům skrze jejich vstupní dendridy [9, p. 65–66].

Účelem neuronové sítě je naučit se plnit zadanou úlohu. Rozdíl oproti běžným algoritmům je ale ten, že způsob, jakým síť má problém řešit, není explicitně naprogramován, ale je postupně naučen. Základní způsoby učení jsou s učitelem (*supervised*) a bez učitele (*unsupervised*).

Učení s učitelem, pod které spadá i tato práce, spočívá v mapování vstupních dat na data výstupní na základě vzorových příkladů dvojic vstup–výstup. Množině takových dvojic se říká trénovací dataset. Další vlastností trénovacích dat určených pro učení s učitelem, je jejich označení (*label*).

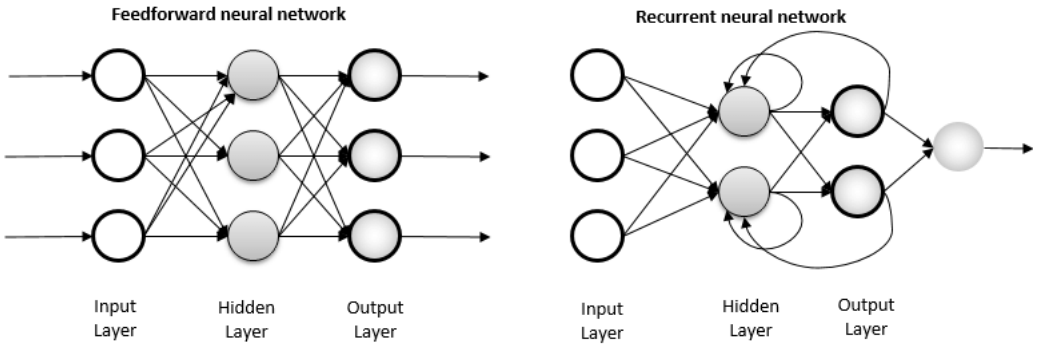
Mezi problémy, které se dají řešit neuronovými sítěmi patří klasifikační a regresní problémy. Konkrétní příklad z oblasti klasifikace může být rozpoznávání objektů na obraze, psaného písma nebo detekce obličejů na videu, ale i mnohé aplikace ve zpracování řeči. Některé z problémů lze řešit sítí, která se skládá z jednotek neuronů, zatímco složitější problémy vyžadují mnohem větší kapacitu sítě a ta pak může obsahovat i tisíce neuronů.

Upravovat samotnou strukturu a chování neuronové sítě lze pomocí jejích hyperparametrů, což jsou parametry určující nastavení neuronové sítě a trénovacího algoritmu. Tyto parametry musí být určeny před začátkem trénování a většinu nelze později měnit. Mohou určovat kapacitu modelu, velikost záběrného pole, velikost filtrů, ale i regulovat samotný proces učení nastavením počtu epoch, po které se model má učit, nebo také počet dat předaný v jedné dávce (tzv. *minibatches*) během učení sítě.

3.1 Organizace feedforward sítí

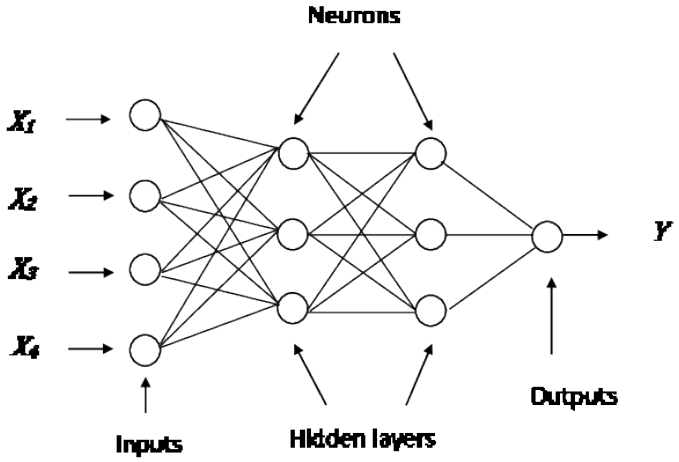
Feed forward neuronové sítě (*Multi Layer Perceptron – MLP*) jsou typ umělých neuronových sítí, kde se nevyskytují cykly ve výpočetním grafu, tedy zpětná propojení vrstev, takže informace se pohybuje pouze jedním směrem, od vstupní vrstvy přes skryté vrstvy

až po vrstvu výstupní. Sítě, které obsahují cykly, se nazývají rekurentní. Rozdíl znázorňuje obrázek 3.1.



Obrázek 3.1: Příklad grafu feed forward sítě a rekurentní neuronové sítě. Lze si všimnout orientace šipek u feed forward sítě, které směřují pouze jedním směrem, zatímco u rekurentní sítě šipky směřují i k předešlým uzlům grafu [13].

Struktura neuronové sítě je organizována do vrstev, které se skládají z neuronů. Feedforward síť je tvořena třemi typy vrstev (viz obrázky 3.1 a 3.2). Vstupní vrstva slouží k předání hodnot do sítě, ale nijak tyto hodnoty nemodifikuje. Nezměněné jsou zkopírovány do první skryté vrstvy. Následují skryté vrstvy, z nichž poslední je napojena na výstupní vrstvu. Ta má obvykle méně neuronů než předešlé vrstvy a hodnoty na výstupu mohou představovat třídy, do kterých má být klasifikován vstup v případě klasifikátorů, nebo predikce hodnot na základě vstupních dat v případě regrese. S počtem jednotlivých vrstev souvisí pojem hloubka sítě, která je rovna počtu všech vrstev neuronové sítě od vstupní až po výstupní vrstvu. Pojem **[[uvozovky lepe]]** "hluboká neuronová síť"označuje takovou síť, která má dvě nebo více skrytých vrstev. Existuje mnoho typů vrstev, například plně propojené, pooling, s přeskočením nebo vrstvy konvoluční.

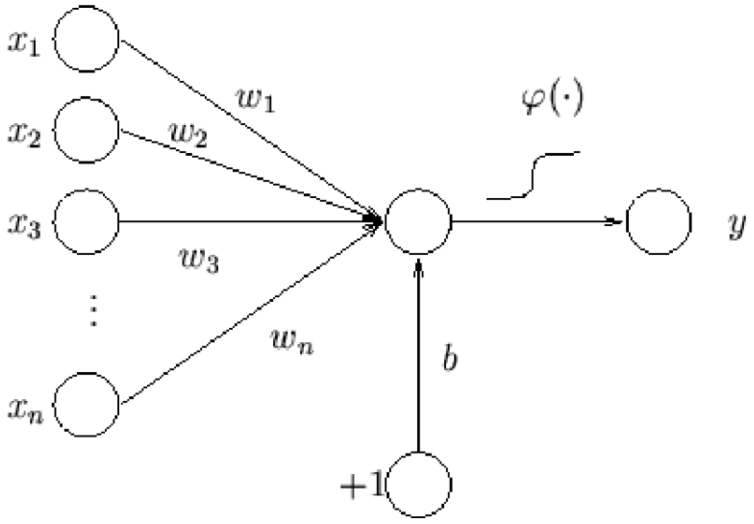


Obrázek 3.2: Schéma neuronové sítě, která má 2 skryté vrstvy

3.2 Umělý neuron

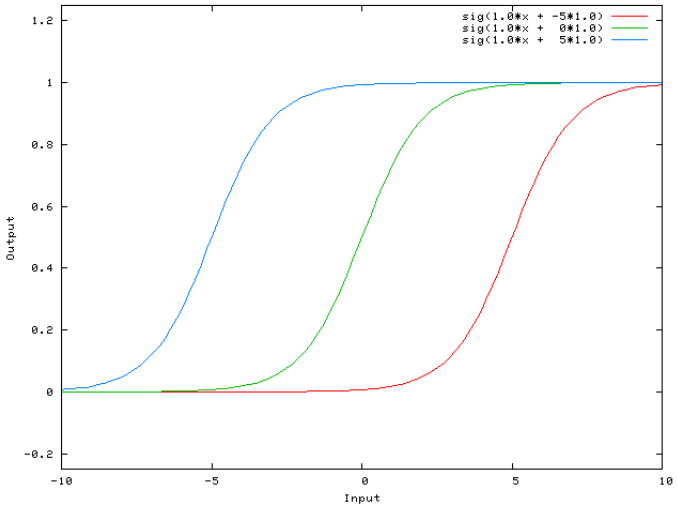
Základní stavební jednotka neuronových sítí je umělý neuron (*artificial neuron*) (viz obrázek 3.3). Tento model je založen na principu reálných neuronů, které se nacházejí v organizmech. Umělý neuron obsahuje libovolně mnoho vstupních propojení, přes které se mu předávají data v podobě vstupního vektoru $\overrightarrow{x} = [x_1, x_2, \ldots, x_n], x_n \in \mathbb{R}$. Sám neuron obsahuje hodnotu bias $b \in \mathbb{R}$ a vektor vah $\overrightarrow{w} = [w_1, w_2, \ldots, w_n], w_n \in \mathbb{R}$, jenž je upravován během trénování neuronu.

Výstupní hodnota závisí na vstupních datech, aktuálním vnitřním stavu (hodnoty vah a biasu) a na zvolené aktivační funkci. Vstupní hodnoty jsou váhovány, což znamená, že každá vstupní hodnota je vynásobena s váhou na daném vstupním spojení. S použitím definovaných vektorů lze napsat, že vstupní vektor je vynásoben s vektorem vah.



Obrázek 3.3: Schéma umělého neuronu

Hodnota bias b, která je přičtena k sumě násobků vah a vstupních hodnot, je prahová hodnota modifikující dobu, kdy se aktivuje neuron a změní svůj výstup. Matematicky to znamená, že s grafem aktivační funkce horizontálně pohybuje doleva nebo doprava v závislosti na tom, je-li hodnota biasu pozitivní nebo negativní. Toto posunutí je znázorněno na obrázku 3.4. V závislosti na řešeném problému může být žádoucí, aby i hodnota bias byla modifikována během učení společně s ostatními váhami.



Obrázek 3.4: Vliv hodnoty bias na aktivační funkci

Výstup neuronu se vypočítá jako

$$y = f\Big((\sum_{k=1}^{n} w_k x_k) + b\Big) \tag{3.1}$$

kde f je aktivační funkce, $x_k \in \mathbb{R}$ je vstupní hodnota, $w_k \in \mathbb{R}$ je váha, kterou se vstupní hodnota vynásobí a $b \in \mathbb{R}$ je hodnota bias.

3.3 Aktivační funkce

Aktivační, neboli prahová funkce, určuje výstupní hodnotu neuronu. Funkce se vybírá na základě problému, který se má neuronová síť naučit řešit. Správná volba aktivační funkce vede k lepší konvergenci učení sítě. Naopak špatná volba může vést ke stále větší odchylce od správného řešení – může divergovat. Povaha problému může vyžadovat specifické vlastnosti aktivační funkce – lineární nebo nelineární. Pro nestandardní problémy je obvykle potřeba experimentálně zjistit, která funkce bude nejlépe vyhovovat danému problému.

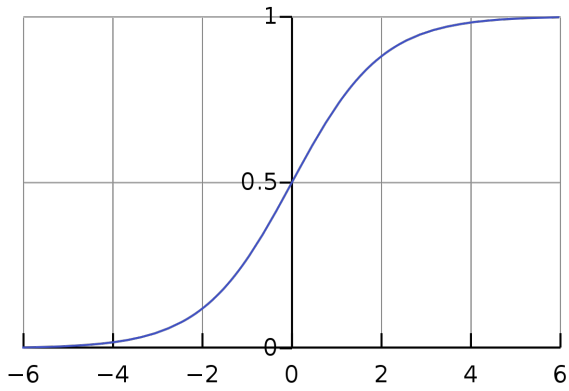
Pokud by veškeré aktivační funkce v modelu byly lineární, tak celkové mapování sítě by bylo omezeno pouze na lineární mapování vstupu na výstup. Reálné problémy ale lineární obvykle nejsou a v případě pokusu modelovat takovým modelem nelineární vztahy by vedlo k velice nepřesným výsledkům, který by byl zapříčiněn podučením (*underfitting*), což znamená, že model, který se učí zakódovat nějaký vzor v datasetu, je příliš jednoduchý. Proto je potřeba zavést do modelu i nelineární aktivační funkce, které tento problém řeší [9, p. 77–78].

