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

ADAPTACE NEURONOVÝCH SÍTÍ NA CÍLOVÉHO PISATELE

ADAPTATION OF NEURAL NETWORKS TO TARGET WRITER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAKUB SEKULA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JAN KOHÚT**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Sekula Jakub**
Program: Informační technologie
Název: **Adaptace neuronových sítí na cílového pisatele**
Adaptation of Neural Networks to Target Writer
Kategorie: Zpracování obrazu

Zadání:

1. Prostudujte základy neuronových sítí pro rozpoznávání textu.
2. Vytvořte si přehled o metodách, které umožňují adaptaci neuronových sítí pro rozpoznávání textu s učitelem a bez učitele.
3. Vyberte nejvhodnější metody a navrhněte metody vlastní.
4. Navrhněte experimenty.
5. Obstarejte si datovou sadu.
6. Implementujte metody a proveďte experimenty nad datovou sadou.
7. Porovnejte dosažené výsledky a diskutujte možnosti budoucího vývoje.
8. Vytvořte stručné video prezentující vaši práci, její cíle a výsledky.

Literatura:

- Meng, Zhong, Jinyu Li, and Yifan Gong. "Adversarial speaker adaptation." *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019.
- Kang, Lei, et al. "Unsupervised Adaptation for Synthetic-to-Real Handwritten Word Recognition." *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2020.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

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

Vedoucí práce: **Kohút Jan, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 30. října 2020

Abstrakt

Tato bakalářská práce se zabývá adaptací neuronových sítí na pisatele s cílem zlepšení rozpoznání ručně psaného písma tohoto pisatele. Metoda, kterou používám, je rychlá, vyžaduje malou trénovací množinu dat a využívá regularizaci, která se snaží udržet distribuci regularizovaných vah adaptační sítě podobnou té z předadaptační sítě. Tuto metodu jsem testoval nad datasetem tištěných textů IMPACT a datasetem ručně psaných textů. Nad datasetem ručně psaných textů se mi na dvou denících podařilo snížit chybovost z počátečních 10,82 % a 1,82 % na chybovost 8,48 % a 0,77 % v rámci malého počtu adaptačních iterací a při použití malého množství trénovacích řádků. Na datasetu IMPACT se mi podařilo snížit chybovost nad polské historické písmo z počáteční chybovosti 32,88 % na 5,30 %.

Abstract

This bachelor's thesis deals with the adaptation of neural networks to a specific writer with an aim to improve recognition of handwritten text of this specific writer. The method that I use is fast, requires small training dataset and uses regularization, which tries to keep the distribution of regularized weights in adaptation network similar to the one in the original network. I tested this method on dataset of printed text called IMPACT and dataset of handwritten text. When testing on dataset of handwritten text I was able to improve recognition on two diaries with preadaptation recognition error rate of 10,82 % and 1,82 % to 8,48 % and 0,77 % with a small number of adaptation iterations and using small amount of training lines. When testing on IMPACT dataset I was able to improve recognition error rate from 32,88 % to 5,30 %.

Klíčová slova

adaptace na pisatele, konvoluční neuronové sítě, rekurentní neuronové sítě, rozpoznávání textu

Keywords

writer adaptation, convolutional neural network, recurrent neural network, text recognition

Citace

SEKULA, Jakub. *Adaptace neuronových sítí na cílového pisatele.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jan Kohút

Adaptace neuronových sítí na cílového pisatele

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jana Kohúta. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Jakub Sekula
10. května 2021

Poděkování

Děkuji svému vedoucímu Ing. Janu Kohútovi za veškerou odbornou pomoc a ochotu při řešení mojí práce. Také děkuji za podporu celé mé rodině v průběhu studia. Výpočetní prostředky mi byly poskytnuty za podpory projektu "e-Infrastruktura CZ" (e-INFRA LM2018140) poskytovaného v rámci programu "Projekty velkých výzkumů, Vývoje a inovací infrastruktur".

Obsah

Kapitola 1

Úvod

Neuronovým sítím, které se věnují problematice rozpoznávání textu, se říká OCR (zkratka z angl. Optical character recognition). Častým problémem je, že se natrénuje neuronová síť pro rozpoznávání ručně psaného textu, ale takto natrénovaná síť má problém s ručně psanými texty, které vypadají vzhledově odlišně od trénovacích textů. Tyto vzhledové odlišnosti mohou například být rozdílný rukopis pisatelů apod. Problémem, jak adaptovat natrénovanou síť na takto odlišný text konkrétního pisatele, se zabývá moje práce.

Moje práce vzniká v rámci projektu PERO[1], který usiluje o co nejlepší rozpoznávání tištěného i ručně psaného textu. Důvodem proč mé zadání práce řeším je, že mimo jiné projekt PERO pracuje na digitalizaci vojenských deníků z 2. světové války. Tyto deníky jsou každý napsaný jiným pisatelem a každý z těchto pisatelů má jiný styl psaní. Aby neuronová síť dokázala dobře rozpoznávat text, který je takto stylově rozdílně napsaný, tak je zapotřebí, aby se na něj adaptovala. K této adaptaci využívám metodu, která řeší analogický problém adaptace na mluvčího. Tato metoda využívá regularizaci pomocí diskriminační sítě, která se snaží udržovat distribuci vah adaptační sítě podobnou té předadaptační. Podobnou distribucí vah, je myšleno podobnou z hlediska funkčnosti. To znamená, že se snaží regularizovat síť tak, aby regularizované části fungovali podobně, jako u předadaptační verze této sítě.

V této práci se nejprve věnuji širší oblasti znalostí, které jsou nezbytné pro pochopení samotné práce, ale i pro rozpoznávání pomocí konvolučních neuronových sítí, které využívají rekurentní vrstvy, jako celku. Dále se v práci věnuji metodám adaptace neuronových sítí a popisu dvou datasetů, které v rámci své práce využívám. Těmito datasety jsou dataset IMPACT, který obsahuje tištěné dokumenty z 16. až 20. století a druhým datasetem je dataset ručně psaných textů, který obsahuje velkou diverzitu jazyků a pisatelů. V poslední části mé práce se věnuji experimentování nad naimplementovaným adaptačním modelem. V této poslední části práce jsou uvedeny experimenty jak nad datasetem IMPACT, tak nad datasetem ručně psaných textů.

[1] Pokročilá extrakce a rozpoznávání obsahu tištěných a rukou psaných digitalizátů pro zvýšení jejich přístupnosti a využitelnosti

Kapitola 2

Strojové rozpoznávání textu

Strojové rozpoznávání textu je problém, kterým se zabývá část informačních technologiích široce známých jako počítačové vidění. Tato oblast má počátky již ve 40. letech minulého století. Programy, které se používají pro rozpoznávání znaků a jejich následný přepis jsou nazývány OCR (zkratka angl. Optical Character Recognition). OCR nástroje mohou mít vícero podob, například to mohou být různé skenery. Já se ve své práci budu soustředit čistě na OCR v podobě programu. V programové podobě se používá dvou přístupů. Prvním z nich je hledání shod ve vzoru. Moderní OCR využívají pro rozpoznávání textu druhou metodu, které se říká extrahování příznaků, a to z toho důvodu, že se ukázala jako efektivnější a obecněji použitelná [16]. Tato metoda je implementována v podobě neuronové sítě a v mojí práci právě tyto neuronové sítě využívám. Pro kvalitní natrénování neuronových sítí je zapotřebí získat velké množství trénovacích dat, ale i dobře natrénované neuronové sítě mají problém se stylově odlišnými daty. Tím, jak naučit neuronovou síť tyto stylově odlišná data kvalitně rozpoznávat (navíc z těchto dat je velmi omezený počet určený pro trénování sítí) se zabývá moje práce. Tato kapitola vysvětluje základní potřebné znalosti nejen pro samotné rozpoznávání textu, ale i pro strojové rozpoznávání pomocí neuronových sítí jako celku.

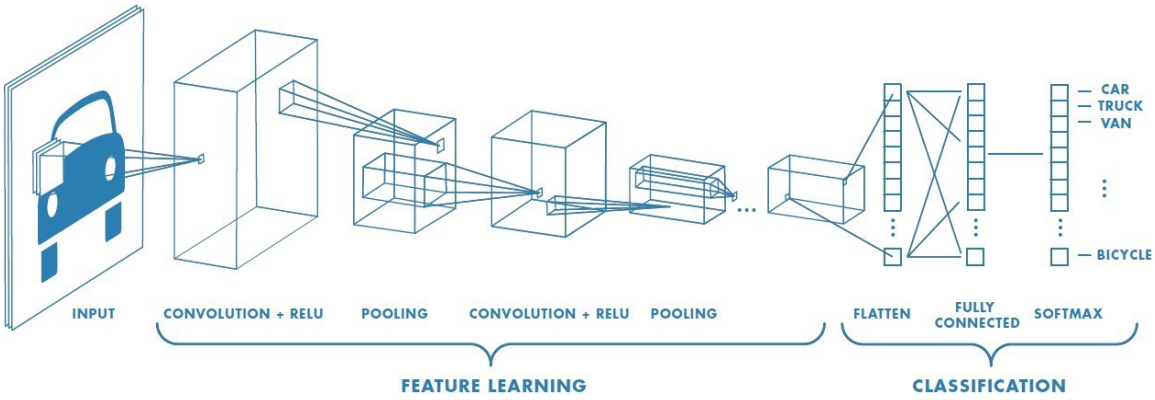
2.1 Konvoluční neuronové sítě

Neuronová síť je matematický model inspirovaný neurony v mozku. Každý neuron v neuronové síti má neomezený počet vstupů a pouze jeden výstup. Každý vstup do neuronu má přiřazenou váhu, se kterou se informace vstupující do neuronu násobí.

Suma všech vstupních hodnot pronásobených vahami je výstupem neuronu. Tato hodnota může být dále upravována aktivační funkcí, což je funkce, která převádí hodnotu do jiného rozmezí, nebo k ní může být přičtena odchylka (angl. bias). Výstupní hodnota neuronu může opět být vstupem jiného neuronu a nebo to může být výsledná hodnota sítě.

Vstupem neuronové sítě jsou data, která chci, aby síť rozpoznávala. Vstupem neuronových sítí pro rozpoznávání textu, které používám v této práci, je výřez textu v podobě obrázku, respektive jeho maticová reprezentace. Tato matice obrazu je tvořena třemi kanály. Je to dáno tím, že každý kanál reprezentuje jednu barvu (jednu složku) barevného spektra RGB. Síť z obrázku získává a určuje příznaky (angl. features), což jsou hlubší informace z matice obrazu.

K získávání těchto příznaků se používají konvoluce a filtry (neurony). Filtry jsou trojrozměrné matice, které se skládají z dvojrozměrných matic zvaných kernel. Každý filtr dokáže



Obrázek 2.1: Ukázka dopředného průchodu nad RGB obrázkem na vstupu. Feature learning je část sítě, která zpracovává vstupní obrázek a získává z něj příznaky. Část classification už rozhoduje o samotném zařazení obrázku do tříd (převzato z [11]).

v natrénované sítí ze vstupních dat získat informace. Tyto informace mohou být základní tvary jako například hrany, až po komplexnější, kterými jsou oči, pusa a podobně. Konvoluce je matematická operace, při které se posouvá matice filtru přes vstupní matici dat tak, že konvoluční filtr se řádkově posunuje po matici obrazu a jakmile dorazí na pravý kraj obrazu matice, tak se přesune na začátek a posune se o úroveň níž. Tento proces se opakuje pro celou matici vstupu. V každém kroku tohoto posunutí se provádí prvkové násobení mezi maticí vstupu a maticí filtru v oblasti, kterou filtr překrývá (filtr bývá typicky podstatně menší než vstupní matice obrazu). Hodnoty filtru reprezentují vstupní váhy neuronu. Výsledek těchto prvkových násobení, v jednom kroku posuvu, se sečte a tvoří jednu hodnotu nové matice. Jedna operace konvoluce typicky využívá vícero filtrů, takže v jedné konvoluci je N filtrů, které procházejí přes obrázek. Konvoluce jsou vidět na obrázku 2.1. Výsledkem konvoluce nad celou maticí obrazu je nová matice, které se říká matice příznaků (angl. feature map). Konvolucí v neuronových sítí bývá velké množství a zároveň s tím se využívá velké množství filtrů, které se postupně učí rozpoznávat příznaky v sítí. Síť jako celek je rozdělená do vrstev (angl. layers). Mohou to být například vrstva konvoluční nebo vrstva snižování velikosti matice (angl. pooling a tento anglický pojem budu nadále používat) atd. Výstupu těchto vrstev se říká aktivace sítě, výstup nebo příznaky.