Z pohledu učení je také důležité, aby aktivační funkce byla diferencovatelná. To umožňuje použití učících metod založených na výpočtu gradientu, jako je algoritmus backpropagation.

Logická sigmoida

$$f(x) = \frac{1}{1 + \exp(-x)} \tag{3.2}$$

[[popsat]]

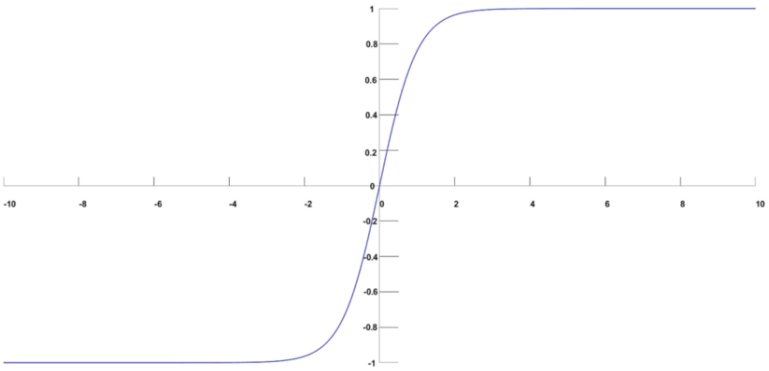


Obrázek 3.5: Graf aktivační funkce sigmoid

Softmax

$$f(x) = \frac{1}{1 + \exp(-x)} \tag{3.3}$$

[[popsat]]

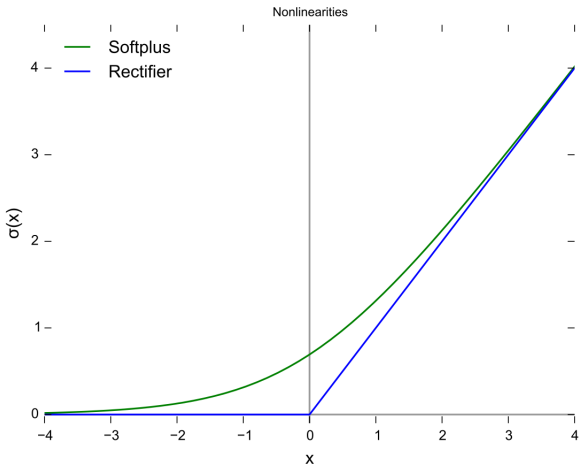


Obrázek 3.6: Graf aktivační funkce softmax

ReLU

Rectified Linear Unit je nejčastěji používaná aktivační funkce. Vyžaduje-li neuronová síť nějakou nelinearitu, je ReLU pro většinu případů ideální. Pro každou zápornou hodnotu x vrací 0 a pro kladnou hodnotu x vrací tutéž hodnotu x, jak udává rovnice

$$f(x) = max(0, x) \tag{3.4}$$

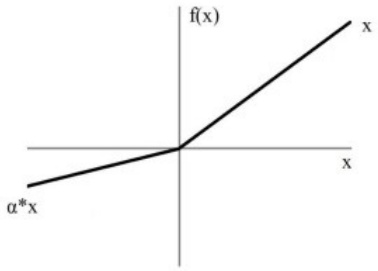


Obrázek 3.7: Graf aktivační funkce ReLU

PReLU

Parametrizovaná ReLU (PReLU) [6] je nelineární aktivační funkce, která se používá v případě, že chceme produkovat na výstup malý nenulový gradient i v případě záporné vstupní hodnoty x. V tom případě je vstupní hodnota vynásobena parametrem α a to představuje výsledek. Parametr α se společně s ostatními váhami učí během učícího procesu.

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases} \tag{3.5}$$



Obrázek 3.8: Graf aktivační funkce PReLU

3.4 Konvoluční vrstvy

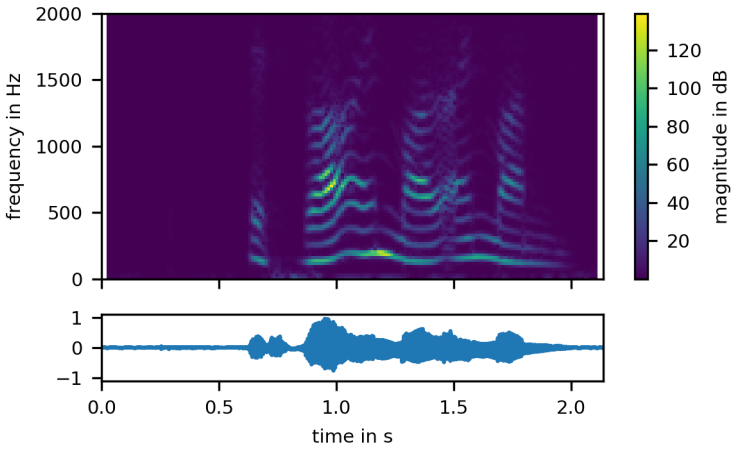
Konvoluční neuronová síť (CNN) [11] je typ feed forward neuronových sítí, která obsahuje jednu či více konvolučních vrstev. CNN jsou vhodné pro zpracování dat v mřížkovitém uspořádání. To může být ve 2–D obrázek ve formě pole pixelů či spektrum nahrávky, nebo vzorky zvukové nahrávky v čase v 1–D. Konvoluční vrstvy používají místo běžného násobení matic speciální typ lineární operace – konvoluci.

Diskrétní konvoluce je definována jako

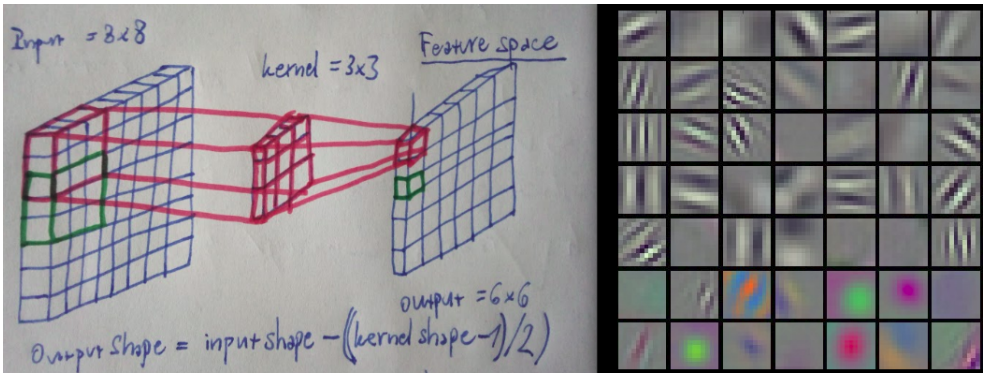
$$(f \star g)_k = \sum_{i=-\infty}^{\infty} f_i g_{k-i} = \sum_{i=-\infty}^{\infty} f_{k-i} g_i \tag{3.6}$$

kde $\star$ je konvoluční operátor, f je funkce signálu, funkce g je konvoluční jádro a f_i a g_i jsou hodnoty funkce na indexu i.

Účelem CNN je v prvních vrstvách sítě extrahovat lokální příznaky ze vstupu. Pro 2–D vstup to mohou být úsečky v různých úhlech nebo části křivek a pro 1–D tóny o různé frekvenci. V pozdějších vrstvách neurony kombinují extrahované lokální příznaky do komplexnějších příznaků (oko, nos), až dokud nesformují v konečných vrstvách například celý obličej, znak, kočka či jiný objekt, který se síť učí detekovat. V některých modelech zpracování řeči je zvuk transformován na 2–D spektrogram (viz obrázek 3.9) a lze s ním pracovat stejně, jako při zpracování obrazu.



Obrázek 3.9: Ukázka nahrávky řeči a korespondující spektrogram, na němž jsou zobrazeny frekvence (osa y) z nahrávky řeči v čase (osa x). Zvuk se na spektrogram převádí pomocí operace short time fourier transform (STFT)



Obrázek 3.10: Obrázek vlevo zobrazuje, jak probíhá konvoluování jádra se vstupními 2–D daty a mapování lokálních příznaků na prostor příznaků (feature space), což je výstupem konvoluce. Jádro je postupně aplikováno skrze celý vstup. Obrázek napravo zobrazuje, jak mohou vypadat lokální příznaky.

Konvoluování jádra přes nějaký vstup (například obrázek či zvuk) je ekvivalentní s detektorem, který detekuje nějaký lokální příznak. Při aplikaci takového detektoru přes nějaký vstup zaznamenává všechny pozice, kde se příznak nacházel [9].

Během trénování CNN jsou učeny váhy sítě (filtry) k detekci různých úrovní komplexnosti příznaků. Aby síť dokázala detekovat příznak nezávisle na jeho transformaci (pootočení, převrácení, posuvu), měla by být invariantní. Toho lze docílit tak, že nneurony budou sdílet některé váhy (filtry).

Konvoluční operace je definována jako

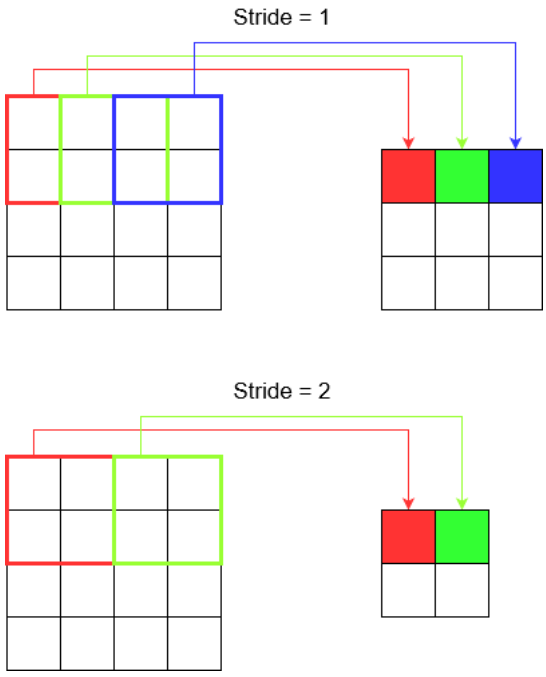
$$s[n] = (x \star w)[n] \tag{3.7}$$

kde x jsou vstupní data obvykle ve formě multidimenzionílního pole (tenzoru) a funkce w je konvoluční jádro. Výstupem operace je mapa příznaků. Konvoluční operaci lze definovat i pro vyšší dimenze. Máme–li například 2–D vstupní obrázek, pak i konvoluční jádro bude dvourozměrné. Přestože v mnoha knihovnách lze nalézt implementovanou operaci konvoluce, často se jedná o cross–korelaci.

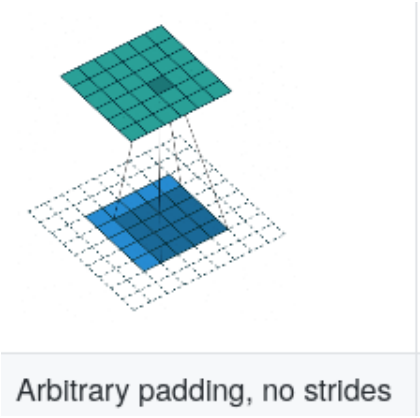
Průběh a chování konvoluce lze řídit hyperparametry *stride*, *padding* a *dilation*. Hodnota stride modifikuje velikost kroku konvolučního jádra. Výchozí hodnota kroku je 1. Chování při různých hodnotách lze vidět na obrázku 3.11.

Hodnota padding určuje, o kolik se mají rozšířit vstupní data. V případě malého rozměru konvolučního jádra může dojít k nechtěné změně rozměrů vstupních dat a proto se tato data po stranách rozšíří na takový rozměr, aby po konvoluci byl výsledný tvar v požadovaném rozměru. Vyplnění ukazuje obrázek 3.12.