Aby byla rozpoznávací schopnost sítě co nejlepší, je potřeba, aby síť obsahovala nelinearity, protože i samotné data na vstupu obsahují nelinearity. Mohou to být například skokové barevné přechody mezi pixely v obrázku apod. Toho se dosahuje předáním výstupní hodnoty neuronu do nelineární aktivační funkce, zařazením pooling vrstvy za konvoluční vrstvu atd. Aktivační funkce převede vstupní hodnotu na hodnotu v jiném rozsahu. Velmi častou aktivační funkcí bývá ReLU, která převede vstupní hodnotu na rozmezí $(0, \infty)$. Další operací, která se provádí v neuronových sítích, je pooling. Pooling je operace zmenšování dimenzionality matice za účelem zařazení nelinearity do sítě a zmenšení reprezentace vstupu. Díky této operaci se sníží výpočetní čas sítě, celková zdrojová náročnost a také se zabraňuje přetrénování (problém přetrénovaní popisuji v textu dále). Pooling funguje podobně jako konvoluce se svým principem posouvání klouzavého okna v podobě filtru přes vstupní matici, ale v případě poolingu je filtr pouze jeden. V každém místě překryvu mezi pooling maticí a maticí obrazu se získá (v závistlosti na typu poolingu) hodnota, která je vybrána jako zástupná hodnota pro tento maticový překryv a tvoří jednu hodnotu zmenšené matice. Přesunutím pooling matice přes celou matici obrazu, vznikne nová matice, která má menší

rozměry než matice, která byla na vstupu do pooling vrstvy sítě. Pooling je znázorněn na obrázku 2.1. Typů poolingů existuje vícero druhů, například Average pooling, Max pooling, který já používám ve své práci.

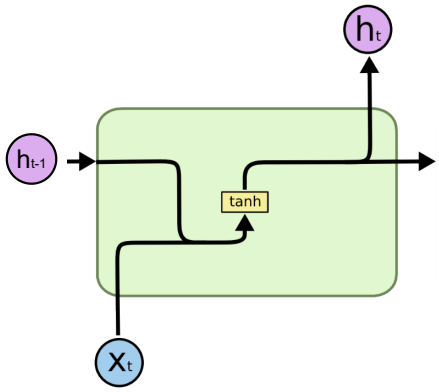
Operace konvoluce a pooling se několikrát zopakují až do bodu, kdy se matice zmenší natolik, aby byla vhodná pro operaci zploštění (angl. flattening). Zploštění převede vícedimenzionální matici na vektor. Tento vektor pak slouží jako vstup do plně propojené vrstvy neuronové sítě. V plně propojené vrstvě je každá hodnota vstupního vektoru připojena na každý neuron plně propojené vrstvy. Tato plně propojená vrstva je dále napojena buď na výstupní vrstvu a nebo na další skryté vrstvy sítě a až jejich výstup je přiveden na výstupní vrstvu. Tento výstup pak může být dál zpracován. Cestě dat z první vrstvy sítě, až po poslední vrstvu se říká dopředný průchod sítí (angl. forward pass). Ukázka dopředného průchodu sítí je vidět na obrázku 2.1.

Trénování sítí. Dříve, než dokáže síť kvalitně plnit svůj účel, je potřeba, aby byla natrénována. K trénování sítí jsou obecně používané 2 metody, a to učení s učitelem (angl. supervised learning) a učení bez učitele (angl. unsupervised learning). V této práci využívám k trénování sítí přistup učení s učitelem. Tato metoda spočívá v tom, že na vstup sítě předám data, která chci, aby se síť naučila rozpoznávat. Síť pak nad daty provede dopředný průchod a výstup sítě pak s očekávaným výstupem předám tzv. chybové funkci, která vypočítá chybu, které se síť dopustila, oproti očekávanému výstupu. Chybová funkce pak plní funkci učitele, která hodnotí výstup sítě oproti očekávanému výstupu a na základě gradientu jí vypočítané chyby se upravují hodnoty vah neuronů. Chybová funkce je více popsaná v podkapitole 2.3. Pro můj případ učení sítě rozpoznávat text, jsou vstupními daty výřezy řádků z obrázku a chybové funkci jsou předány přepisy řádků z těchto obrázků. To, kolik vstupních dat v jedné iteraci dopředného průchodu je, určuje parametr zvaný velikost dávky (angl. batch size). V případě předávání výřezů řádků textu na vstup, by velikost dávky 32 znamenala, že na vstup sítě půjde, v jedné iteraci trénovacího cyklu sítě, 32 takto vyřezaných obrázků řádků.

Jakmile chybová funkce vypočítá chybu, které se síť dopustila, je gradient této chyby takzvaně zpětně propagován sítí (angl. backpropagation). Zpětná propagace gradientu chyby se provádí průchodem sítí v opačném pořadí, než při dopředném průchodu a tomuto průchodu s gradientem chyby se říká zpětný průchod (angl. backward pass). Při zpětné propagaci se postupuje sítí a pro každý neuron v síti se upraví jeho váhy tak, aby se při dalším dopředném průchodu vylepšila rozpoznávací schopnost sítě na požadovaných datech.

Úkolem trénování je najít minimum chybové funkce vůči vahám sítě, protože to je bod, ve kterém je schopna síť nejlépe rozpoznávat požadované trénovací data. Asi nejčastěji používaným algoritmem, který slouží k nalezení minima dané funkce, je stochastický gradientní sestup (angl. Stochastic gradient descent). Stochastický gradientní sestup je numerická metoda (vzhledem ke komplexnosti sítě nelze použít analytické řešení), která má za cíl nalezení lokálního minima. Jejím problémem je, že nedokáže najít globální minimum a tím pádem nemusí najít bod, ve kterém síť určuje trénovací data s největší přesností. Vypočítat gradient nad všemi trénovacími daty by bylo příliš náročné a to je důvod, proč se využívá dávka (angl. batch). V jedné dávce jsou náhodně vybraná data z celkové množiny trénovacích dat, jejichž počet je dán parametrem velikost dávky. Z tohoto důvodu se používá i označení stochastický, neboli náhodný.

Při procesu trénování se běžně používají dva typy dat. Jedny data slouží pro samotné trénování sítě (trénovací data) a druhé slouží pro ověřování kvality rozpoznávání sítě na datech, které v průběhu trénování neměla k dispozici (testovací data). Aby síť dobře zvládala

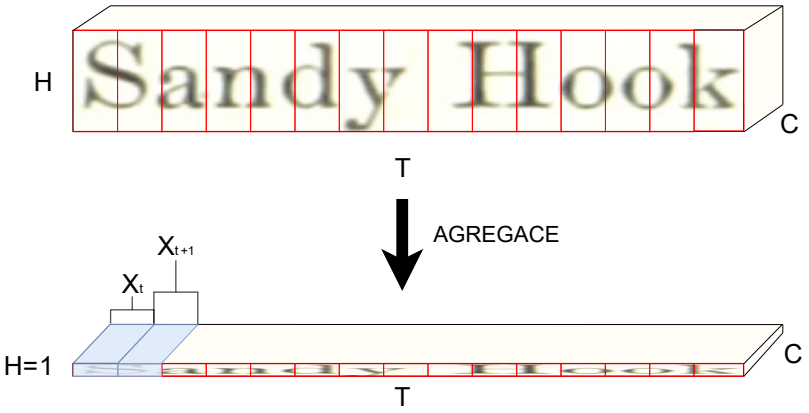


Obrázek 2.2: Blok klasické rekurentní sítě (převzato z [13]).

řešit požadovaný problém, je potřeba, aby byla trénována na velkém množství dat. V praxi je však třeba se omezit na menší množství trénovacích dat. Při trénování neuronových sítí na malých datových sadách může nastat situace, kdy se síť dokáže dokonale naučit rozpoznávat trénovací data a to způsobí zhoršení na datech podobných datům trénovacím, ale přesto rozdílným. Takovémuto naučení se říká přeučení a snahou je se mu vyhnout. Metodám, které se používá při trénování a jejichž úkolem je zamezit síti se přetrénovat, se říká regularizační metody. Vzhledem k tomu, že se zabývám problémem adaptace OCR na stylově odlišné texty, u nichž mám velmi omezenou trénovací sadu, by toto velmi malé množství trénovacích dat znamenalo, že si síť tato data poměrně rychle zapamatuje, dojde k přetrénovaní a tím se výrazně zhorší rozpoznávací schopnost na datech cílových (testovacích). Právě kvůli tomuto důvodu je potřeba rozumět regularizaci sítí a použít nějakou z regularizačních metod, které jsou více popsané podkapitole 2.4.

2.2 Rekurentní neuronové sítě

Stejně jako u lidí je zapotřebí, aby síť pro lepší rozpoznávací schopnost na textu, nebo obecně jakýchkoliv datech, měla k dispozici co nejširší kontext informací. Samotné konvoluční neuronové sítě mají kontext, nicméně ten je u nich poměrně malý. Existují sítě, které drží kontext celého vstupu a říká se jim rekurentní neuronové sítě. Využití při rozpoznávání textu může být například situace, kdy písmenko ve slově není čitelně napsané, ale síť díky znalosti slova, ve kterém se ono písmeno nachází, případně většího počtu slov nebo vět, dokáže určit, o jaké písmenko se pravděpodobně jedná. Mimo oblast rozpoznávání textu, se rekurentní neuronové sítě dají použít kdekoliv, kde se rozpoznávají informační sekvence, které i mohou být pomocí rekurentních sítí predikovány. Za účelem zapamatování informací je potřeba, aby si rekurentní síť uchovávala kontextové informace. Tyto kontextové informace jsou uchovány ve vnitřním stavu bloku sítě a to bez toho, aby byly časově limitovány. Blok klasické rekurentní vrstvy je vidět na obrázku 2.2. x_t je vstup v aktuálním čase, jak je vidět na obrázku 2.3 a tento vstup se konkatenuje s výstupem rekurentního bloku v minulém časovém okamžiku označeném jako h_{t-1}. Každý blok rekurentní sítě bere jako vstup časové okamžiky obrázku (rozdělenou šířkovou dimenzi). Lze si představit, že obrázek 2.2 je za sebou několikrát a vždy na vstup bere x_t a h_{t-1}, další pak x_{t+1} a h_t. Tyto dva zkonkatenované vstupy jsou předány aktivační funkci, jejíž výstup je zároveň výstupem bloku v tomto čase. Ovšem při trénování klasických rekurentních neuronových sítí, dochází

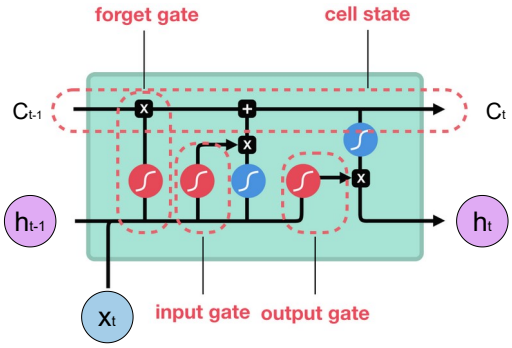


Obrázek 2.3: LSTM na svůj vstup bere vektor. Matice, s kterou počítá síť, je třírozměrná a z toho důvodu musí být upravena na 2d podobu a dále upravená, před vstupem do LSTM. Procesu transformace třídimenzionální matice na dvojdimenzionální se říká agregace. Osy na obrázku popsané jako H,T a C značí výšku matice, časovou osu (šířku matice) a kanálovou dimenzi.

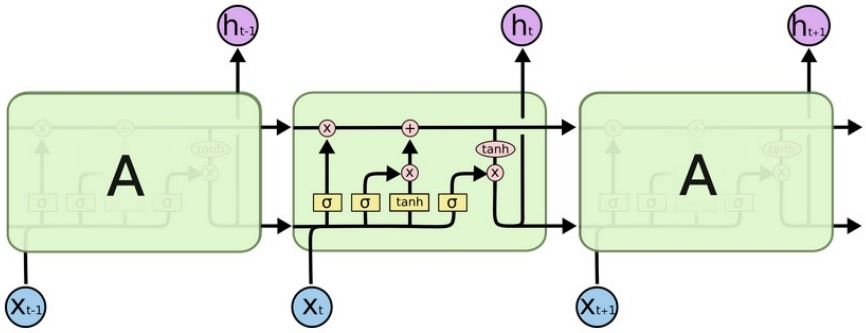
k problémům při zpětné propagaci. Problémy, které jsou u klasických rekurentních sítí, jsou problém vyhasínání gradientu (angl. vanishing gradient) a problém s velkým nárůstem gradientu (angl. exploding gradient) [2]. Problém s vyhasínáním je způsobem tím, že díky architektuře rekurentní sítě, se při zpětné propagaci zmenšuje v čase gradient až do chvíle, kdy je tak malý, že nemá vliv na váhy u v čase vzdálenějších rekurentních bloků. Problém velkého nárůstu gradientu je způsoben tím, že se v sítí gradienty nakumulují a to má pak za následek příliš velkou korekci vah a nakonec jejich oscilaci a tím pádem destabilizaci sítě při trénování. V extrémním případě může dojít i přetečení hodnot v sítí.

Za účelem eliminace těchto problémů byl vynalezen rekurentní síťový blok LSTM (zkratka z angl. Long short-term memory) [9]. Jedná se o upravené rekurentní sítě, které jsou přímo navrženy tak, aby nedocházelo k problémům s vyhasínání gradientu a přílišného nárůstu gradientu. Rozkreslený LSTM blok v čase je vidět na obrázku 2.5. LSTM na svůj vstup bere vektor a to znamená, že vygenerovaný výstup klasické neuronové sítě, kterým je trojdimenzionální matice (je tvořena kanálovou dimenzí, výškou a šířkou), musí být upraven na dvojdimenzionální podobu a následně ještě rozdělen v horizontální ose tak, jak je zobrazeno na obrázku 2.3. Procesu transformace trojdimenzionální matice na dvojdimenzionální nebo obecněji procesu, kdy chci více hodnot převést na jednu hodnotu, se říká agregace. Při agregaci matice pro vstup do LSTM vrstvy sítě, je potřeba zmenšit výšku 3d matice H na výšku 1. Tímto získám dvojdimenzionální matici, ze které po rozdělení na horizontální ose na časové okna, dostanu vektory, které je již možné předat LSTM vrstvě a na kterých se již dokáže učit. Toto rozdělení je provedeno konvolucí nad dvojdimenzionální maticí. Tyto vektory jsou na obrázku 2.3 označené jako x_t a x_{t+1}. Díky svojí architektuře LSTM blok umožňuje propagovat gradient chyby skrz blok bez ztrát.

Schopnost snadné propagace gradientu chyby je dána linkou v LSTM bloku. Přístup k této vnitřní lince, na jejímž výstupu je hodnota, které se říká stav buňky (angl. cell state), je omezen a to pomocí bran. Každý LSTM blok obsahuje tři takovéto brány. Jsou jimi zapomínací brána (angl. forget gate), vstupní brána (angl. input gate) a výstupní brána (angl. output gate). Přímý přístup k této lince z nich mají dvě. Vstupní brána přidává do



Obrázek 2.4: Ukázka LSTM buňky s bránami (v obrázku angl. gate) a stavem buňky (v obrázku angl. cell state). c_t, c_{t-1} jsou stav buňky, respektive minulý stav buňky, h_t a h_{t-1} jsou výstup současného a výstup minulého LSTM bloku. Vlnka v červeném kroužku značí aktivační funkci sigmoid a vlnka v modrém kroužku značí aktivační funkci tangens (převzato z [15]).



Obrázek 2.5: Obrázek zobrazující LSTM blok rozkreslený v čase. Každý LSTM blok obsahuje strukturu, která je vidět na prostředním bloku a ve které x a + v kroužku značí násobení respektive sčítaní po prvcích a dále obsahují brány, které jsou ve žlutých obdélnících. x_t značí vstup bloku a h_t výstup (převzato z [13]).

linky informace pomocí operace sčítání a zapomínací rozhoduje o ponechání, nebo odebrání informací, které jsou výstupem minulého LSTM kroku, pomocí násobení hodnot v rozmezí nula až jedna s hodnotami v lince bloku. Schopnost bezztrátového toku informací je dána tím, že při zpětné propagaci se gradient chyby u operace sčítání pouze větví, ale jeho samotná hodnota není pozměněna. U násobení může nastat situace, kdy se síť naučí generovat na vstup násobení samé hodnoty jedna a v tu chvíli by nebyl gradient ovlivněn (protože lokální gradient funkce, která násobí hodnotou jedna, je vzhledem ke vstupu jedna a díky tomu se gradient chyby nijak nezmění) a propagoval by se dál bloky bez ztráty.

Princip LSTM je takový, že vstup do bloku x_t se zkonkatenuje s výstupem minulého časového kroku, na obrázku 2.4 označeným h_{t-1} (stejně jako u klasické rekurentní sítě), případně s náhodně inicializovanou hodnotou, pokud se jedná o první vstup do rekurentní vrstvy sítě a následně tento zkonkatenovaný vektor je předán na všechny brány. První bráně se říká zapomínací. K rozhodnutí o tom, jaké hodnoty, z předchozího vnitřního stavu, se zapomenou (na základě vstupu a předchozího výstupu), používá zapomínací brána následující

rovnici:

$$f_t = \sigma \times (W_f \times [h_{t-1}, x_t] + b_f) \tag{2.1}$$

kde W_f je váhová matice, h_{t-1} je výstup předchozího LSTM kroku, xt je současný vstup a b_f je přičtená hodnota zvaná odchylka (angl. bias). Hranaté závorky značí konkatenaci. Tyto parametry jsou předány aktivační funkci sigmoid, značené σ.

Funkce sigmoid převede vstupní hodnotu na hodnotu v rozmezí 0 až 1. Pak je každá takto získaná hodnota vynásobená hodnotou z minulého stavu buňky a z toho tedy vyplývá, že 0 informaci odebere kompletně a 1 ji celou ponechá. Vstupní brána funguje obdobně, ale rozhoduje o tom, jaké hodnoty budou do stavu buňky přidány, což je způsobeno tím, že k hodnotám na lince bloku přispívá přes operaci sčítání. Rovnice vstupní brány je:

$$i_t = \sigma \times (W_i \times [h_{t-1}, x_t] + b_f) \tag{2.2}$$

Výstup této brány je prvkově násoben s vektorem vycházejícím z funkce tangens. Vektor vycházející z tangens, je vytvořen podle rovnice:

$$G_t = \tanh \times (W_c \times [h_{t-1}, x_t] + b_f) \tag{2.3}$$

Takto pronásobený vektor je přidán do stavu buňky prvkovým sčítáním. Nový stav buňky se vypočítá podle rovnice:

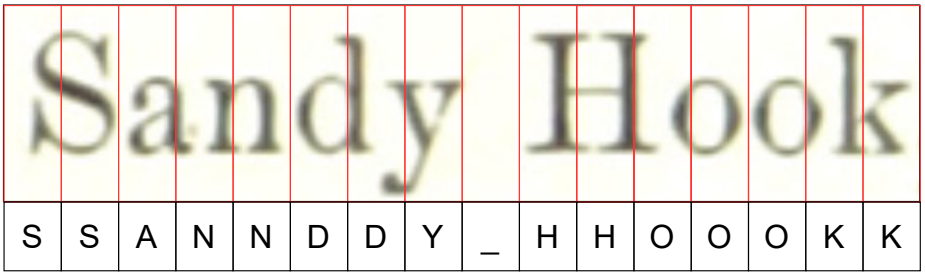
$$C_t = C_{t-1} \times f_t + G_t \times i_t \tag{2.4}$$

Tento nový stav je společně se současným výstupem h_t předáván dalšímu bloku LSTM (dalšímu v čase, jedná se o ten samý blok), pokud následuje další blok. h_t se vypočítá podle rovnic:

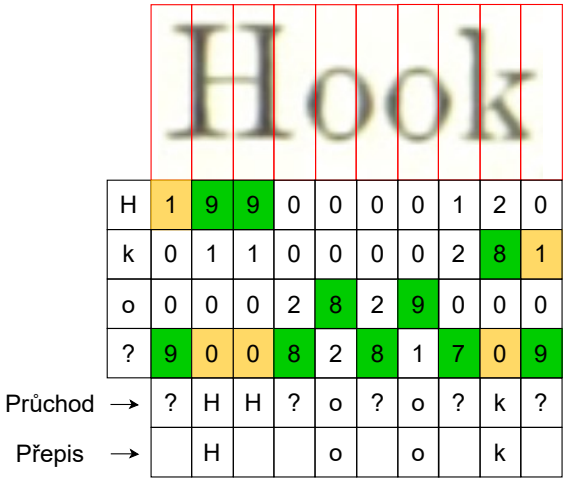
$$o_t = \sigma(W_o \times [h_{t-1}, x_t] + b_o) \tag{2.5}$$

$$h_t = o_t \times \tanh(C_t) \tag{2.6}$$

Popsaná LSTM vrstva brala jako vstup časová okna zleva doprava $t == 0$ až $t = max$. LSTM, která bere takovéto vstupy, dokáže predikovat výstup na základě minulosti. Principiálně je ale taky možné předávat LSTM časová okna zprava doleva a tím pádem bude LSTM predikovat na základě budoucnosti. V praxi se také používá takzvaná obousměrná (angl. bidirectional) LSTM, což je síť, která dostává jak kontext minulosti, tak zároveň kontext budoucnosti.



Obrázek 2.6: Ukázka ručního anotování textu. Potržítko značí mezeru.



Obrázek 2.7: Výsledný přepis získaný z CTC. Hodnoty u písmen značí pravděpodobnosti výskytu vynásobené 10. Zelené políčka značí výskyt písmena s nejvyšší pravděpodobností v jednotlivých časových okamžicích. Zeleně označené časové okna značí nejpravděpodobnější průchod. Přepis se získá výběrem písmen s největší pravděpodobností a shluknutím stejných písmen v sousedních časových oknech. Žluto oranžová barva v jednotlivých oknech značí možnost záměny se zelenou. Při průchodu přes tyto zaměněné žluto oranžové okna by byl stále získán správný přepis.

2.3 Chybová funkce pro rozpoznávání textu

Problémem, který s rozpoznáním správnosti výstupu sítě, která rozpoznává text je, že můžu mít několik obrázků, na kterých je stejný text a přesto mohou být různé délky. Tento problém by šel řešit anotováním (označováním vstupů sítě pro chybovou funkci) jednotlivých obrázků a to tak, jak je znázorněno na obrázku 2.6. Toto řešení je ale poměrně problematické z toho důvodu, že takto anotovat vstup by bylo nesmírně náročné a taky významně sporné. Spornost je zapříčeněna stejnými znaky ve vedlejších časových okamžicích (časové okamžiky jsou vytvořeny konvolucí nad vstupem). Naivním řešením tohoto problému je sousední stejné znaky shlukovat, nicméně to by vedlo ke špatným přepisům například u slova jako je "Hook"z obrázku 2.6. Při takovémto přístupu by ze slova "Hook"vzniklo slovo "Hok"což je chybný přepis. Řešením problémů, které jsem popsal, je použít chybovou funkci CTC (zratka z angl. Connectionist Temporal Classification) [7].

Vstupem chybové funkce CTC je matice sítě, která má dimenzi časovou (horizontálně rozdělenou podle konvolucí) a dimenzi znakovou. O co se CTC snaží je, aby síť na svém

výstupu generovala matici pravděpodobností výskytu písmen v znakové dimenzi. Takováto matice je vidět na obrázku 2.7. V dimenzi znakové jsou všechny znaky, které síť dokáže rozpoznávat a je rozšířená o speciální prázdný znak (angl. blank), který je na obrázku 2.7 zaznačený jako otazník. Aby bylo možné takovouto pravdivostní matici generovat, tak je potřeba, aby výstupní matice ze sítě, byla předána aktivační funkci softmax, která převede hodnoty ve znakové dimenzi do rozsahu 0 až 1 a to tak, aby suma hodnot ve znakové sadě v jednom časovém okamžiku byla rovna 1. A proto hodnoty 0 až 1 u znaků ve znakové sadě značí pravděpodobnost výskytu znaků v jednotlivých časových okamžicích.