[[Tyto tri obrazky stylove sjednotit a dat je vedle sebe mozna]]

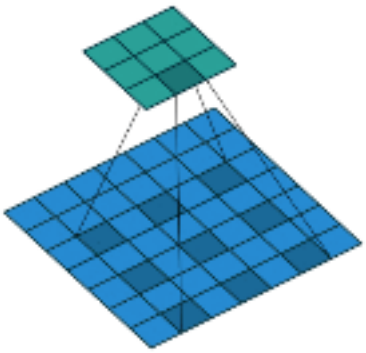


Obrázek 3.11: Hodnota stride ovlivňuje konvoluční krok jádra nad zpracovávanými daty. Čím větší hodnota stride, tím menší je výstupní rozměr dat



Obrázek 3.12: Nastavením hodnoty padding se zpracovávaná data na okrajích rozšíří o danou hodnotu a toto rozšíření bude vyplněno nulami či jakoukoli jinou hodnotou

Hodnota dilation podporuje rozšíření pole působnosti bez ztráty rozlišení nebo pokrytí dat konvolučním jádrem během konvoluční operace (viz obrázek 3.13). Zatímco pole působnosti roste exponenciálně, počet parametrů roste pouze lineárně. Této vlastnosti se využívá například v temporal convolutional networks (TCN), kde dilation faktor nabývá hodnot $1, 2, 4, 8, \ldots$.



Obrázek 3.13: Vizualizace hodnoty dilation upravující pole působnosti

3.4.1 Bottle–neck vrstva

[[Mozna, kdyz bude cas]]

3.5 Učení neuronových sítí

Cílem trénování je naučit neuronovou síť vykonávat nějaký úkol. Trénování založené na metodě učení s učitelem vyžaduje dostatečně velký dataset obsahující dvojice vstup–výstup a obnáší volbu objektivní funkce, kterou se během trénování snažíme minimalizovat.

Formálně je cílem učení aproximovat nějakou funkci f^*. Síti je předána vstupní hodnota x, pro kterou síť definuje mapování na výstupní hodnotu jako $y = f(x; \theta)$, kde θ je parametr, který se síť učí tak, aby dosáhla nejlepší aproximace funkce [7, p. 163].

3.5.1 Objektivní funkce

Objektivní funkce (loss funkce, cost funkce) je funkce, která je během učení minimalizována nebo maximalizována, v závislosti na konkrétním úkolu a kontextu použití. Tato funkce mapuje událost či hodnoty na reálné číslo reprezentující [[**spravit uvozovky**]] "cenu", která je asociována s touto událostí či hodnotami. Je–li tato funkce použita pro optimalizační problém, pak je cílem hodnotu minimalizovat a funkci se pak říká loss funkce, protože její výstupní hodnota nám udává velikost chyby, která se počítá na základě rozdílu mezi výstupem sítě a odpovídajícími trénovacími daty z datasetu. Pak platí, že čím menší chyba, tím lépe síť provádí svůj úkol, na který je trénována. Použije–li se hodnota objektivní funkce s opačným znaménkem, tak lze funkci použít jako hodnotící metriku.

V rámci učení sítě je žádoucí, aby gradient objektivní funkce byl dostatečně velký (prudký) a předvídatelný. V takovém případě bude dobře sloužit pro účely trénování. V případě malého gradientu by funkce saturovala (byla by příliš plochá) a to by mohlo negativně ovlivnit trénování sítě [7].

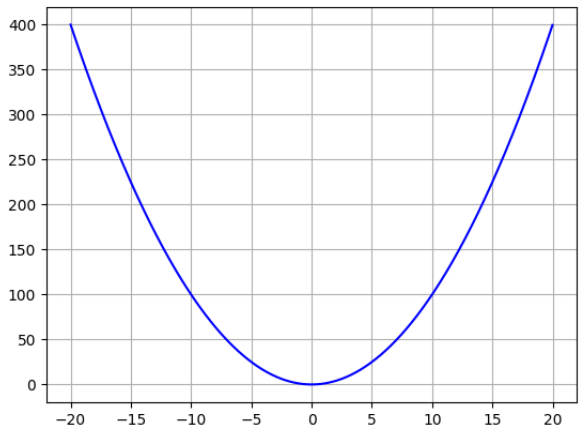
Nejčastěji se pro regresní problémy používají objektivní funkce *Mean Squared Error Loss* a *Mean Absolute Error Loss*.

MSELoss (Mean Squared Error Loss)

MSELoss je nejčastěji používaná objektivní funkce při řešení regresních problémů. Vypočítá se jako

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{3.8}$$

kde N je počet trénovacích dat v datasetu, y_i je předpovězená hodnota a $\hat{y}_i$ je odhadnutá hodnota. Vzhledem k druhé mocnině je výsledek vždy pozitivní nezávisle na znaménku hodnot y_i a $\hat{y}_i$, jak lze vidět na obrázku 3.14. Druhá mocnina také zajišťuje, že čím větší je rozdíl mezi předpovídanou a aktuální odhadnutou hodnotou, tím více se chyba projeví. Funkce dosahuje L2 regularizace.



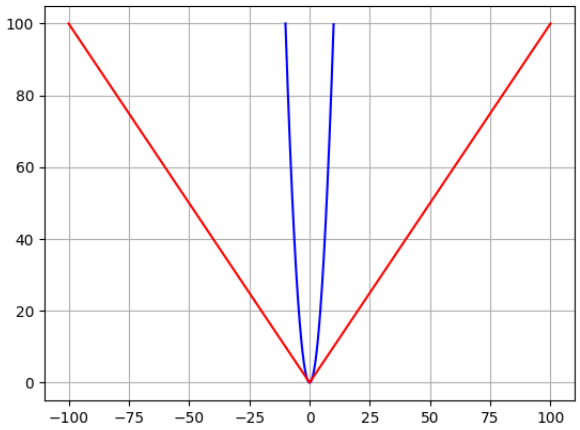
Obrázek 3.14: Graf objektivní funkce MSE Loss

MAELoss (Mean Absolute Error Loss)

MAE je velmi podobná objektivní funkce jako MSE, ale s téměř opačnými vlastnosti. Stejně jako MSE, ani tato funkce nenabývá negativní hodnoty, ale narozdíl od MSE, která má tuto vlastnost díky druhé mocnině rozdílu, MAE toho dosahuje tak, že rozdíl předpovídané hodnoty y_j a odhadnuté hodnoty $\hat{y_j}$ je uzavřen v absolutní hodnotě. Vypočítá se jako

$$MAE = \frac{1}{N}\sum_{j=1}^{N}|y_j - \hat{y_j}| \tag{3.9}$$

Výhodnou MAE je její lineární průběh, takže se větší chyby projeví úměrně více než chyby menší, oproti MSE, kde je závislost kvadratická. Nevýhodou je, že kvůli absolutní povaze není diferencovatelná v hodnotě $x = 0$, což může mít negativní následky pro výpočet gradientu.



Obrázek 3.15: Graf objektivní funkce MAE Loss (červeně)

[[Hinge loss mozna]]

3.5.2 Backpropagation

Backpropagation (backprop) je algoritmus typu učení s učitelem pro učení neuronových sítí. Algoritmus zpětně prochází neuronovou síťí a na základě hodnoty cost funkce $J(\theta)$ počítá její gradient metodou *gradient descent*. Cílem je upravit všechny váhy v síti na základě toho, jak moc každá váha přispívá k celkové chybě. Algoritmus backpropagation počítá řetězové pravidlo (*chain rule*).

Algoritmus se skládá ze 3 opakujících se kroků. Prvním krokem je forward fáze, ve které se počítá pro každou dvojici vstupu a referenčního výstupu

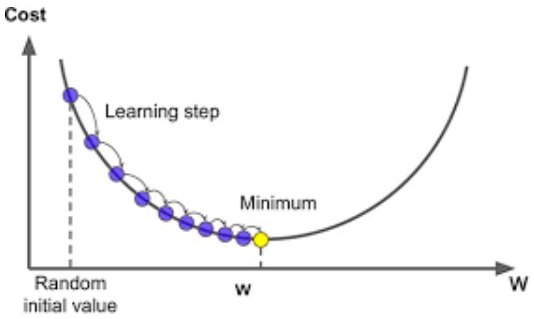
[**?**] [**?**, p 197]

- zpetne sireni chyby - adaptacni algoritmus, podil neuronu na chybe, - 3 opakujici se faze uceni:

1) feedforward - dopredu 2) zpetne sireni chyby - Backpropagation 3) uprava vah a biasu na zaklade chyby pomoci gradient descent - chain rule

Gradient descent

Gradient descent je iterativní optimalizační algoritmus pro hledání lokálního minima diferencovatelné objektivní funkce $J(\theta)$, kde $\theta \in \mathbb{R}^d$ reprezentuje učené parametry modelu, úpravou těchto parametrů v opačném směru, než je hodnota gradientu. Výpočet je prováděn postupným pohybem ve směru největšího klesání, které je určeno zápornou hodnotou gradientu. Rychlost pohybu závisí na velikosti kroku, což udává *learning rate*. Správná volba *learning rate* ovlivňuje rychlost, jakou je nalezeno minimum funkce. Při nízké hodnotě bude výsledek přesnější, ale nalezení minima bude výpočetně náročnější, protože v každém kroku se počítá nová hodnota gradientu. Při větší hodnotě je riziko, že minimum bude přeskočeno [16].



Obrázek 3.16: Ukázka algoritmu gradient descent při hledání minima dané funkce postupným přibližováním dle nastavené velikosti kroku

[[popsat vice vliv LR na loss]]

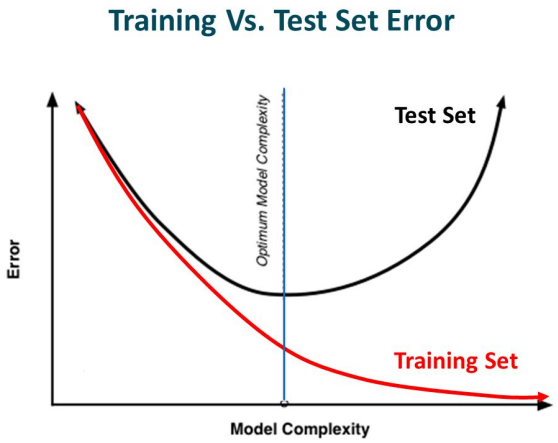
3.5.3 Underfitting a generalizace

Generalizace, neboli *test error*, je vlastnost modelu, která udává, jak dobře model pracuje s dříve neviděnými daty, tedy s daty, které nebyly použity pro jeho učení. Během učení vzniká učící chyba, která je trénováním postupně minimalizována, ale kvalita modelu je pak měřena na testovacích datech. Testovací chyba by ideálně měla být co nejblíže chybě na trénovací sadě. Z rozdílu mezi těmito chybovými hodnotami se dají diagnostikovat problémy s naučením sítě – podučení (*underfitting*) a přeučení(*overfitting*).

Underfitting nastane v případě, že model již nedokáže v důsledku jeho kapacity zmenšit dostatečně jeho chybovou hodnotu na trénovacím datasetu.

Overfitting nastane, když rozdíl mezi trénovací chybou a testovací chybou je příliš velký.

Kapacita je vlastnost modelu určující, kolik se toho dokáže model naučit. V případě nízké kapacity se model nedokáže naučit všechny příznaky z trénovacích dat a v případě vysoké kapacity se mohou přeučit zapamatováním si jejich příznaků a tím sice dosáhnou nízké trénovací chyby, ale špatných výsledků během testování [7, p107].