U CTC jsou důležité koncepty průchodu. Průchod znamená, že z každého časového okna vyberu jednu hodnotu (jedno písmeno) a pravděpodobnost průchodu je dána pronásobením pravděpodobností přes časové kroky. Z průchodu se získává výsledný přepis. Tento přepis se získává tak, že z průchodu vyberu znaky, po jednotlivých časových oknech, které mají nejvyšší pravděpodobnost výskytu a jestliže jsou v nějakých sousedních časových oknech dva stejné znaky, tak je shluknu do jednoho. Problém, který byl u naivního přístupu se shlukováním sousedních znaků, které by ale díky tomuto shluknutí dávaly špatný přepis, je u CTC vyřešen použitím speciálního prázdného znaku. Tento znak odděluje jednotlivá písmena a jestliže jsou ve vstupním obrázku zdvojená písmena, tak se tento prázdný znak nageneruje mezi tyto písmena a díky tomu se neshluknou. Příklad získávání přepisu z průchodu je vidět na obrázku 2.7. Validní průchod je takový, ze kterého přepisem získám přepis správný. Na obrázku 2.7 jsem zelenou barvou naznačil nejpravděpodobnější průchod sítí. Krom tohoto průchodu, který je validní, jsou v obrázku některé časová okna označené žluto oranžovou barvou a tyto časové okna lze nahradit za zelené ve sloupcích a i při tomto nahrazení by síť vygenerovala správný přepis. Díky koncepci průchodů se řeší problém různé délky vstupních obrázků s tím samým textem a to díky tomu, že po síti nepožaduji, aby generovala písmena na konkrétní pozici, ale pouze požaduji, aby generovala validní průchody. Výsledná chyba CTC se vypočítá podle rovnice:

$$E = -\ln(\sum_{\pi \in B} p(\pi|x)) \tag{2.7}$$

kde B je množina validních průchodů π je současný průchod, x vstupní obrázek textu a p označuje pravděpodobnost.

Váhy sítě se upravují na základě gradientu chyby od chybové funkce. Gradient CTC funkce se vypočítá prvkovým odečtením ground truth matice (ground truth znamená pravdivá a tento anglický pojem budu nadále v textu používat), vytvořenou CTC, od výstupu sítě. To, jak se tato ground truth matice vytváří, je mimo rozsah této práce.

2.4 Regularizace

Aby se model neuronové sítě nepřeučil, a tím pádem nedošlo k zhoršení na cílových datech, je zapotřebí použít nějaký druh regularizace. K přetrénování dochází, když se síť trénuje na malém počtu dat a obsahuje velké množství vah. Velké váhy v síti znamenají velký vliv. U sítě s mnoha velkými vahami by bez regularizace docházelo s počtem trénovacích iterací ke zlepšení na trénovacích datech, ale zhoršení rozpoznávání sítě na datech cílových (data, na kterých se síť netrénovala, ale požaduji na nich podobnou rozpoznávací úspěšnost, jako na datech trénovacích) a nakonec k přetrénování.

Často používanými metodami jsou L1 a L2 regularizace, zahazování (angl. dropout) [18] a augmentace. L1 a L2 regularizace, kterým se říká váhové regularizace, oproti droupoutu a augmentaci, přidávají term do chybové funkce. Důvodem této úpravy je snaha, aby nedošlo

Obrázek 2.8: Ukázka výsledku augmentace na vstupním obrázku řádku je vidět v podobě šumu.

k vzniku velkých vah v síti, protože vysoce váhové neurony i při malém vstupním podnětu vyprodukují velký výstup a tím se podněty od ostatních neuronů stávají méně podstatné. L1 a L2 přidávají vliv vah sítě do výsledné chyby a takovéto úpravě výsledné chyby, se říká penalizace vah a je provedena podle rovnic:

$$L1 = chyba + \lambda \times \sum_{i=1}^{N} |w_i| \tag{2.8}$$

$$L2 = chyba + \lambda \times \sum_{i=1}^{N} w_i^2 \tag{2.9}$$

V rovnicích *chyba* značí výsledek chybové funkce, lambda je parametr ovlivňující sílu regularizace a w označuje váhy v síti. Díky přidání vlivu vah se velké váhy promítnou do výsledné chyby více, což povede při zpětné propagaci k větší úpravě velkých vah a jejich snížení. Obě metody se snaží snížit hodnoty vah k nule, nicméně L1 přidává součet hodnot vah do výsledné chyby a tím penalizuje všechny váhy podobně, L2 (také zvaná anglicky jako weight decay) více penalizuje velké váhy před menšími.

Zahazování v každé iteraci vynechá náhodně zvolené neurony, tím pádem se s těmito neurony nepočítá při průchodu sítí. To znamená, že síť při každém dopředném průchodu pracuje s lehce pozměněným a zjednodušeným modelem. Díky tomuto přístupu nevznikají mezi jednotlivými neurony v síti tak pevné a závislé propojení, které by jinak napomáhaly přeučení sítě. Při zahazování je důležitý parametr míra zahazování (angl. dropout rate), který vyjadřuje pravděpodobnost, že se s konkrétním neuronem bude při tréninku sítě počítat. Množství zahazování 0 znamená, že žádné zahazování nebude, 1 znamená že se s neuronem počítat nebude.

Augmentace je typ regularizace nad trénovacím vstupem. Augmentace využívá různých datových manipulací. Jejím účelem je transformace vstupních dat tak, aby se od sebe určitým způsobem lišily. Augmentace nad vstupním trénovacím obrázkem znamená, že nad ním provede jednu, nebo více operací, které ho změní. Změní ho tak, že nad ním provede operaci zvetšení, zmenšení, rotaci, přidání šumu do obrázku nebo například změní perspektivu pohledu na obrázek. Změnou trénovacího obrázku se napomáhá tomu, že i když síť na vstup dostane obrázek, který už v průběhu tréninku viděla, tak bude pozměněný a díky tomu se váhy sítě nebudou mít takovou tendenci zvětšovat vůči tomuto obrázku a tím pádem nebude docházet k přeučení. Ukázka augmentace řádku textu je na obrázku 2.8.

Adversarial adaptation Cílem mé práce je vytvořit síť, která bude schopna se na velmi malém množství dat naučit lépe rozpoznávat jemně odlišné data od dat trénovacích. Při řešení tohoto problému budu vycházet z článku Adversarial Speaker Adaptation [12]. Autoři článku jsou zaměstnanci firmy Microsoft a pracují v rámci skupiny jazyk a řeč. V tomto článku se zabývali řešením analogického problému jako řeším já, avšak jejich problém byl

v rozpoznávání řeči. V rámci automatic speech recognition (angl. zkratka ASR), neboli automatického rozpoznávání řeči, se ukázalo, že i velmi dobře natrénovaná síť na trénovacích datech, kterých je velké množství, má problém, pokud existují mezi trénovacími a cílovými daty akustické nepřesnosti. Jako příklad akustické nepřesnosti jsi lze představit například přízvuk. Tým těchto zaměstnanců přišel s řešením, které se ukázalo jako efektivní. Hlavní myšlenku autorů článku Adversarial Speaker Adaptation a jak tuto metodu aplikovat na problém rozpoznávání textu vysvětlím v další kapitole.

Kapitola 3

Metody adaptace neuronových sítí

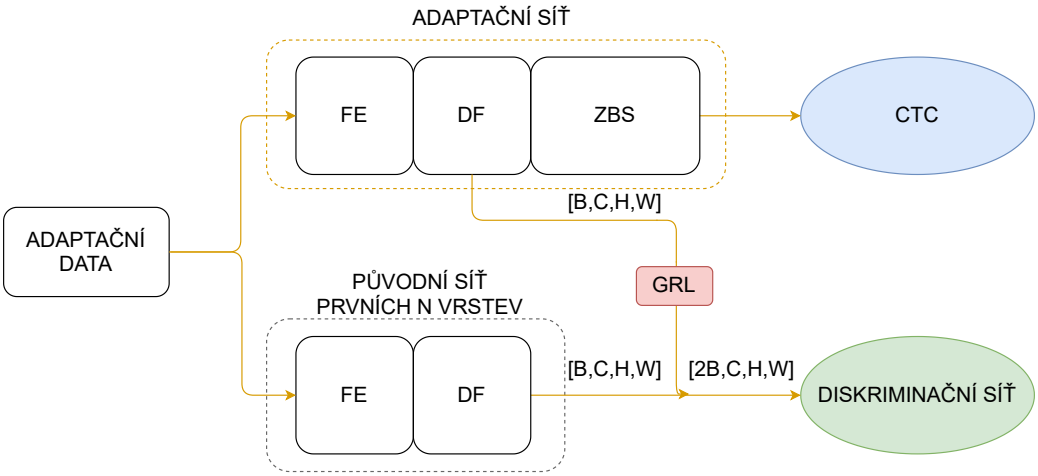
V této kapitole se zabývám řešením problému adaptace neuronových sítí. Toto řešení vychází z článku Adversarial Speaker Adaptation [12] a v této kapitole je popsán způsob využití metody z článku, ve kterém se jedná o adaptaci na mluvčího, pro adaptaci na pisatele. Součástí této kapitoly, je také popis toho, jak by šla provádět adaptace naivním způsobem a o problémech naivního způsobu.

3.1 Naivní řešení problému adaptace

Problém špatného rozpoznávání na stylově odlišných datech by bylo možné řešit naivním způsobem a to tak, že na dobře natrénovaném modelu neuronové sítě bych spustil trénovaní na adaptačních trénovacích datech. Tento způsob by měl za následek, že síť by se prvně zlepšila na cílových datech, ale v závislosti na počtu adaptačních řádků, by došlo k přetrénování sítě a to by znemožnilo další zlepšení na cílových datech a velmi pravděpodobně by způsobilo dokonce zhoršení rozpoznávání na těchto cílových datech. Tímto způsobem by se však z trénovacích dat, díky přetrénování, nedokázalo získat maximum, v tom smyslu, že kdyby k přetrénování nedošlo, tak by byla rozpoznávací schopnost sítě na cílových datech podstatně lepší. Aby nedošlo k přetrénování, tak je obecně potřeba použít regularizaci. V tomto případě by bylo možné vstupní trénovací data augmentovat, a tím pádem dostat lepší výsledek na cílových datech.

Další možností, jak naivně řešit problém adaptace, je vstupní adaptační data zamíchávat s původní datovou sadou, na které byl model natrénován. Takovéto zamíchání dat by vedlo ke zlepšení sítě na nových datech, nicméně by takovýmto způsobem řešená adaptace trvala mnoho iterací a dalším problémem je potřeba původních dat. Tyto data mohou být velmi špatně získatelná a nebo dokonce je ani není možné získat. Další problém může být ten, že původní síť je natrénovaná na podobný problém, jako je získávání přepisů textu, ale přeci jen jsou její původní data odlišná. Pro příklad může být síť trénovaná pro získávání textu ze značek atd., kde by mixování adaptačních dat (například texty z historických knih) a původních dat nedávalo příliš smysl.

Řešení, které v rámci práce využívám, představili autoři článku Adversarial Speaker Adaptation a popisem tohoto řešení se budu zabývat v následující podkapitole 3.2. Nezanedbatelnou předností tohoto řešení je, že pro adaptaci nevyžaduje původní data.



Obrázek 3.1: Podoba modelu vycházející z článku Adversarial adaptation. Samotnou architekturu obou sítí prezentuji v části o experimentech 5. FE v obrázku reprezentuje extraktor příznaků, DF pak označuje hluboké příznaky a ZBS reprezentuje zbylou část sítě. Hodnoty v hranatých závorkách značí velikosti matice, které z vrstvy sítě vycházejí, písmena B,C,H,W značí velikost dávky, C značí kanálovou dimenzi, H výšku matice a W je šířka matice.

3.2 Adversarial Adaptation

Autoři článku Adversarial Speaker Adaptation přišli s řešením problému adaptace neuronových sítí v rámci výzkumné skupiny jazyk a řeč pro jejich model rozpoznávání mluveného slova. Tento problém je podobný problému adaptace neuronových sítí na pisatele. Díky hlavní myšlence článku je možné provést adaptaci pouze na cílové množině dat, bez nutnosti používat původní data. Řešení těchto autorů je v úpravě několika posledních vrstev sítě a zachování podobné distribuce vah pro několik prvních vrstev neuronové sítě (váhy se mohou lišit, ale vrstvy sítě, by měly produkovat podobné výstupy jako před adaptací). Tuto myšlenku představuji v následující podkapitole.

3.3 Princip metody adaptace

Princip metody adaptace představené autory spočívá ve vytvoření dvou neuronových sítí, z nichž jedna je označená jako síť původní a druhá jako síť adaptační a následné trénování adaptační sítě za použití nových dat, ale takovým způsobem, aby se distribuce vah několika prvních vrstev adaptační sítě příliš nevzdálila distribuci stejných vah původní sítě. Architektura této soustavy je zobrazena na obrázku 3.1. Původní síť je předtrénovaná na velkém množství dat, na kterých funguje dobře, a hodnotami jejích vah jsou inicializovány váhy adaptační sítě, takže adaptační síť a síť původní mají na začátku procesu adaptace naprosto stejnou rozpoznávací schopnost. Jinými slovy to znamená, že na stejná vstupní data budou produkovat stejné aktivace. Původní síť v procesu adaptace se nijak netrénuje, je zafixovaná, její váhy zůstávají po celou dobu adaptace stejné. Jejím účelem, respektive účelem jejích vah, je být ukazatelem, jak moc se distribuce vah adaptační sítě v průběhu adaptace liší od distribuce vah původní sítě. Každá ze sítí se dá pomyslně rozdělit na dvě části. První část je v článku označována jako extraktor příznaků (angl. feature extractor) a druhá část jako hluboké příznaky (angl. deep features). První část sítě lze chápat jako ex-

traktor informací z obrázků a jedná se o několik prvních vrstev sítě. Hluboké příznaky jsou část sítě následující za extraktory příznaků. Aktivace některé hlubší vrstvy obou sítí jsou vstupy do diskriminátoru. Jelikož se aktivace obou sítí, pro vstup do diskriminátoru, berou za stejnými vrstvami, tak původní síť v procesu adaptace není kompletní, ale využívají se pouze a jen vrstvy, které jsou před vstupem do diskriminační jednotky.

3.4 Diskriminační síť

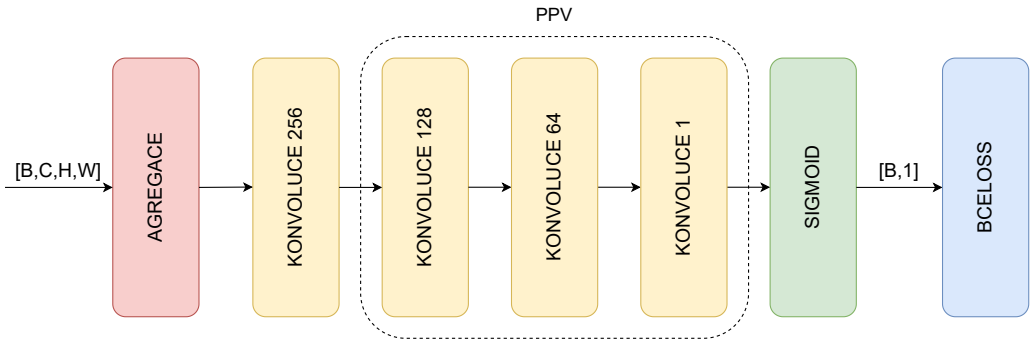
Diskriminátor je neuronová síť, jejímž úkolem je rozhodovat o tom, z které větve, neboli z které sítě, jí přišla aktivace. Obecně je úkolem diskriminátoru rozhodnout o tom, odkud mu předaný vstup došel. V rámci článku je jeho úkolem regularizovat adaptační síť. Regularizuje ji tak, že od její chybové funkce se do adaptační sítě propaguje gradient, který má za úkol zachovat distribuci vah adaptační sítě podobou těm původní sítě, aby obě sítě na stejné vstupy reagovali podobně, respektive aby část adaptační sítě, na kterou působí diskriminátor, produkovala podobné výstupy, jak stejná část původní sítě. Problém je, že pokud bych použil samostatný diskriminátor, tak by přijal data a rozhodl by o tom, z které sítě došly, následně by se spočítala chyba, které se dopustil při svém rozhodnutí (diskriminátor trénuji také metodou učení s učitelem) a při zpětné propagaci gradientu této chyby, by došlo k úpravě vah jak diskriminátoru, tak vah obou sítí a to takovým způsobem, že váhy těchto sítí by se začaly od sebe příliš vzdalovat a neprodukovaly by podobné výstupy. Tím pádem by došlo k zlepšení rozpoznávací schopnosti diskriminátoru. Nastal by přesný opak toho, co potřebuji. Tento problém řeším tak, že mezi adaptační síť a diskriminátor přidám jednotku otáčení gradientu (angl. Gradient reversal layer) [5], která při zpětné propagaci otáčí znaménko gradientu vycházejícího z chybové funkce diskriminátoru. Rovnice podle kterých se řídí vrstva otáčení gradientu jsou:

$$R_\lambda(x) = x \tag{3.1}$$

$$\frac{dR_\lambda}{dx} = -\lambda \tag{3.2}$$

ve kterých R_λ je pseudofunkce, která je definována dvěma rovnicemi, z nichž (3.1) popisuje její dopředný průchod a druhá popisuje její chování zpětného průchodu (3.2). (3.2) reprezentuje výpočet lokálního gradientu, který se pak přinásobí ke gradientu přicházejícímu z diskriminační sítě. λ je parametr, který v mém případě ovlivňuje velikost regularizace od diskriminátoru. Díky tomuto otočení znaménka bude mít gradient vycházející z diskriminátoru efekt regularizace na váhy adaptační sítě a to takový, že se bude snažit o to, aby regularizované části adaptační sítě, produkovaly podobné výstupy jako stejné části původní sítě.

Očekávaný účinek diskriminační sítě na průběh adaptace. Očekávaným průběhem adaptace je, že diskriminátor, na začátku adaptace nebude schopný rozpoznávat, z které části sítě vstup došel. Po tom, jakmile gradient od CTC funkce více změní váhy adaptační sítě, tak diskriminátor bude schopný lépe rozpoznávat původ vstupu. V tuto chvíli se diskriminátor bude díky GRL snažit držet distribuci vah adaptační sítě (ty, které jsou před vstupem do diskriminátoru) podobnou těm původní sítě. Díky tomuto dojde ke zhoršení rozpoznávací schopnosti adaptační sítě a samotný diskriminátor pak opět bude mít větší problém rozlišovat mezi vstupy, protože si budou podobnější. Kvůli zhoršení rozpoznávání



Obrázek 3.2: Architekruta diskriminační sítě. Hodnoty v hranatých závorkách reprezentují matice na vstupu a výstupu sítě. B v takovéto reprezentaci matice značí velikost dávky, C kanálovou dimenzi, H výšku matice a W šířku matice.

adaptační sítí, pošle CTC do adaptační sítě větší gradient, což opět způsobí menší chybovost diskriminátoru a ten opět v dalším kroku přiblíží distribuci vah adaptační sítě té původní sítě. Tento proces se opakuje, ale takovým způsobem, aby se chyba na adaptačních a cílových datech zmenšovala. Dochází tak k efektu oscilace chybovosti na diskriminátoru. V procesu adaptace očekávám takovýto způsob soupeření mezi CTC a diskriminátorem.

Architektura diskriminátoru. Diskriminační sít, kterou jsem vytvořil, má poměrně jednoduchou architekturu, která je vidět na obrázku 3.2. Skládá se z konvoluční vrstvy, která agreguje výškovou dimenzi, agregační vrstvy, která vypočítá průměr hodnot nad šířkou matice, jedné konvoluční vrstvy, která zvětší kanálovou dimenzi na 256 a pak následují tři plně propojené vrstvy, jejichž výstupní kanálové dimenze jsou 128, 64 a 1. Za těmito vrstvami ještě následuje aktivační funkce Sigmoid, z které jsou výstupy předány chybové funkci BCELoss[1]. První konvoluční vrstva srazí výškovou dimenzi vstupní matice, ale je do samotné architektury přidána v závislosti na vstupu. Vstupem je matice, která má buďto 4, nebo 3 dimenze. U čtyřdimenzionální matice jsou dimenze velikost dávky, počet kanálů, výška matice a šířka matice. U třídimenzionální jsou vnitřní dimenze stejné jako u čtyřdimenzionální, ale neobsahuje výškovou dimenzi. Vzhledem k tomu, že agregační vrstva je do architektury zařazena právě kvůli snížení výškové dimenze, tak se použije a nebo nepoužije v závislosti na dimenzionalitě vstupní matice.

Vstup diskriminační sítě. Vstupem diskriminační sítě jsou zkonkatenované aktivace n-té vrstvy původní a adaptační sítě, podle jejich prvních dimenzí. Velikost první dimenze matic odpovídá velikosti dávky. Zkonkatenované matice jsou předány diskriminátoru. Pokud bych ve svojí práci používal velikost dávky 32, tak bych po zkonkatenování obou matic vycházejících ze sítí, podle jejich prvních dimenzí, dostal matici, která má velikost první dimenze 64.

Chybová funkce diskriminátoru. Chybová funkce diskriminátoru bere jako parametry výstupní matici diskriminátoru a cílovou matici, což je matice očekávaných hodnot. Úkolem této chybové funkce je určit chybu, jaké se síť dopustila v predikci toho, odkud výstup sítě přišel. Matice výstupu z diskriminátoru, vstupující do chybové funkce, má rozměry $Bx1$,

[1] Binární křížová entropie

kde B označuje velikost dávky. Pro případ, kdy je velikost dávky vstupní matice původní a adaptační sítě 32, tak výstupní matice z diskriminátoru bude mít rozměry $64x1$. Tato jediná hodnota je reprezentativní pro každý řádek vstupující do diskriminátoru. Jedná se o reprezentaci jednoho řádku v dávce vyprodukovanou buď původní, nebo adaptační sítí.

Cílovou matici jsem si pak ručně vytvořil a to tak, že jsem si zvolil jako reprezentativní hodnotu pro adaptační síť 0, hodnotu reprezentující původní síť 1 a podle velikosti první dimenze výstupní matice diskriminátoru jsem vytvořil matici, s první polovinou míst 0 a druhou polovinou 1 (protože první polovina hodnot bude vygenerovaná adaptační sítí a druhá polovina hodnot původní sítí), tak aby součet všech těchto pozic byl roven velikosti první dimenze matice z diskriminátoru, která je velikost dávky.

Z důvodu, aby hodnoty výstupní matice diskriminátoru byly v rozmezí od 0 do 1 (protože cílová matice má hodnoty 0 a 1), jsem do diskriminační sítě přidal na úplný konec aktivační funkci Sigmoid, jak je vidět na obrázku 3.2, která vstupní hodnotu převede do intervalu 0 až 1.

Autoři článku pracovali s chybovou funkcí křížovou entropií (angl. Cross Entropy Loss). Pro můj případ tato chybová funkce rozhoduje o tom, jestli výstupní hodnoty diskriminátoru odpovídá správné očekávané hodnotě reprezentující síť, neboli jak moc špatně, nebo jak moc dobře diskriminátor určil původ dat. Křížová entropie dokáže klasifikovat výstup do 0 až N možných kategorií a vzhledem k tomu, že já mám pouze 2 kategorie (jednu pro původní síť a jednu pro adaptační síť), tak jsem se rozhodl jako svoji chybovou funkci pro diskriminátor použít binární křížovou entropii, která funguje obdobně jako křížová entropie, ale rozhoduje o rozřazení do 2 kategorií, namísto N kategorií jako u křížové entropie. Výsledná chyba binární křížové entropie se pak vypočítá podle vzorců:

$$Chyba = -\frac{1}{N}\sum_{n=1}^{N} y_n \times log(x_n) + (1 - y_n) \times log(1 - x_n) \tag{3.3}$$

ve kterých x je vektor aktivací, y je vektor očekávaných hodnot a N je velikost dávky.