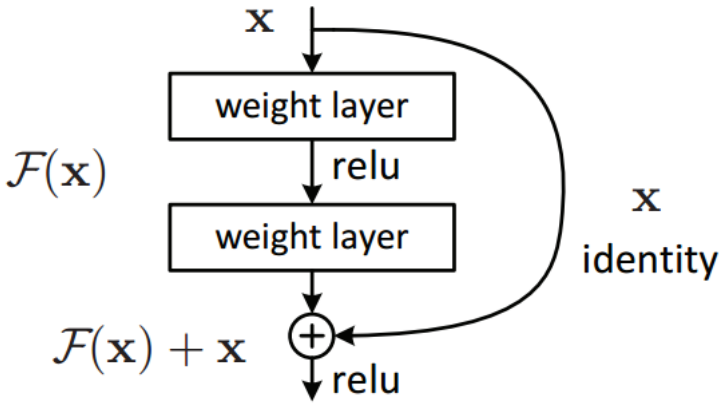
Obrázek 3.17: Vliv kapacity modelu trénovací a testovací chybu. Obrázek inspirován podle [3]

3.5.4 Regularizace

Regularizace je souhrn postupů, které říkají, jak modifikovat učící algoritmus tak, aby se zredukovala chyba generalizace, zatímco testovací chyba zůstane stejná [7, p117].

3.5.5 Reziduální spojení

Velmi hluboké neuronové sítě přinesly v mnoha aplikacích velmi dobré výsledky. S přibývající hloubkou se ale stává její trénování složitější. Jeden z problémů, který se při učení projevoval, byl explodující a mizející gradient [2]. Tento problém byl adresován zavedením normalizačních vrstev [8] a počáteční optimalizované inicializace, což dovolilo sítím konvergovat s využitím *stochastic gradient descent* se zpětnou propagací (*back–propagation*) [11]. Poté se objevil problém degradace, který nebyl zapříčiněn přetrénováním, a způsobil, že s přibývající hloubkou sítě její přesnost náhle prudce klesla, což indikovalo, že systémy nelze optimalizovat stejným způsobem a přidání dalších vrstev pouze zvýšilo trénovací chybu. Problém degradace lze řešit zakomponováním reziduálních spojení, které je zobrazeno na obrázku 3.18 [5].



Obrázek 3.18: Reziduální spojení mezi vstupem a výstupem stavebního bloku sítě [5]

[[prepsat F na to specialni]] Reziduální spojení, neboli identitní mapování, je zkratka mezi jednou či více vrstvami. Problém mizejícího gradientu řeší tím, že používá aktivace z předchozí vrstvy, dokud váhy následující vrstvy nejsou naučeny. Spojení lze formulovat jako $F(x) + x$, kde $F(x)$ je výsledná hodnota transformace jednotlivými vrstvami a x je původní vstup, který vrstvy přeskočí a následně je sečten s výstupem $F(x)$. Výhodou je, že jeho použitím se nezvyšuje počet parametrů ani výpočetní náročnost [5][18].

3.5.6 Význam validační množiny v trénování

[[Mozna ne jako subsekci ale jen jako jeden odstavec]] Většina algoritmů strojového učení má nějakou sadu hyperparametrů, kterou je upravováno chování algoritmu. Hodnoty hyperparametrů obvykle bývají nastavovány ručně ještě před spuštěním procesu učení a hodnota se v průběhu nemění, protože hodnoty by bylo obtížné optimalizovat. Některá nastavení se nicméně mohou stát hyperparametrem a být upravována během trénování, ale není vhodné je měnit na základě výsledku učení na trénovací sadě, protože by mohlo dojít k přetrénováníoverfitting) v důsledku **[[CEHO??]]**. Pro tento případ potřebujeme validační sadu, která je odlišná od trénovací sady. Po každém zpracování trénovací sady následuje validační sada, po jejímž skončení jsou optimalizovány hyperparametry[][]. **[[[kniha 117-118] kap 5.3 = Hyperparameters and validation set]] [[najit jeste nejakej zdroj s popisem a pripadne nejaky zajimavejsi info.]]**

V této kapitole byly popsány základní principy neuronových sítí, jejich základní struktura a stavební bloky. V návaznosti na to byly zmíněny konvoluční sítě, které jsou pro tuto práci stěžejní. Dále byl popsán proces trénování sítí a algoritmy back–propagation a gradient descent. Nakonec byly zmíněny některé oblasti, které se zabývají optimalizacemi sítě, kladoucí si za cíl zlepšit například dobu trénování nebo kvalitu naučení sítě, což může mít za následek lepší výkon sítě při její evaluaci nebo umožnit nasazení ve zdrojově omezeném prostředí.

Kapitola 4

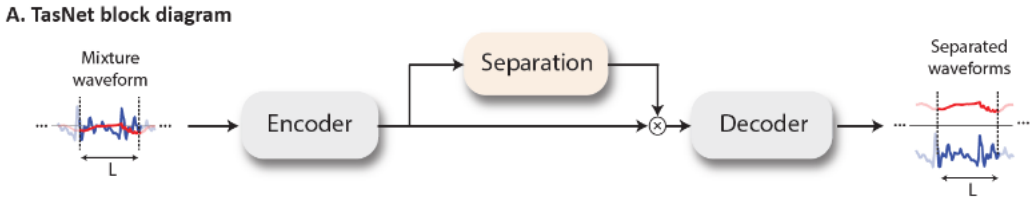
TasNet - Time–Domain Audio Separation Network

Tato kapitola vychází z referenční studie **TasNet: Surpassing Ideal Time-Frequency Masking for Speech Separation** [12] a popisuje její navrhnuté řešení pro separaci mluvčích v časové doméně.

Přestože metody pro zpracování řeči v takovém akustickém prostředí, ve kterém se současně prolíná mnoho řečových signálů, v poslední době zaznamenaly velké zlepšení, stále trpí mnoha nedostatky. Přesnost systémů, odezva a nároky na výpočetní výkon jsou u těchto metod natolik zásadní, že znemožňují nebo velmi omezují jejich nasazení mimo výzkumné prostředí, například v aplikacích, které by mohly zpracovávat řeč v reálném čase.

Většina dosavadních postupů byla založena na převodu směsi řečových signálů do časově–frekvenční (T–F) reprezentace (spektrogramu) pomocí short–time fourier transformation (STFT) [17]. Tento převod ale měl pro využití v reálném čase příliš vysokou odezvu a navíc T–F reprezentace nebyla optimální pro separaci mluvčích.

Pro překonání nedostatků předešlých metod byla navržena architektura fully–convolutional time–domain audio separation network (Conv–TasNet), založena na hlubokém učení a konvolučních neuronových sítích. Model prvně použije konvoluční enkodér k převodu krátkých segmentů směsi mluvčích na odpovídající nezápornou reprezentaci, která je optimalizovaná pro extrakci jednotlivých mluvčích. Samotné separace je docíleno aplikací masek na danou reprezentaci. Masky pro každého mluvčího pro každý segment v každém časovém kroku jsou odhadnuty v TCN, která je tvořena opakující se posloupností konvolučních bloků se zvyšující se časovou dilatací. Po aplikaci masek jsou separovaní mluvčí rekonstruováni lineárním dekodérem. Tuto posloupnost operací zobrazuje zjednodušený obrázek 4.1. Dále v kapitole budou jednotlivé části popsány detailněji.



Obrázek 4.1: Zjednodušený model architektury TasNet

Problém separace mluvčích na jednokanálové nahrávce lze definovat jako odhad C zdrojů mluvčích $s_1(t), \ldots, s_c(t) \in \mathbb{R}^{1\times T}$ na diskrétním signálu směsi $x(t) \in \mathbb{R}^{1\times T}$, kde T je délka nahrávky a kde

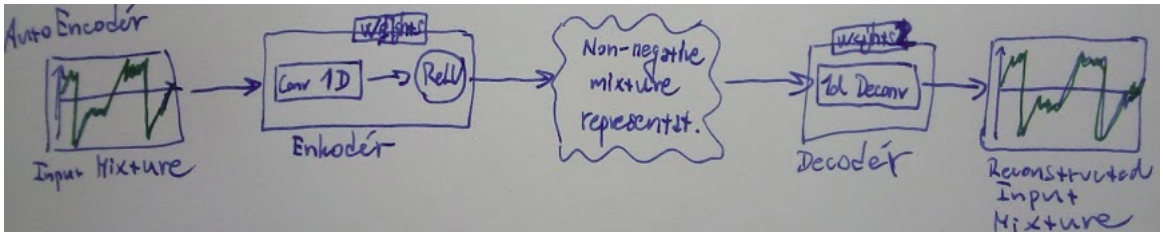
$$x(t) = \sum_{i=1}^{C} s_i(t) \tag{4.1}$$

Cílem je odhadnout $s_i(t), i = 1, \ldots, C$ ze signálu směsi $x(t)$.

[[sem mozna tabulka hyperparametrů a jejich pojmenovani X, R, N, L, P, ...]]

4.1 Konvoluční auto–enkodér

Konvoluční auto–enkodér převádí vstupní segmenty nahrávky na nezápornou reprezentaci a následně zase zpět z reprezentace na původní nahrávku, jak ukazuje obrázek 4.3.



Obrázek 4.2: Schéma konvolučního autoenkodéru

4.1.1 Enkódování směsi

[[spravit znak pro konvolucni operator v math env]] Každý segment směsi o délce L, $x_k \in \mathbb{R}^{1\times L}$, kde $k = 1, \ldots, \frac{T}{L}$, je transformován na nezápornou reprezentaci $w \in \mathbb{R}^{1\times N}$ pomocí 1–D konvoluční operace jako

$$w = ReLU(x \star U) \tag{4.2}$$

kde $U \in \mathbb{R}^{N\times L}$ obsahuje N vektorů, každý délky L, které reprezentují bázové funkce enkodéru. Operace $\circledast$ značí konvoluční operaci. ReLU je nelineární aktivační funkce, která byla blíže popsána v podkapitole 3.3.

4.1.2 Dekódování extrahovaných mluvčích

Pro převod z reprezentace zpět do podoby audio nahrávky slouží lineární dekodér. Pomocí 1–D dekonvoluční operace rekonstruuje původní signál x jako $x \in \mathbb{R}^{1\times L}$. Tuto operaci lze definovat jako

$$\hat{x} = wV \tag{4.3}$$

kde každý řádek v matici $V \in \mathbb{R}^{N\times L}$ představuje jednu bázovou funkci dekodéru s délkou L.

4.2 Separační modul

Cílem separačního modulu je najít váhovanou funkci, neboli masku, pro každého zdrojového mluvčího pro každý výstup enkodéru v každém časovém kroku. Formálně lze zapsat, že cílem

je odhadnout C masek $m_i \in \mathbb{R}^{1\times N}, i = 1, \ldots, C$, kde C představuje počet mluvčích ve směsi. Vektory masek m_i mají takové omezení, že $\sum_{i=0}^{C} m_i = 1$, kde 1 je jednotkový vektor v $\mathbb{R}^{1\times N}$. Toto omezení garantuje, že rekonstruované zdroje po sečtení zformují původní směs $\hat{x} = \sum_{i=1}^{C} \hat{s}_i$.

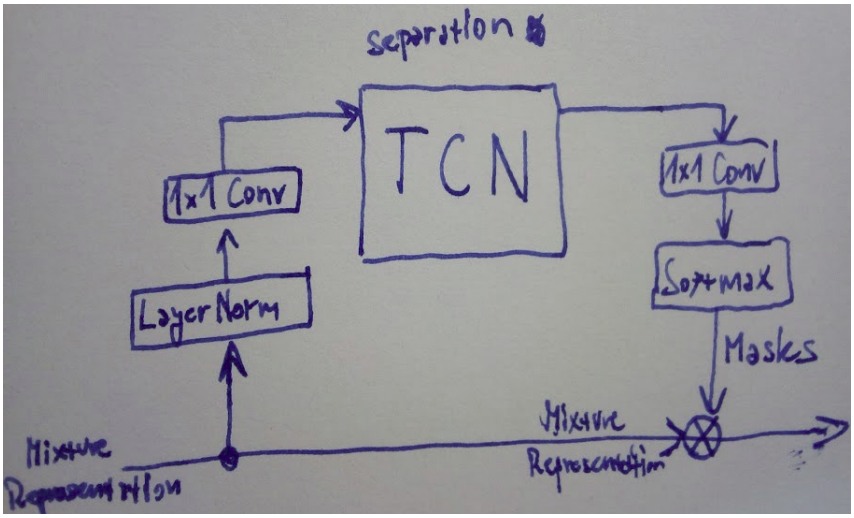
Separace je pro každý zdroj provedena vynásobením odpovídající masky m_i s nezápornou reprezentací (výstupem enkodéru) w, jako

$$d_i = w \odot m_i \tag{4.4}$$

kde $d_i \in \mathbb{R}^{1\times N}$ je reprezentace každého ze zdrojů a operace $\odot$ je [[not sure]] vektorový součin.