3.5 Výsledná chyba při adaptaci

Každá chybová funkce v mé práci používá ke své minimalizaci princip algoritmu stochastického gradientního sestupu, nicméně v jeho vylepšené podobě, kterou je optimalizační algoritmus Adam [4]. Tento algoritmus funguje na principu stochastického gradientního sestupu, ale navíc dokáže dynamicky měnit parametr rychlost učení (angl. learning rate) pro každou váhu sítě zvlášť. Rychlost učení je parametr, kterým se reguluje krok u stochastického gradientního sestupu. Moc velká hodnota tohoto parametru může zapříčinit nenalezení minima funkce a moc malá hodnota příliš dlouhou dobu k nalezení minima. V procesu adaptace se vyskytují 2 chybové funkce, jedna u diskriminátoru a druhá je CTC funkce, která trénuje adaptační síť. Aby se při zpětné propagaci do vah sítě promítly gradienty z obou chyb, tak by při nejsnazším řešení bylo nutno použít 2 optimalizátory, které v mém případě implementují algoritmus Adam. Jeden pro gradient chyby z diskriminátoru a druhý pro gradient chyby z CTC. Ve chvíli, kdy bych použil 2 optimalizátory, tak by se musel spočítat zpětný průchod diskriminátorem 2x, což by způsobovalo vyšší časovou náročnost celého procesu. Proto jsem ve svojí práci vyřešil zpětnou propagaci za pomoci pouze jednoho optimalizátoru a to tak, že výsledná chyba celého adaptačního modelu se vypočte podle vzorce:

$$chyba = chyba\_ctc + chyba\_diskriminator \tag{3.4}$$

kde chyba je výsledná chyba sítě, chyba_ctc je výsledná chyba chybové funkce CTC, chyba_diskriminator je chyba vypočítaná BCELoss chybovou funkcí. Při zpětné propagaci se gradient z diskriminátoru násobí s parametrem lambda v jednotce otáčení gradientu a tím se ovlivňuje velikost regularizace ze strany diskriminátoru. Díky tomuto řešení se v sítí vyskytuje pouze jedna výsledná chyba a díky tomu používám pouze jeden optimalizátor. To taky znamená, že zpětný průchod diskriminátoru se vypočítá pouze jednou.

Kapitola 4

Dataset

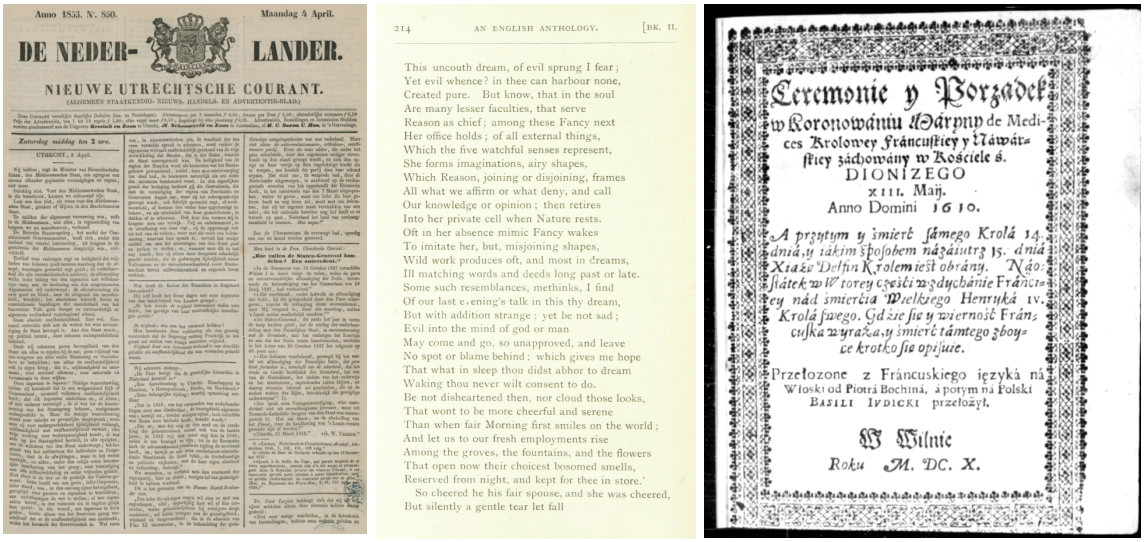
Pro správné natrénování neuronových sítí je mimo jiné potřeba mít velké množství trénovacích dat. Množině těchto dat se říká dataset. Data se před vstupem do neuronové sítě anotují. Anotování je proces označování dat, respektive označování toho, co má síť na základě vstupní informace vygenerovat. V rámci vytváření OCR je anotování stránek textu proces, kdy se na stránce vstupního dokumentu označí oblasti, kde se nachází text, a pouze z těchto oblastí se získávají ground truth přepisy tohoto textu z obrázku, které pak slouží jako reference pro chybovou funkci. V rámci mé práce pracuji s dvěma typy datesetů, z nichž jeden je dataset, který obsahuje stránky tištěných dokumentů a více ho popisuji v části 4.1 a druhý je dataset ručně psaných dokumentů, který jsem popsal v podkapitole 4.2.

OCR jako takové, když dostane na svůj vstup stránku textu, tak mimo rozpoznání jednotlivých písmen potřebuje zjistit kde na stránce je text a co na stránce vůbec text je a co není. Před tím, než je vstupní dokument předán neuronové sítí, která umí rozpoznávat text, tak je třeba detekovat řádky v dokumentu. Pro toto je vytvořená další síť, která je trénovaná právě pro tuto detekci.

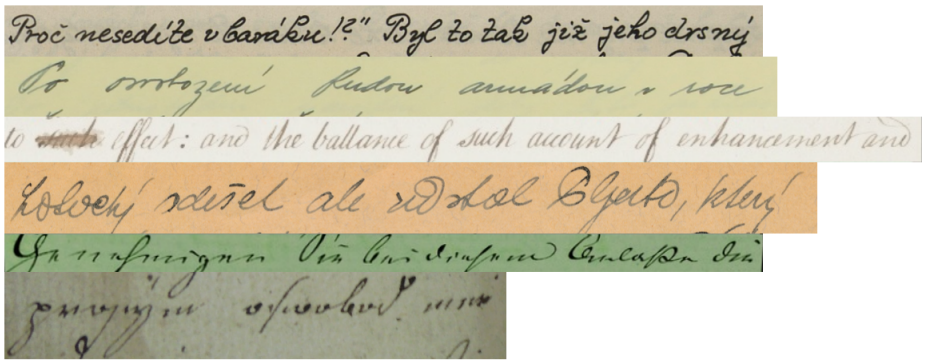
4.1 Dataset IMPACT

IMPACT [14] je projekt financovaný EU, který si klade za cíl zlepšit dostupnost historických textů. Aby síť měla co nejlepší rozpoznávací schopnost na cílových datech, tak je potřeba, aby trénovací data byla různorodá, což dataset IMPACT splňuje. Dataset obsahuje 600 000 fotek stránek dokumentů z 9 evropských jazyků. Jsou jimi bulharština, čeština, nizozemština, angličtina, francouzština, polština, španělština, slovinština a němčina. Dataset se skládá z velkého množství různých knih, novin, žurnálů a dokumentů napsaných na psacím stroji. Všechny tyto druhy publikací pocházejí z 16. až 20. století. Ukázky textů, které jsou součástí datasetu IMPACT jsou k vidění na obrázku 4.1. Dataset IMPACT obsahuje dokumenty se složitým strukturováním stránek, dvoustránkové dokumenty, dokumenty s tabulkami a různou grafikou na stránkách. Dokumenty jsou také různé kvality a mají různé vlastnosti, například barevnou hloubku a rozlišení.

Z 600 000 stran dokumentů má asi 45 000 stran k dispozici textové anotace. Projekt dal požadavek každé z 9 evropských knihoven na naskenování 50 000 reprezentativních vzorků dokumentů z jejich repozitářů. Na vytváření ground truth anotací se podílelo několik partnerů projektu a také samotné knihovny. Aby anotace byly standardizovány, tak partneři a knihovny společně sepsali pravidla a podrobnou specifikaci. Při takto velkém počtu a různorodosti dokumentů, byl anotátorům předán k dispozici poloautomatizovaný



Obrázek 4.1: Ukázka obrázků z datasetu IMPACT, na kterých je vidět velká diverzita co se kvality, jazyků a stylů týče.



Obrázek 4.2: Ukázka řádků z datasetu ručně psaného písma. Na těchto řádcích je dobře vidět různorodost ručně psaných textů, které jsou součástí datasetu.

software na vytváření ground truth anotovací Aletheia [3]. Tento software mohli v rámci vytváření anotací používat. Výstup OCR Aletheia bylo poté potřeba vizuálně zkontrolovat, případně doupravit nebo opravit chyby. Samotná kvalita anotací při takto rozsáhlém datasetu prakticky nemůže být dokonalá a proto si autoři stanovili jako cílovou přesnost 99.95 %. Takovýmto způsobem získané ground truth anotace jsou uloženy v souboru ve formatu XML.

Samotný dataset IMPACT v sobě neobsahuje informace o pozicích řádků, takže pro potřeby trénování na něm, byly tyto detekce provedeny v rámci projektu PERO Ing. Oldřichem Kodymem [10].

Při vytváření datové sady, která obsahem vychází z datasetu IMPACT, jsem používal skript impact_parser.py vytvořený Ing. Janem Kohútem. Tento skript přijímá jako parametry různé vlastnosti dokumentů z datasetu, jakými jsou například jazyk, ve kterém je dokument napsán, jeho kvalitu, jeho typ atd. Na základě těchto parametrů skript vybere odpovídající dokumenty a jejich řádky pak umístí do výstupního dokumentu, který pak může sloužit jako dokument s trénovací sadou.

	Bentham	READ	Brno_HWR	Dopisy	Kroniky	PERO server
Počet řádků	11 473	190 505	4 406	90 091	25 663	211 272

Tabulka 4.1: Počty řádků v jednotlivých datasetech, které tvoří celý dataset ručně psaných textů.

4.2 Dataset ručně psaných textů

Další datová sada, kterou v rámci své práce využívám je dataset ručně psaných textů, který se samotný skládá z několika menších datasetů ručně psaných textů a dat, které se v rámci projektu PERO podařilo shromáždit. Díky skladbě datasetu, který se sestává z několika menších datasetů, je v něm velká diverzita dat. Většina dokumentů obsažených v tomto datasetu je v jednom ze tří jazyků, jsou jimi čestina, němčina a angličtina. Mimo tyto jazyky ale dataset obsahuje i menší množství jiných jazyků. Datasety, z kterých je sestaven dataset ručně psaných textů, jsou dataset Bentham [6][19], dataset Brno_HWR, dataset READ [20], české dopisy [8], kroniky a ostatní dokumenty získané v rámci Projektu PERO.

Dataset Bentham vznikl v rámci projektu tranScriptorium[1], který pracuje na zpřístupnění kulturních zdrojů v podobě historických dokumentů. Samotný dataset Bentham byl pojmenován po anglickém filosofovi a reformátorovi Jeremym Benthamovi, jehož vlastnoruční zápisky tvoří tento dataset. V rámci projektu bylo dobrovolníky přepsáno více než 6000 dokumentů od tohoto autora, nicméně já mám k dispozici množství takto přepsaných dokumentů a stran menší a to přesně 11 473 řádků textu.

Dataset READ vznikl v rámci soutěže zaměřené na rozpoznávání textů Handwritten Text Recognition (HTR) [17]. Většina dat v datasetu pochází z dopisů německého politika 19. století, které jsou napsané německy. Nicméně v datasetu se nacházejí i jiné cizojazyčné ručně psané texty od několika jiných autorů.

Brno_HWR je dataset, který je vytvářený v rámci projektu PERO. Tyto přepisy mají velkou diverzitu, co se počtu pisatelů týče. K tvorbě tohoto datasetu může přispět každý tak, že si stáhne náhodnou stranu textu z adresy[2] a přepíše ji vlastnoručně. Pokud je tento přepsaný text přepsán v pořádku, tak se stává novou částí Brno_HWR. Dalšími datasety, které jsou součástí datasetu ručně psaných textů, jsou české dopisy a kroniky. Ostatní dokumenty a jejich přepisy v rámci datasetu ručně psaných dokumentů jsou zejména v češtině a jsou získané z PERO serveru. Celkový počet anotovaných řádků, které mám k dispozici, je vidět v tabulce 4.1.

Díky skladbě z mnoha různých menších a i cizojazyčných datasetů a také díky tomu, že k přepisování těchto dokumentů pomohlo velké množství dobrovolných přepisovatelů, je k dispozici velké množství stylově odlišných přepisů. Díky těmto vlastnostem dataset splňuje požadavky, které na něj v rámci adaptace kladu. Ukázka několika řádků z datasetu je vidět na obrázku 4.2.