Nakonec jsou preprezentace d_i rekonstruovány zpět na korespondující nahrávky zdrojů $\hat{s}_i, i = 1, \ldots, C$ pomocí dekodéru jako

$$\hat{s}_i = d_i V \tag{4.5}$$



Obrázek 4.3: Schéma kompletního separačního modulu, jehož vstupem je nezáporná reprezentace směsi mluvčích

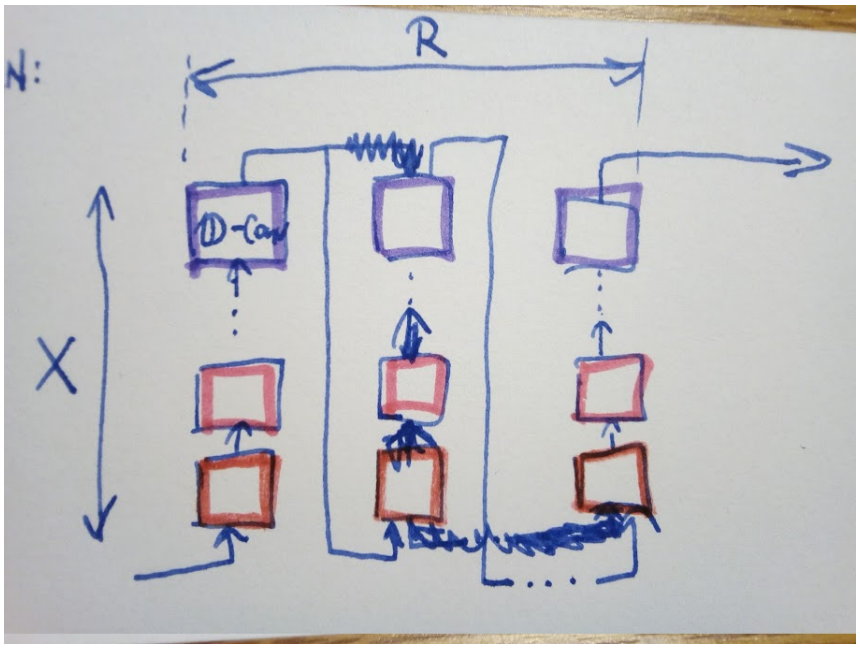
[[obrazky mozna nektere zcuknout k sobe nebo vedle sebe]]

Na začátku separačního modulu je přidána lineární operace 1×1–conv jako bottleneck, která určuje počet kanálů na vstupu a výstupu následující sekvence konvolučních bloků.

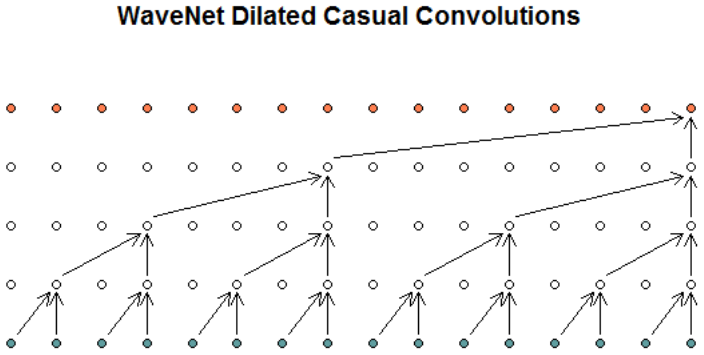
4.2.1 Temporal Convolutional Network

Jádrem separační části je *temporal convolutional network* skládající se z naskládaných 1–D konvolučních bloků s časovou dilatací jak lze vidět na obrázku 4.4. Každá vrstva v TCN obsahuje bloky se zvyšující se časovou dilatací, která se zvyšuje exponenciálně v závislosti na parametru X, který udává počet konvolučních bloků [[v jedné vrstvě]] a nabývá hodnot $1, 2, 4, \ldots, 2^{X-1}$. Taková sekvence bloků je opakována R–krát. Exponenciální růst dilatačního faktoru zajišťuje dostatečně velké okno pro využití výhod dlouhých časových závislostí v signálu řeči. Obrázek 4.5 zobrazuje rostoucí časovou dilataci na vstupních vzorcích signálu.

Výstup posledního bloku posledního opakování v TCN je předán 1×1 konvoluční vrstvě, která má $N \times C$ filtrů a nakonec aktivační funkci softmax k odhadu C vektorů masek pro každého z C cílových mluvčích.



Obrázek 4.4: Jádro separačního modulu – naskládané konvoluční bloky s časovou dilatací odhadující masky na základě nezáporné reprezentace[[**pridat dilation factor 2 mocnina X minus 1 do obrazku stacked–resblocks jpg**]]



Obrázek 4.5: Vizualizace časové dilatace

4.2.2 Konvoluční bloky - mozna ne jako podkapitolu

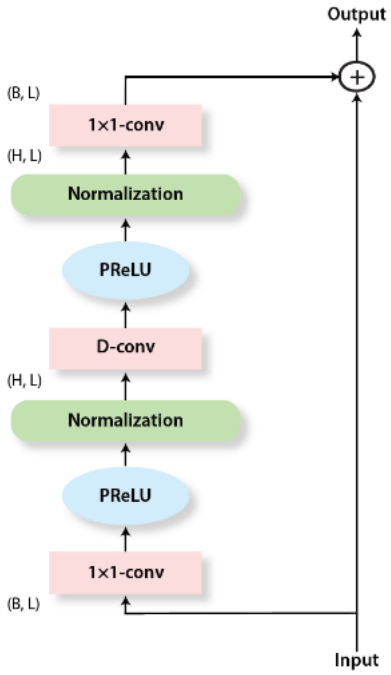
[[Pozn: vrstvy bloku a residualni spojeni]] Navržená architektura dále nahradila standardní konvoluci uvnitř 1×1 konvolučních bloků za *depthwise separable convolution* (S–conv), která pomáhá snížit počet parametrů a ukázala se jako efektivní ve zpracováni obrazu[[citace 26 27]].

Depthwise separable convolution se skládá ze dvou, po sobě jdoucích, operací – depthwise convolution (D–conv) a standardní konvolucí 1×1–conv s velikostí konvolučního jádra 1:

$$D - conv(Y, K) = concat(y_j convk_j), j = 1, \ldots, N \tag{4.6}$$

$$S - conv(Y, K, L) = D - conv(Y, K) convL \tag{4.7}$$

kde $Y \in \mathbb{R}^{G \times M}$ je vstup do S–conv, $K \in \mathbb{R}^{G \times P}$ je konvoluční jádro o velikosti P, dále $y_j \in \mathbb{R}^{1 \times M}$ a $k_j \in \mathbb{R}^{1 \times P}$ jsou řádky matic Y a K. $L \in \mathbb{R}^{G \times H \times 1}$ je konvoluční jádro o velikosti 1. Operace 1×1–conv se chová jako plně propojená vrstva a transformuje příznaky do potřebných rozměrů.



Obrázek 4.6: Jeden konvoluční blok obsahující reziduální spojení mezi vstupem a výstupem[[graficky zde ukazat a zakrouzkovat co je sconv a co je dconv]]

V každém konvolučním bloku jsou operace 1×1–conv a D–conv následovány nelineární aktivační funkcí PReLU [6], která byla popsána v kapitole 3, a normalizační vrstvou. Konvoluční bloky dále obsahují reziduální spojení [5] mezi vstupem a výstupem, které výrazně zjednodušuje trénování velmi hlubokých neuronových sítí díky ponechané referenci na vstupní netransformovaná data.

[[zvýraznit na obrázku konvolučního bloku residuální spojení a hodit tam sipku nebo neco ze to je to spojeni]]

[[Kam uvozovky, kam bold, kam italic ZEPTAT SE]]

4.3 Normalizace

Při trénování neuronových sítí může docházet k fenoménu jménem [[**uvozovky**]] vnitřní kovariantní posuv *internal covariate shift*, kvůli kterému je nutné pečlivě inicializovat parametry a volit menší hodnotu learning rate, což zpomaluje trénování sítě. K adresování problému slouží zakomponování normalizace do architektury neuronové sítě. Normalizace je aplikována na každý mini–batch trénovacích dat [8][1].

Konvoluční bloky obsahují normalizační vrstvy, které můžou významně ovlivnit výkon sítě. Ve studii bylo experimentováno s channel–wise layer normalization (cLN) [[**cite**]], global layer normalization (gLN) a batch normalization (BN) [8].

Metoda cLN je aplikována na vstup separačního modulu pro zajištění invariance při změně měřítka vstupních dat. Tato metoda je vhodná pro použití v kauzální i nekauzální konfiguraci, je–li použita v konvolučních blocích. Normalizace cLN je aplikována na každý segment y_k.

Metoda gLN je aplikována globálně na každý příznak na rozměry kanálu i času. Tuto normalizaci lze použít pouze při nekauzální konfiguraci, protože výpočet probíhá na základě celého vstupu. Při kauzální konfiguraci může být použita také metoda BN.

Normalizace BN je počítána následovně:

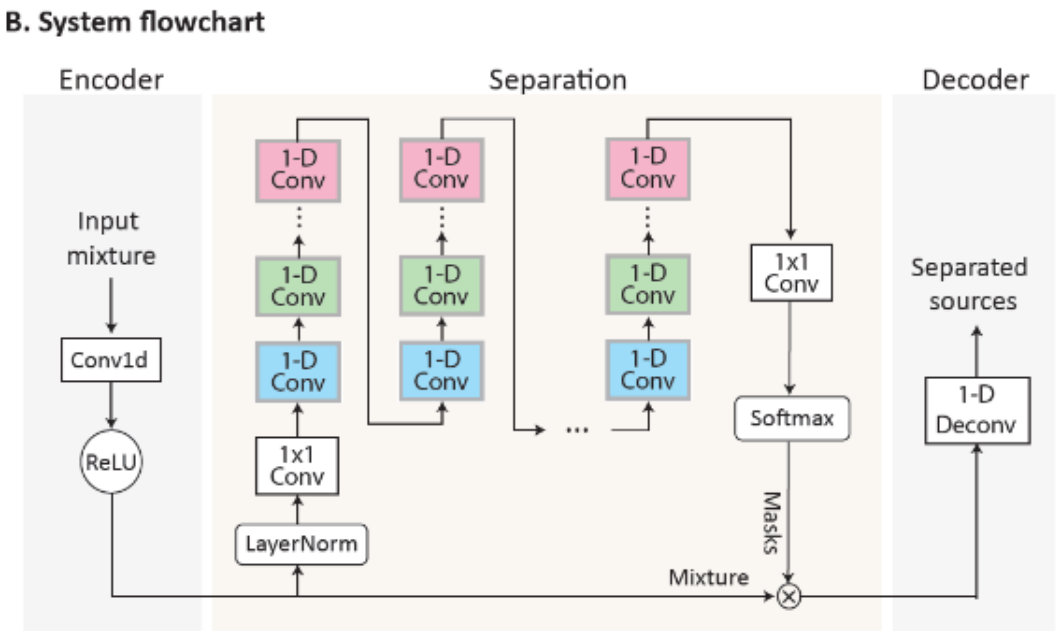
$$y = \frac{x - E[x]}{Var[x] + \varepsilon} * \gamma + \beta \tag{4.8}$$

$$E[x] = \frac{1}{NT} \sum_{NT} Y \tag{4.9}$$

$$Var(Y) = \frac{1}{NT} \sum_{NT} (Y - E[Y])^2 \tag{4.10}$$

[[**Asi jeste popsat jednotlive promenne ve vzorcich**]]

Jak vypadá kompletní schéma architektury po sestavení a propojení jednotlivých modulů lze vidět na obrázku 4.7.



Obrázek 4.7: Schéma architektury TasNet

V této kapitole byla popsána navržená architektura Conv–TasNet pro separaci navzájem se překrývajících mluvčích v časové doméně, skládající se z konvolučního auto–enkodéru, který převádí vstupní směs mluvčích na nezápornou reprezentaci, ze které je následně v separačním modulu složeném z konvolučních bloků se zvyšující se časovou dilatací odhadnuty multiplikativní funkce (tzv. masky) pro každý ze zdrojů. Ty jsou nakonec aplikovány na reprezentaci a výsledek lineárním dekodérem převeden zpět na hlasovou, nyní již separovanou nahrávku.

Kapitola 5

Implementace sítě

Pro implementaci neuronových sítí vzniklo mnoho frameworků jako PyTorch, Tensorflow, Keras a další, které umožňují síť poskládat z již předdefinovaných modulů. Jejich vlastnosti lze modifikovat dle potřeby pomocí argumentů při jejich instanciaci nebo při jejich použití. Kromě těchto bloků a mnoha dalších možností frameworky obsahují i metody pro práci s datasety, funkce pro vykreslování grafů a pro matematické operace.

Mým úkolem bylo implementovat neuronovou síť podle architektury TasNet [12] pro separaci mluvčích v časové doméně. V rámci zadání jsem si zvolil použít framework s názvem PyTorch[1], který má kvalitní dokumentaci a aktivní uživatelskou základnu. PyTorch funguje nad jazykem Python[2] a v celé práci používám jeho aktuálně nejnovější verzi (Python 3.8.2). Mimo jiné podporuje práci s datasety, včetně implementace vlastního nahrávání a manipulace s daty. Základní jednotkou, se kterou síť pracuje, je tensor. Ve frameworku PyTorch je **torch.Tensor** třída reprezentující multidimenzionální matici obsahující data jednoho typu. Na tensory jsou převáděna data během načítání datasetu a následně předávána síti ke zpracování. Samotná implementace probíhala na systému Kali linux (4.19.0-kali5-amd64), který je založený na systému Debian. Aplikace byla vyvíjena v textovém editoru Vim[3] a byla během implementace spouštěna a testována v příkazovém procesoru Bash[4] (Bourne Again SHell) verze 5.0.16. Bash jsem zvolil i pro implementaci některých pomocných skriptů kvůli jednoduchému spouštění z terminálu a možnostem automatizace některých opakujících se příkazů. Pro správu verzí jsem používal verzovací systém Git[5].

Trénování sítě probíhalo z počátku na CPU *intel i5* mého osobního notebooku *Lenovo Y50-70*, což se ale ukázalo jako naprosto nevhodné vzhledem k náročnosti výpočtu kvůli nedostatečnému výkonu. Začal jsem tedy používat službu *Google colab*, která poskytuje na omezenou dobu (12 hodin) stroje, které navíc obsahují technologii *cuda*, což mi umožňilo výpočet provádět na GPU, čímž se trénování zrychlilo několikanásobně. Ještě více pomohlo předplacení služby, čímž se mi prodloužila doba, po kterou jsem mohl síť trénovat, na 24 hodin a zvýšila se šance na získání silnějších strojů. Modely byly trénovány na GPU *T80* a *T100* v závislosti na tom, která z nich mi zrovna byla přidělena pro dané sezení (*session*).

[1] https://pytorch.org/
[2] https://www.python.org/
[3] https://www.vim.org/
[4] https://www.gnu.org/software/bash/
[5] https://git-scm.com/

[[zamyslet se co tu chci ukazat v tom algu, pripadne vypustit parsovani, instanciaci atd]]

parsování argumentů programu;
instanciace modelu TasNet;
if *args.checkpoint*! = *NULL* **then**
    načtení checkpointu;
end
instanciace třídy SegmentDataset a dataloaderu pro trénovací data;
instanciace třídy SegmentDataset a dataloaderu pro validační data;
for *epocha v args.Epochs* **do**
    `/* trénování */`
1   loss = 0;
2   **for** *směs_mluvčích z trénovacího datasetu* **do**
        `/* cyklus načítá z dataloaderu segmenty nahrávek, které jsou generovány v třídě SegmentDataset */`
3       $odhadovan\_separace = TasNet(sms\_mluvch)$;
4       $loss = sisnr(optimln\_separace, odhadovan\_separace)$;
5   propagace chyby a úprava vah;
6   **end**
    `/* validace */`
7   vypnout gradient;
8   **for** *směs_mluvčích z validačního datasetu* **do**
9       $odhadovan\_separace = TasNet(sms\_mluvch)$;
10      $validacni\_loss = sisnr(optimln\_separace, odhadovan\_separace)$;
11   **if** *hodnota validační loss neklesla potřetí v řadě* **then**
12          $learning\_rate = learning\_rate/2$;
13   **end**
14 **end**
15 na konci epochy uložit checkpoint;
end

Algoritmus 1: Zjednodušený algoritmus běhu programu pro trénování sítě

5.1 Implementace tříd a modelu

Model neuronové sítě byl logicky rozdělen na několik dílčích částí, z nichž každá je reprezentována jednou třídou. Jednotlivé třídy obsahují metody pro nahrávání a transformaci dat, segmentaci nahrávek a další jsou přímo stavebním blokem neuronové sítě. Některé parametry předané třídám při jejich instanciaci jsou hyperparametry, jejichž hodnoty byly určeny parametry z příkazové řádky při volání programu. Každá z tříd bude popsána dále v textu.

Třídy SegmentDataset a AudioDataset

Třídy **SegmentDataset** a **AudioDataset** slouží pro načítání korespondujících trojic nahrávek směs–zdroj1–zdroj2 z adresáře, kde je uložen dataset. Nahrávky jsou ve formátu wav a vzorkovány na 8KHz. Třídy tato data dále transformují na tenzory a normalizují na jednotkovou varianci a zero–mean.

Tyto třídy používají některé pomocné funkce definované v souboru **tools.py** pro dorovnání nedostatečně velkých nahrávek ve zpracovávaném mini–batchi nulami na velikost nejdelší z nich, protože data obsažená v mini–batchi musejí mít stejnou velikost.

Při spuštění programu jsou nahrány názvy wav souborů do polí, které jsou poté buď náhodně či sekvenčně procházeny a postupně dochází k načítání trojic nahrávek na aktuálním indexu.

Obě třídy jsou periodicky volány instancí třídy Dataloader[6], která v cyklu posílá požadavky na data ke zpracování. Rozdíl mezi třídami je ten, že třída AudioDataset předává trojice nahrávek v původním stavu a délka epochy tak trvá tolik iterací, kolik je nahrávek ve zpracovávaném datasetu a je kvůli tomu používána během testování sítě, kde chceme vyhodnocovat celé nahrávky. Třída SegmentDataset je použita pro trénovací a validační dataset, kde jsou nahrávky rozdělovány na segmenty o parametrem dané délce a až poté jsou poskytnuty ven. Segmentace je detailně popsána v podkapitole 5.2.

Po vyčerpání datasetu obě třídy vyvolají vyjímku **StopIteration**, která je detekována Dataloaderem a ten ukončí cyklus, ve kterém se dataset zpracovával. Tím v případě trénování končí učení sítě a začíná validace, v případě validace končí jedna epocha a v případě testování končí vyhodnocování sítě.

Třída Tasnet

Třída Tasnet.py reprezentuje model neuronové sítě. Atributy třídy představují jednotlivé vrstvy neuronové sítě. Síť obsahuje konvoluční vrstvy, enkodér, dekodér a *temporal convolutionan network*, která je tvořena sekvencí konvolučních bloků, které jsou rozebrány později. Konvolučním vrstvám jsou nastaveny hodnoty *padding* a *stride*, které ovlivňují chování konvolučních vrstev. V této třídě jsou také inicializovány váhy (filtry) konvolučních vrstev algoritmem Xavier [4]. Ve funkci $forward()$, která je volána, když jsou instanci sítě předána data ke zpracování, jsou sekvenčně volány jednotlivé vrstvy. Během zpracování předaných dat je v této sekvenci vypočítána maska a aplikována na zpracovávanou směs. Tím vzniknou dvě separované nahrávky, které jsou předány na výstup v podobě tensoru.

Třída ResBlock

Třída ResBlock reprezentuje jeden konvoluční blok, který je opakován se zvyšující se dilatací v *temporal convolutional network* v separační části určené k odhadu masek. Podobně jako třída Tasnet, i tato třída obsahuje inicializaci a zřetězení vrstev. V inicializaci jsou nastaveny hodnoty pro konvoluční operace jako počet konvolučních jader, časová dilatace a počet kanálů. Tyto operace jsou ve funkci *forward* zřetězeny do sekvence konvolučních operací a aktivačních funkcí, podle vzorové architektury TasNet. Třída ResBlock je specifická svým residuálním spojením, kde dochází k rozdělení dat při zpracování, kdy jedna kopie je transformována operacemi v bloku a na konci sečtena s daty, které transformacemi neprošly a tento součet je výstupem bloku. Residuální spojení jsou podrobněji popsána v kapitole 3.

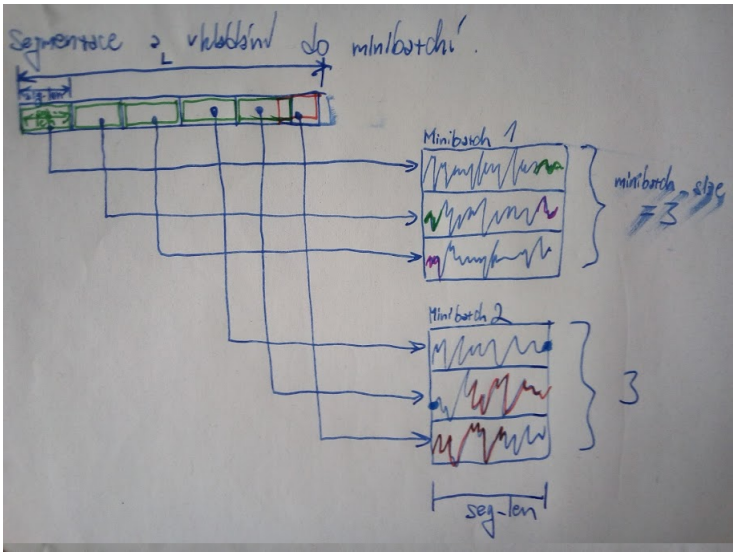
5.2 Segmentace nahrávek

Pro účely trénování sítě byly vstupní nahrávky rozdělovány na segmenty o délce 4, případně 2 sekund. Délku segmentů L_s lze nastavit přes argument `--segment_length` při volání skriptu *train.py*. Tato hodnota reprezentuje jeden z hyper–parametrů sítě. Výchozí hodnota

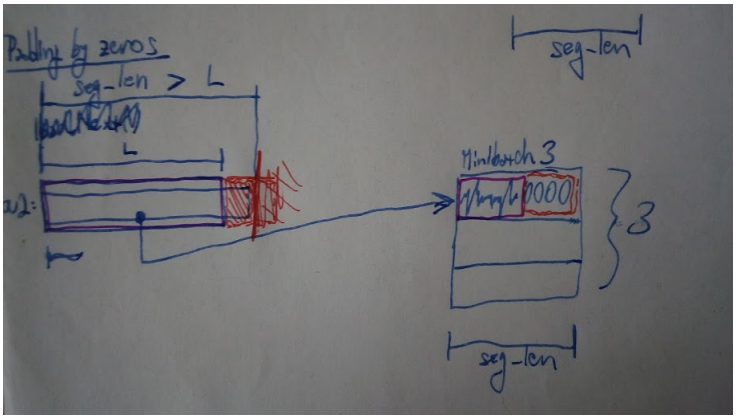
[6] https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader

je 4 sekundy. Při nižších hodnotách se prodlužuje délka trénování, protože se z jedné vstupní nahrávky o délce L vygeneruje až L/L_s segmentů a tím narůstá počet dat, který síť musí zpracovat.