[1] Projekt tranScriptorium http://transcriptorium.eu/

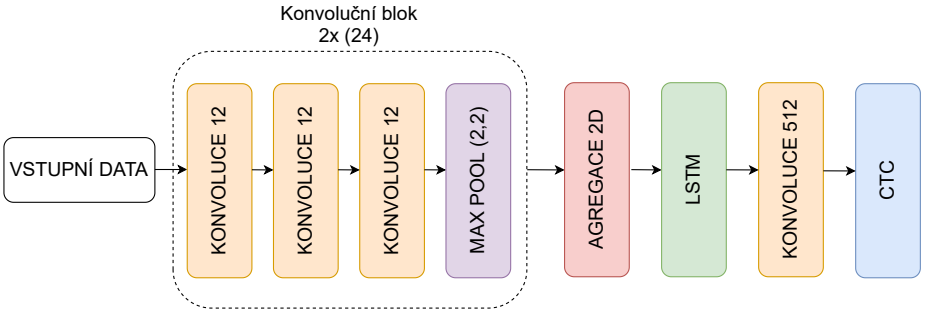
[2] Projekt PERO, ručne psaný dataset https://pero.fit.vutbr.cz/handwritten_dataset

Kapitola 5

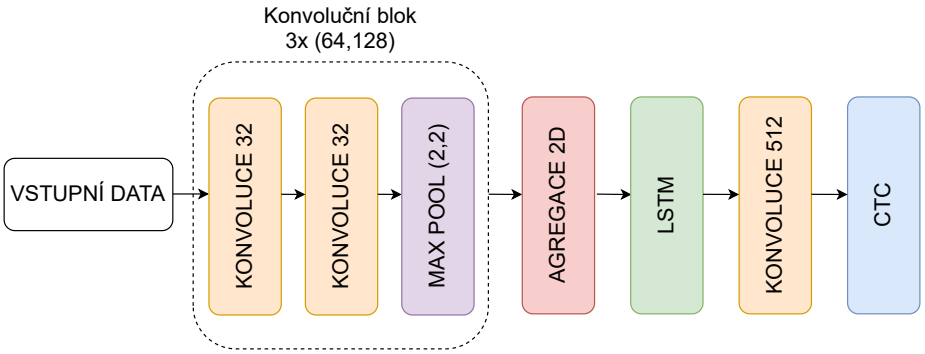
Experimenty

V této kapitole se zabývám popisem experimentů s vytvořeným modelem podle předlohy z článku Adversarial Speaker Adaptation, kterou jsem popsal v kapitole zabývající se metodami adaptace neuronových sítí 3. Experimenty popsané v této kapitole jsou prováděny nad datasetem knihovních textů IMPACT a nad datasetem ručně psaných textů. Hlavním smyslem experimentů je ověřit přínos metody adaptace neuronových sítí z článku Adversarial Speaker Adaptation a zjistit, jak se bude soustava sítí popsaná v článku chovat při různých velikostech regularizačního parametru lambda, při různých rozděleních původní a adaptační sítě a při různých počtech adaptačních řádků.

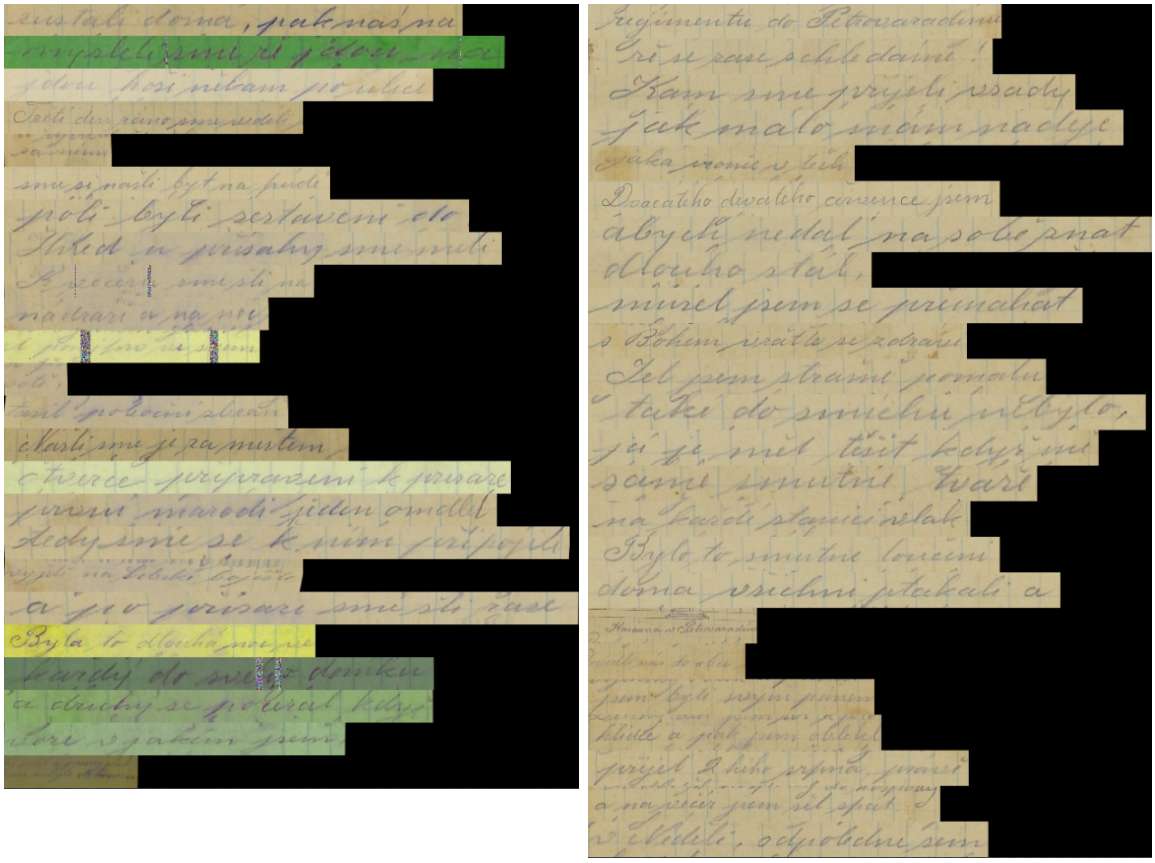
Architektura sítí Při experimentech jsem použil dva typy architektur v závislosti na datasetu. Pro natrénování původní sítě na datasetu IMPACT jsem použil jednodušší ze dvou architektur, která je zobrazená na obrázku 5.1. Architektura sítě pro experimentování nad datasetem ručně psaných textů je zobrazena na obrázku 5.2. Tyto architektury se liší v počtu konvolučních bloků a počtem filtrů použitých v konvolucích. Architekura 5.1 je dostatečná pro tištěná data, která jsou obsahem datasetu IMPACT, nicméně takováto architektura není dostatečná pro komplikovanější data, které jsou v datasetu ručně psaných textů. V mnou používaných sítích jsou, oproti nákresu, ve skutečnosti LSTM vrstvy čtyři, z nichž každá přijímá jiné kontextové okno. Každá z těchto LSTM vrstev je obousměrná. Konvoluce 512 je v sítí proto, že ve znakové sadě je 512 znaků, které síť dokáže rozpoznávat. Architektura obou sítí koncepčně vychází z článku [1].



Obrázek 5.1: Architektura používaná při experimentech adaptace na datasetu IMPACT. Konvoluční bloky jsou 2 za sebou, s tím, že druhý rozšíří kanálovou dimenzi matice na 24.



Obrázek 5.2: Architektura používaná při experimentech adaptace na datasetu ručně psaných dokumentů. Konvoluční bloky jsou 3 za sebou, s tím, že druhý rozšíří kanálovou dimenzi matice na 64 a třetí na 128.



Obrázek 5.3: Ukázka augmentovaných dat a dat neaugmentovaných. Na levé části obrázku jsou vidět augmentované výřezy řádků. V pravé části jsou k vidění neaugmentované výřezy řádků pocházející ze stejného deníku jako data na levé části obrázku. Na augmentovaných datech je zejména vidět změna barvy, jasu a maskování.

Techniky používané při adaptaci. V rámci adaptačních experimentů používám datový augmentátor CGM, který do vstupního obrázku přidává náhodné maskování, upravuje velikost obrázků, sklon obrázků atd. Krom těchto úprav také upravuje světlost, saturaci a jiné vlastnosti. Jiné regularizační techniky, krom diskriminační sítě, nepoužívám a to zejména z důvodu, že testování všech kombinací by zabralo velké množství času. Augmentaci používám z toho důvodu, že v praxi při trénování na malé datové sadě se bude pravděpodobně vždy používat. Ukázka augmentovaných a neaugmentovaných dat je na obrázku 5.3. Jako optimalizátor používám optimalizátor Adam.

	Agregační vrstva		Po prvním bloku	
	μ	σ	μ	μ
λ=0	6,43	0,53	6,43	0,53
λ=1	**6,09**	**0,30**	6,29	0,29
λ=2	6,18	0,38	**6,18**	**0,15**
λ=4	6,21	0,46	6,33	0,39
λ=8	6,59	0,45	6,57	0,53

Tabulka 5.1: Výsledné zprůměrované hodnoty chybovosti (ve sloupci označeném μ v %) šesti pokusů pro každou hodnotu parametru lambda v závislosti na agregační vrstvě a na aktivacích po prvním síťovém bloku. Součástí tabulky je také sloupec označený jako σ, ve kterém jsou směrodatné odchylky pokusů.

5.1 Průběh experimentů

V této části textu používám termíny trénovací, testovací, adaptační a cílová sada. Trénovací a testovací sada používám v kontextu trénování původní sítě. Adaptační a cílová sada používám ve smyslu trénovací a testovací sady adaptační sítě.

Experimenty, které jsem prováděl, konceptuálně vychází z experimentů popsaných v článku Adversarial Speaker Adaptation. Z celkové sady dat, na které jsem chtěl neuronovou síť adaptovat, jsem pro každý pokus vyčlenil 200, 100, 50 a 20 řádků do adaptační sady. Takovéto počty řádků jsem volil, protože jsou realisticky získatelné z adaptačních textů. Zbytek z celkového počtu adaptačních dat jsou cílová data, na kterých pak budu ověřovat úspěšnost adaptace. Na těchto počtech řádků jsem prováděl experimenty s rozdílnými hodnotami parametrů lambda. Experiment s každou hodnotou parametru lambda na stejném počtu adaptačních řádků jsem opakoval několikrát a to z toho důvodu, že soustava sítí pro adaptaci má sice váhy v původní a adaptační sítí inicializované, ale samotný diskriminátor své váhy inicializované nemá a při spuštění adaptace se pak do vah diskriminátoru přiřadí náhodné hodnoty. V kombinaci s tím, že v rámci experimentů je vstup augmentován, dochází k tomu, že výsledek adaptace, respektive její úspěšnost na cílových datech, je proměnná a to i přes to, že se pouští nad stejnými adaptačními daty. Z tohoto důvodu by samotný výsledek jednoho pokusu s každou lambdou nebyl reprezentativní. Jako reprezentativní počet opakování jsem zvolil šest pokusů, protože větší počet opakování již nezpůsoboval větší odchylky ve směrodatné odchylce a ani v průměrech jednotlivých opakování.

Lambdy volené pro regularizaci ze strany diskriminátoru jsem volil hodnotově podobné jako v článku, nicméně následně jsem zkoušel i vyšší hodnoty parametru lambda a to zejména proto, že v článku nebylo uvedeno, proč volili zrovna takovéto hodnoty. Autory zvolené hodnoty parametru lambda byly 1,3 a 5. Po pokusech, které jsem prováděl, jsem zjistil, že tyto hodnoty jsou vhodné z toho důvodu, že mění gradient z CTC části sítě několika procentní změnou. Jedná se avšak o změnu gradientu, která je velikostně přijatelná. Se zvyšující se hodnotou parametru lambda roste procentuální změna gradientu a tím i efekt regularizace od diskriminátoru, nicméně to se děje do zvýšení parametru, po kterém se trénování začne kazit a chyba na datech začne příliš oscilovat. A to jak na trénovacích, tak na cílových.