Poslední segment by byl kratší, pokud by délka nahrávky byla kratší než délka segmentu. V takovém případě je segment doplněn nulami do délky ostatních segmentů jak lze vidět na obrázku 5.2. Toto doplnění je nutné kvůli dávkovému zpracování (**batches**), které neumožňuje rozdílnou délku elementů v dávce. Pokud je délka nahrávky delší než délka segmentu, tak se poslední segment vezme od konce bez nutnosti ho pak nulami doplňovat. Skládání segmentů do minibatche zobrazuje obrázek 5.1. Při vzorkovací frekvenci $f_s = 8000Hz$, obsahují segmenty $f_s * L_s$ vzorků vstupní nahrávky.



Obrázek 5.1: Dělení nahrávky na segmenty o délce **segment_len** a jejich následné vkládání do minibatche. Lze si všimnout, že pokud poslední segment není dostatečně dlouhý, tak se vezme od konce nahrávky a jeho počáteční vzorky budou duplicitní s posledními vzorky předešlého segmentu

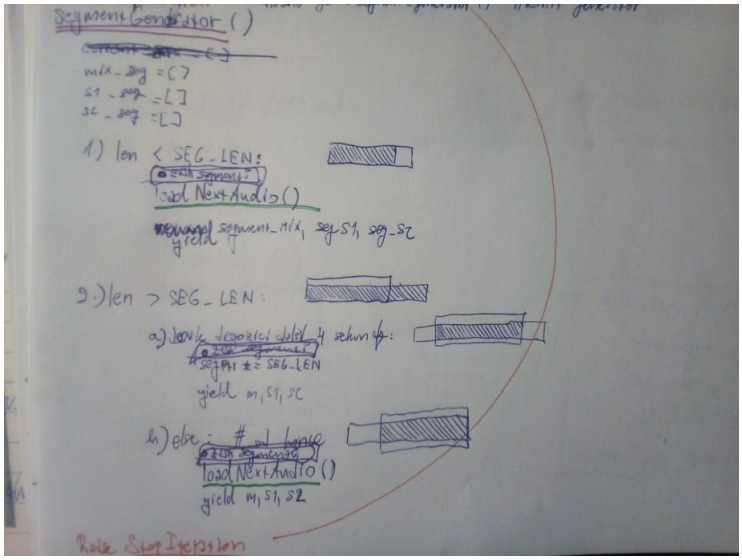


Obrázek 5.2: Je–li zpracovávána nahrávka, jejíž délka je kratší než je délka segmentu, je tento segment doplněn z prava nulami do požadované délky

[[mozna tento odstavec jeste nejak upravit a doplnit obrazek kde je nacitani asi 3 audio nahravek]] Algoritmus 2 a obrázek 5.3 zobrazují proces segmentace. Funkce `segmentGenerator()`, která slouží jako generátor, pracuje současně se třemi nahrávkami – se směsí dvou mluvčích, s nahrávkou prvního mluvčího a s nahrávkou druhého mluvčího, které jsou do atributů `current_mixture`, `current_source1` a `current_source2` třídy `SegmentDataset` nahrávány funkcí `loadNextAudio()` poté, co z předchozí trojice nahrávek již nelze vygenerovat další segmenty.

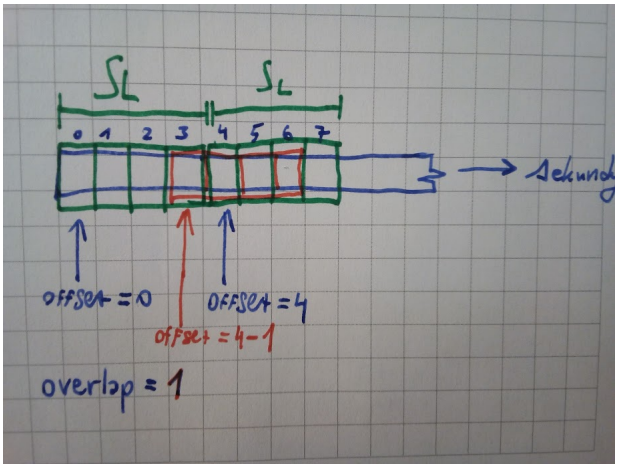
```
Function segmentGenerator() : int[] is
    s1_segment = [];
    segptr = 0;
    while je dostupná další trojice nahrávek do
        if current_mixture_len < SEGMENT_LENGTH then
            /* aktuální nahrávka je kratší než délka segmentu          */
1           s1_segment = current_source1[:];
2           yield s1_segment;
        else
            if segptr + SEGMENT_LENGTH < current_mixture_len then
                /* z aktuální nahrávky vzít segment                    */
3               s1_segment = current_source1[(segptr+SEGMENT_LENGTH)];
4               segptr+ = SEGMENT_LENGTH;
5               yield s1_segment;
            else
                /* nelze již načíst celý segment, takže se vezme od konce
                   */
6               start_index = (current_mixture_len - SEGMENT_LENGTH);
7               s1_segment =
                 current_source1[start_index:current_mixture_len];
8               yield s1_segment;
            end
        end
    end
end
```

Algoritmus 2: Algoritmus segmenace nahrávek používá výhody python generátoru, který, narozdíl od běžných funkcí, po vrácení hodnoty příkazem `yield` neztrácí svůj vnitřní stav a při jeho dalším zavolání pokračuje tam, kde skončil. V uvedeném příkladě je znázorněna pouze segmentace nahrávky jednoho mluvčího, ale analogicky se segmentují i nahrávky směsi mluvčích a nahrávky druhého mluvčího



Obrázek 5.3: Způsob segmentace nahrávek

Kód segmentace také umožňuje určit délku, o kterou se segmenty budou překrývat. Této funkcionality je docíleno zavedením proměnné `overlap`, která obsahuje délku překrytí a při segmentaci je od proměnné `offset`, který určuje začátek následujícího segmentu, tato hodnota odečtena, jak ukazuje obrázek 5.4. Při trénování je překrytí nulové, protože by to zvýšilo počet trénovacích dat jejich částečnou duplikací, což by negativně ovlivnilo délku trénování.



Obrázek 5.4: Segmentace nahrávek s nenulovým překrytím

5.3 Pomocné skripty

Během implementace se vyskytla potřeba zjednodušit si některé repetetivní úkony a jelikož upřednostňuji práci z terminálu, tak jsem si pro ně naprogramoval několik skriptů za použití jazyků python3 a bash.

Skript ***gupload.sh*** pro odesílání zdrojových souborů na *Google Drive*[7], ze kterého *Google Colab*[8] načítá data a kód, který lze následně po částech spouštět z grafického rozhraní. K interakci s Google Drive jsem použil nástroj *rclone*[9], který nabízí mnoho operací pro vzdálenou práci s Google Drive z terminálu. S pomocí *rclone* lze stahovat či nahrávat data, vzdáleně měnit adresářovou strukturu, mazat, vytvářet a upravovat soubory a podobně.

Skript ***gdownload.sh*** je určený pro snadné stahování dat z Google Drive. Stejně jako předchozí skript, používá operace poskytované nástrojem rsync. Při zavolání stáhne adresář obsahující data z posledního trénování, nebo adresář, jehož název je předán jako argument skriptu v přikazové řádce. Stažený adresář obsahuje checkpointy, soubory obsahující výstup trénování, soubor obsahující dvojice hodnot $[zpracovno_nahrvek, loss]$, které lze jednoduše vykreslit do grafu, a nakonec soubor s výsledky testování obsahující hodnoty metrik pro testovaný checkpoint.

Dále jsem vytvořil několik jednoduchých skriptů pro rychlé spuštění trénování a testování modelu s různými přednastavenými parametry. Bylo tak možno mít mezi skripty rozdílné cesty k adresářům pro účely trénování a ladění chyb nebo třeba nastavení času, po kterém se proces má sám ukončit. Pro spuštění trénování modelu tak lze zavolat skript ***nntrain.sh***, pro testování skript ***nntest.sh*** a pro inferenci (separaci mluvčích na předané nahrávce) skript ***nninference.sh***.

Implementace samotného modelu ve frameworku PyTorch nebyla náročná. Vzhledem k netriviálnímu úkolu jsem musel implementovat některé části, jako třeba segmentace nahrávek, sám. To přineslo mnoho potíží při odstraňování chyb v samotném modelu. Síť jsem prvně trénoval a ladil na CPU svého notebooku, takže probíhalo velmi pomalu (v řádu desítek hodin i při nízké velikosti modelu), a některé chyby se objevily až v průběhu trénování nebo dokonce na jeho konci. Situaci razantně zlepšil přechod na Google Colab, kde jsem síť trénoval na poskytovaných GPU a trénování tak zabralo **[["pouze"(opravit uvozovky)]]** několik hodin, takže se případná chyba objevila dříve, ale i přesto mi lazení sítě zabralo mnoho času. Pro další usnadnění samotné implementace jsem vytvořil několik skriptů, které mi usnadnily spouštění a lazení sítě.

[7] https://www.google.com/drive/
[8] https://colab.research.google.com/notebooks/intro.ipynb
[9] https://rclone.org/

Kapitola 6

Experimenty a vyhodnocení

- trenovani s ruznymi hyperparametry, uspesnost a tabulky s hyper parametry a dosazenymi vysledky a hodnotami sisnr, sdr atd. - model size comparison. - porovnani s vysledky ze studie - obrazky separovanych mluvcich - signalu. - spektra - grafy trenovani loss a vysledkuu. - pametova narocnost modelu

Neuronové sítě představují mocný nástroj mnoha využití, který ale musí být naučen, jak danou činost provádět. Trénování sítí je výpočetně náročný úkol, jehož náročnost se, krom dalších parametrů, může lišit v závislosti na velikosti datasetu a sítě. Obecně čím větší dataset a rozměr sítě, tím déle trénování trvá. Je proto podstatné, na jakém stroji je síť učena a od toho se odvíjí čas trénování.

Po dostatečném natrénování probíhá vyhodnocení sítě. Výsledky trénování jsou měřeny metrikami, které se volí na základě problému, který se síť učila řešit. Výsledky testování sítě jsou následně porovnávány v závislosti na velikosti daného modelu. Velikost modelu je ovlivněna hyper–parametry, které určují například počet vrstev sítě, velikost segmentace a další vlastnosti.

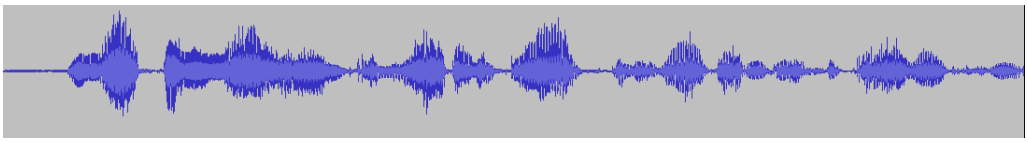
Cílem experimentů je zjistit, které z hyper–parametrů mají největší vliv na kvalitu separace a tedy na výsledky vyhodnocení, a jak moc lze model zmenšit, aby stále dával dostatečně dobré výsledky separace mluvčích. Tato kvalita je měřena primárně metrikami STOI, SI–SNR a PESQ, které byly popsány v kapitole 2. **[[ref na zdroje u metrik]]**

6.1 Dataset

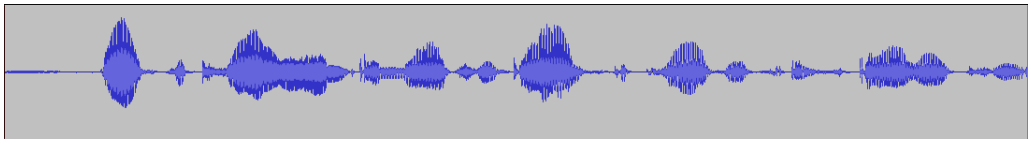
Dataset je množina dat, na kterých je síť učena a testována. Dataset se dá rozdělit na 3 podmnožiny, které jsou navzájem exkluzivní. Trénovací dataset je určen k trénování sítě a bývá největší. Validační dataset se prochází po zpracování trénovacího datasetu v průběhu trénování a slouží pro ověření, že hodnota loss na dosud neviděných datech stále klesá a nedochází tak k přetrénování. Při zpracování tohoto datasetu je síti zabráněno v učení. Poslední je testovací dataset, na kterém probíhá vyhodnocení sítě, tedy výpočet hodnot jednotlivých metrik.

Trénování a vyhodnocení modelu proběhlo na množině jednokanálových nahrávek směsí dvou mluvčích. Množina byla vygenerována náhodným výběrem různých mluvčích z Wall Street Journal (WSJ0) a vytvořením směsi mluvčích z těchto dvou náhodně vybraných nahrávek. Celková délka trénovacích dat je přes 10 hodin a přes 6 hodin validačních dat. Nahrávky jsou převzorkovány na 8kHz a během trénování zarovnány na zero means a jednot-

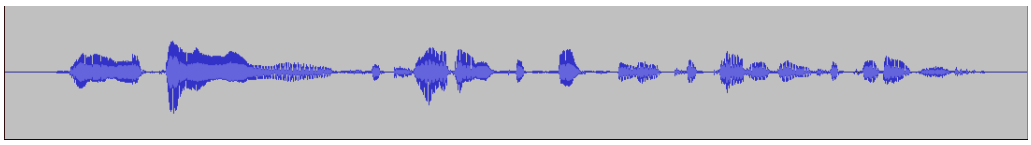
kovou varianci [[[**studie str 5 Dataset**][**49 - ze studie odkaz na script na generovani a popis na netu**]]].



Obrázek 6.1: Ukázka nahrávky směsi dvou mluvčích



Obrázek 6.2: První mluvčí ze směsi



Obrázek 6.3: Druhý mluvčí ze směsi

Lze si všimnout, že sečtením signálů separovaných mluvčích na obrázku 6.2 a 6.3 dostaneme přesně signál směsi, což lze vyjádřit vztahem

$$x[t] = \sum_{i=1}^{C} s_i[t] \tag{6.1}$$

, kde $x[t] \in \mathbb{R}^{1\times T}$ je diskrétní signál směsi a $s_i[t] \in \mathbb{R}^{1\times T}$, kde $i = 1, \ldots, C$, je jeden z C zdrojů[cite studie str3 vlevo].

[[**doplnit info o zero means a jendotkove varianci**]]

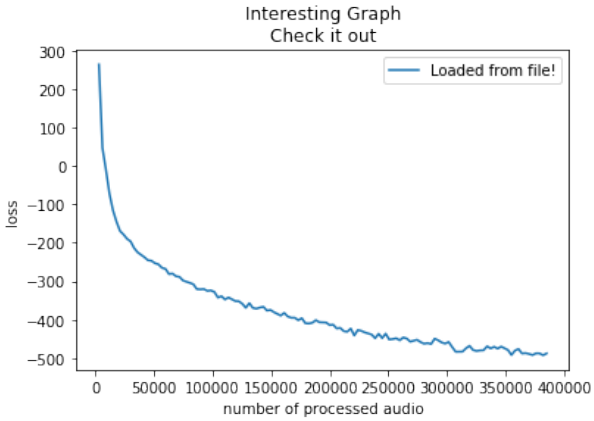
6.2 Průběh trénování

[[**miniÚvod - konkretni hyperparams, vykon a cas trenovani, seg–len, bylo natrenovano nekolik modelu ... konkretni hyperparams, vykon a cas trenovani, seg–len**]]. Bylo natrénováno několik, různě velkých modelů, které se lišily velikostí jejich hyper–parametrů, konkrétně počtem konvolučních bloků, počtem jejich opakování a velikostí segmentů zpracovávaných nahrávek z datasetu.

[[**https://www.tablesgenerator.com/**]].

Tabulka 6.1: Hodnoty hyperparametrů trénovaných sítí a jejich přesnost

	X	R	segment length sisnr
2	2	4	8.3
3	2	4	9.45
4	3	4	10.2
8	4	2	13.6



Obrázek 6.4: Příklad grafu loss hodnoty během učení

Obrázek 6.5: Hodnota loss při trénování modelů s různou velikostí hyperparametrů

6.3 Experimenty s modely

[[rozdelit tabulku podle HP a spis mensi tabulku ukazujici vysledku pro konkretni hyper parametry. experimenty: sisnr, reportovat ruzne metriky - stoi, pesq, prozkoumat hyper parametry a jak ktere jsou dulezite. Kdyz budu zmensovat sit tak jak se bude zhorsovat.]]

[[miniÚvod]] Pro vyhodnocení úspěšnosti a kvality modelů byly použity metriky si–snr vyjadřující poměr XXX, pesq říkající YYY a stoi vyjadřující ZZZ **[[mozna ref]]**.

6.3.1 Výsledky referenčního modelu TasNet

[[sem nebo do TasNet kapitoly? Jen lehce bych to shrnul ceho dosahli v tech vecech ktere tu merim a vyhodnocuji ja]]

6.3.2 Vliv hyper–parametru X

[[zmensovani a zvetsovani X, vysledky testovani, loss pripadne nejake fajnove grafy porovnani vysledky mezi jednotlivymi X, obrazky, grafy]]

6.3.3 Vliv hyper–parametru R

[[zmensovani a zvetsovani Y, vysledky testovani, loss pripadne nejake fajnove grafy porovnani vysledky mezi jednotlivymi Y, obrazky, grafy]]

6.3.4 Vliv hyper–parametru délka–segmentu

[[zmensovani a zvetsovani SL, vysledky testovani, loss pripadne nejake fajnove grafy porovnani vysledky mezi jednotlivymi SL, obrazky, grafy]]

6.3.5 Vyhodnocení a porovnání modelů

[[shrnuti a porovnani jednotlivych zmen mezi sebou komplexne. Pripadne porovnani vysledku s ref studii, obrazky, grafy]]

6.4 Možná rozšíření a navrhnutá vylepšení

- variabilnější dataset, mikrofony, šum a bordel prostředí - separace více mluvčích - hlučné prostředí - identifikace konkrétního řečníka - realtime separace

Kapitola 7

Závěr

[[co jak dopadlo, vysledky a vyhodnoceni velikosti modelu a jaky byl nejlepsi]]

Cílem práce bylo implementovat síť podle architektury TasNet pro separaci mluvčích v časové doméně a porovnat vliv velikosti sítě na kvalitu separace. Síť byla implementována za pomoci frameworku PyTorch a jazyku python a natrénována na datasetu obsahujícím jednokanálové směsi dvou mluvčích. Trénování proběhlo na **[[X]]** modelech, které se od sebe lišily počtem opakujících se konvolučních bloků, velikostí časové dilatace a délkou vstupních segmentů směsí. Pro účel vyhodnocení modelů byla použita metrika si-snr, která udává poměr chtěného signálu ku šumu na pozadí, tedy obecně kvalitu separace.

Experimenty ukázaly, že během testování nejlépe dopadla síť, která měla 8 konvolučních bloků po 4 opakováních, s délkou vstupního segmentu $L = 2$ sekundy. Tento model dosáhl po 100 epochách trénování hodnoty až **[[13,4]]** a tím se stal nejúspěšnějším modelem. Při fyzickém poslechu separovaných nahrávek bychom neslyšeli téměř žádný náznak druhého mluvčího. Oproti tomu, nejméně přesný model měl pouze 4 konvoluční bloky, 2 opakování a při délce segmentů $L = 4$ sekundy dosahoval hodnoty SDR pouze **[[9.2]]**.

Zkoušel jsem separovat také nahrávky, které byly úplně mimo dataset, ale výsledek se nedá hodnotit jako úspěšný, jelikož hraje velkou roli prostředí, mikrofon, šum v pozadí a další vlivy, na které byla neuronová síť naučena. Tento problém by se dal překonat rozšířením trénovacího datasetu o větší škálu nahrávek mluvčích, které by byly pořízeny z různých zařízení v různě rušném prostředí.

[[Doplnit ještě neco eh]] Mozna.

Literatura

[1] Ba, J. L., Kiros, J. R. a Hinton, G. E. *Layer Normalization*. 2016.

[2] Bengio, Y., Simard, P. a Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*. 1994, sv. 5, č. 2, s. 157–166.

[3] Chen, S. a Cheng, M.-J. Building an Adaptive Evaluation System: A Design Education Application. *Computer-Aided Design & Applications*. Leden 2006, sv. 3, s. 49–58. DOI: 10.1080/16864360.2006.10738441.

[4] Glorot, X. a Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In: Teh, Y. W. a Titterington, M., ed. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, sv. 9, s. 249–256. Proceedings of Machine Learning Research. Dostupné z: http://proceedings.mlr.press/v9/glorot10a.html.

[5] He, K., Zhang, X., Ren, S. a Sun, J. *Deep Residual Learning for Image Recognition*. 2015.

[6] He, K., Zhang, X., Ren, S. a Sun, J. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. 2015.

[7] Ian Goodfellow, A. C. *DEEP LEARNING*. MIT Press, 2017. ISBN 9780262035613.

[8] Ioffe, S. a Szegedy, C. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015.

[9] Kelleher, J. D. *DEEP LEARNING / John D. Kelleher*. MIT Press, 2019. ISBN 9780262537551.

[10] Kim, J., El-Khamy, M. a Lee, J. End-to-End Multi-Task Denoising for joint SDR and PESQ Optimization. *CoRR*. 2019, abs/1901.09146. Dostupné z: http://arxiv.org/abs/1901.09146.

[11] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E. et al. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*. 1989, sv. 1, č. 4, s. 541–551.

[12] Luo, Y. a Mesgarani, N. *Conv-TasNet: Surpassing Ideal Time-Frequency Magnitude Masking for Speech Separation*. 2018.

[13] Pekel, E. a Kara, S. A COMPREHENSIVE REVIEW FOR ARTIFICAL NEURAL NETWORK APPLICATION TO PUBLIC TRANSPORTATION. *Sigma Journal of Engineering and Natural Sciences*. Březen 2017, sv. 35, s. 157–179.

[14] Qian, W. C. C. X. e. a. Past review, current progress, and challenges ahead on the cocktail party problem. *Frontiers Inf Technol Electronic Eng 19*. Únor 2019, s. 40–63. Dostupné z: https://doi-org.ezproxy.lib.vutbr.cz/10.1631/FITEE.1700814.

[15] Rix, B. J. G. H. M. P. a Hekstra, A. P. Erceptual evaluation of speech quality (pesq)-anew method for speech quality assessment of telephonenetworks and codecs. *Acoustics, Speech, and SignalProcessing*. 2001, sv. 2, s. 40–63. Rix, A. W., Beerends, J. G., Hollier, M. P., and Hekstra,A. P. Perceptual evaluation of speech quality (pesq)-anew method for speech quality assessment of telephonenetworks and codecs. InAcoustics, Speech, and SignalProcessing, 2001. Proceedings.(ICASSP'01). 2001 IEEEInternational Conference on, volume 2, pp. 749–752.IEEE, 2001.

[16] Ruder, S. *An overview of gradient descent optimization algorithms*. 2016.

[17] Wang, D. a Chen, J. Supervised Speech Separation Based on Deep Learning: An Overview. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.* IEEE Press. říjen 2018, sv. 26, č. 10, s. 1702–1726. DOI: 10.1109/TASLP.2018.2842159. ISSN 2329-9290. Dostupné z: https://doi.org/10.1109/TASLP.2018.2842159.

[18] Zaeemzadeh, A., Rahnavard, N. a Shah, M. *Norm-Preservation: Why Residual Networks Can Become Extremely Deep?* 2018.