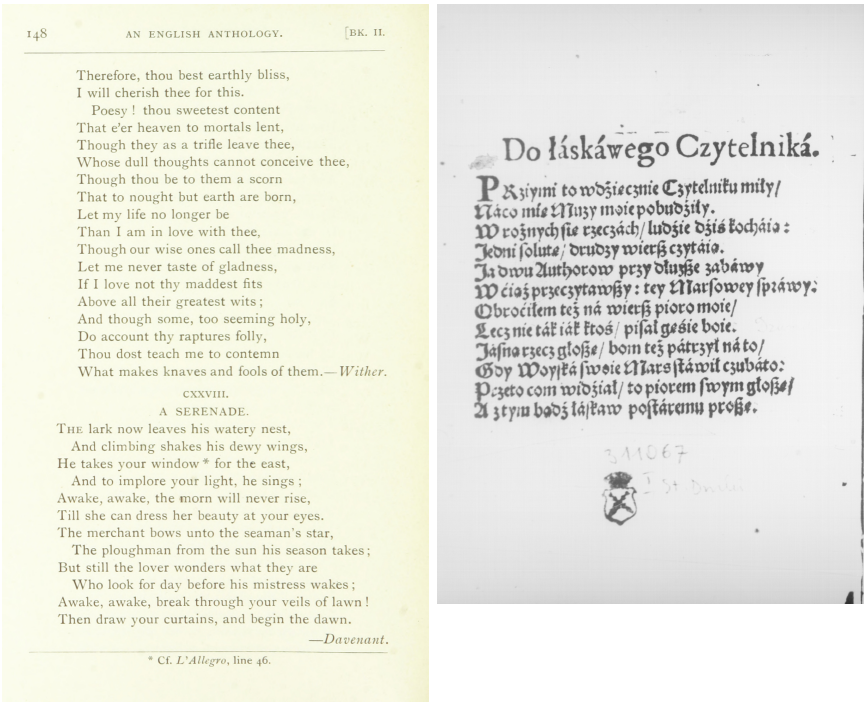
V rámci experimentů mě nejvíce zajímá úspěšnost na cílové sadě. To je z toho důvodu, že v rámci adaptace potřebuji, aby se síť natrénovala na trénovacích datech, která jsou stylově podobná jako zbytek adaptačního textu a to mělo za následek zlepšení schopnosti

rozpoznávání na zbylé části textu, na které se síť neměla možnost učit. Tato zbylá data jsou uložena v cílové sadě.

Samotný postup experimentů byl takový, že jsem pustil adaptaci šestkrát pro lambdy 0, 1, 2, 4 a 8 nejprve pro 200 řádků a pak jsem postupně odebíral řádky z adaptačního datasetu na počty 100, 50 a 20. Řádky v cílové sadě zůstávaly neměnné. Pro měření chyby na cílové sadě jsem se rozhodl experiment pro každou lambdu spustit na předem stanoveném počtu iterací. Inspekcí výsledků jsem se rozhodl vyčíst chybu na cílové sadě při iteraci, kdy chyba na adaptační sadě již oscilovala a nijak výrazně se nezmenšovala. Toto řešení jsem zvolil, protože takto si představuji měření v praxi. V ideálním případě bych výslednou hodnotu chyby četl přímo jako nejnižší hodnotu chyby na cílových datech, jenže toto v praxi nelze. V praxi adaptace bude probíhat tak, že se naanotuje několik řádků z adaptačního textu, což může být okolo mnou zvolených 200, 100, 50, nebo 20, ale zbytek textu se neanotuje. Tím pádem není možnost ověřit úspěšnost na zbytku adaptačního textu, což jsou cílová data. Toto je důvod proč musím číst chybu v závislosti na adaptační sadě a ne cílové.

V článku se krom parametru lambda vyskytuje ještě jeden parametr a tím je index vrstvy sítě, po které získávám aktivaci, kterou předávám diskriminátoru. Abych zjistil, kterou aktivace vrstev sítě je nejvýhodnější předávat diskriminátoru, tak jsem provedl nad datasetem IMPACT pokus s aktivacemi různých vrstev. Výsledky tohoto experimentu ukázaly, že předávat aktivace hlouběji v sítí, po agregační vrstvě, má za následek jemně lepší úspěšnost na cílových datech. Výsledky experimentu s aktivacemi různých vrstev na datasetu IMPACT jsou zobrazeny v tabulce 5.1. Díky těmto zjištěním jsem se rozhodl pro zbytek experimentů použít, jako vstup diskriminátoru, aktivace agregační vrstvy. Předávání diskriminátoru aktivace hlouběji v sítí má za následek také regularizaci větší části sítě.

Při experimentování nad adaptačním modelem jsem se snažil hodnotami lambda většími než 0 dostat k vyšším úspěšnostem než s lambdou 0 (s lambdou 0 je vypnutý efekt diskriminátoru, respektive se nulují gradienty chyby přicházející z diskriminátoru), protože při adaptaci s lambdou 0 se jedná o vylepšenou variantu naivního řešení adaptace popsané v kapitole o metodách adaptace 3. Všechny grafy v následující části textu jsou vytvořené pomocí zprůměrování šesti běhů pro každou iteraci. V další části používám chybu diskriminátoru a chybu adaptační sítě. Chyba diskriminátoru vyjadřuje to, jestli se diskriminátor v predikci zmýlil a jak moc. Pro adaptační síť vyjadřuje chyba procento chybně rozpoznaných znaků.



148 AN ENGLISH ANTHOLOGY. [BK. II.

Therefore, thou best earthly bliss,
I will cherish thee for this.
Poesy! thou sweetest content
That e'er heaven to mortals lent,
Though they as a trifle leave thee,
Whose dull thoughts cannot conceive thee,
Though thou be to them a scorn
That to nought but earth are born,
Let my life no longer be
Than I am in love with thee,
Though our wise ones call thee madness,
Let me never taste of gladness,
If I love not thy maddest fits
Above all their greatest wits;
And though some, too seeming holy,
Do account thy raptures folly,
Thou dost teach me to contemn
What makes knaves and fools of them.—*Wither.*

CXXVIII.

A SERENADE.

THE lark now leaves his watery nest,
And climbing shakes his dewy wings,
He takes your window * for the east,
And to implore your light, he sings;
Awake, awake, the morn will never rise,
Till she can dress her beauty at your eyes.
The merchant bows unto the seaman's star,
The ploughman from the sun his season takes;
But still the lover wonders what they are
Who look for day before his mistress wakes;
Awake, awake, break through your veils of lawn!
Then draw your curtains, and begin the dawn.

—*Davenant.*

* Cf. *L'Allegro*, line 46.

Obrázek 5.4: Ukázka trénovacích dat a dat adaptačních. Na levém obrázku je ukázka stránky dokumentu, který je v anglickém soudobém stylu a je podobná datům, na kterých se učila původní síť. Na pravém obrázku je adaptační text polského historického dokumentu.

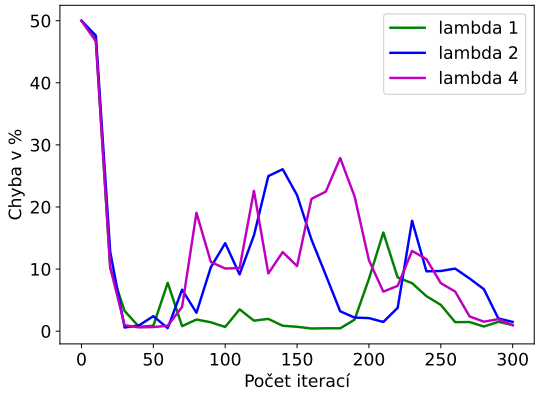
5.2 Experimenty nad datasetem IMPACT

Nad tímto datasetem jsem prováděl experimenty s natrénovanou sítí, jejíž architektura je znázorněna na obrázku 5.1. Experimenty, které jsem se nad tímto datasetem rozhodl realizovat jsou výběr vrstvy sítě, po které předávám aktivace do diskriminační sítě a adaptace na polské historické písmo s tím, že původní síť bude natrénovaná na knihách a takzvaných dokumentech typu "legal", což jsou právní dokumenty. Do procesu učení původní sítě jsem zařadil knihy a právní dokumenty (ve všech jazycích krom polštiny), které odpovídají stylu soudobého textu (občas se zde vyskytují i kaligrafické prvky) a to proto, že jsem chtěl co největší možný trénovací dataset a tyto skupiny jsou vzhledově poměrně podobné. Obecně se dá říct, že soudobé dokumenty v datasetu IMPACT jsou podobné, ale oproti historickým dokumentům se liší markantněji. Ukázka původních trénovacích a adaptačních dat je na obrázku 5.4. Ze získaných řádků jsem navíc vyčlenil 16 000 řádků do testovací sady a trénovací sada obsahovala přes 800 000 řádků textu. S takto vytvořeným datasetem a architekturou sítě znázorněnou na obrázku 5.1 se mi podařilo původní síť natrénovat na chybu nižší než 1 % jak na trénovací, tak na testovací datové sadě.

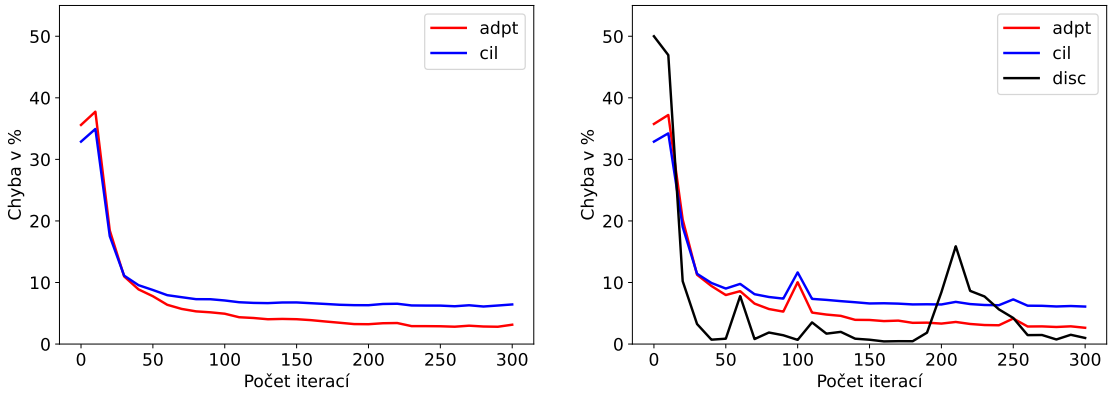
Počty řádků v souboru								
	20		50		100		200	
Chyba	32,88 %							
	μ	σ	μ	σ	μ	σ	μ	σ
λ=0	9,32	0,29	7,28	0,49	6,43	0,54	**5,30**	0,14
λ=1	9,40	0,73	7,28	**0,25**	**6,09**	0,16	5,31	0,29
λ=2	10,08	1,29	7,18	0,29	6,18	**0,15**	5,38	**0,13**
λ=4	9,51	0,49	7,49	0,55	6,21	0,16	5,54	0,70
λ=8	**9,20**	**0,25**	**7,14**	0,39	6,59	0,53	5,60	0,46

Tabulka 5.2: Tabulka výsledků adaptace. Sloupec výsledků pro každý počet řádků je rozdělen na dvě části. V levé části je chyba na cílových datech po 300 iteracích (označená jako μ v %) a v pravé části je vypočítána směrodatná odchylka výsledků spuštění šesti opakování pro každou hodnotu lambda, označená jako (označená jako σ).

Adaptace na datasetu IMPACT. Pro mou natrénovanou původní síť s chybou pod 1 % na testovací sadě, je chyba na adaptační sadě polského písma 36,5 % s tím, že tato chyba se pohybuje při každém ze šesti spuštění v rozmezí 0,9 %, což je závislost chyby na proměnných podmínkách sítě, které jsem popsal výše. Chyba na cílových datech je 32,88 %. Proces adaptace pouštím na pevně daném počtu trénovacích iterací, které jsem si zvolil a to jako 300. Volba tohoto počtu byla dána tím, že síť už po dosažení počtu iterací blízko 300. iteraci, nevykazovala zlepšení na adaptační sadě. Hodnoty úspěšnosti, při pozorování každých 20 iterací, začaly kolem iterace 260 jemně oscilovat kolem určité procentuální chyby na adaptační sadě a další zlepšení nevypadalo jako pravděpodobné, čemuž odpovídaly i chyby v přepisech, které jsem vizuálně kontroloval.



Obrázek 5.5: Průběhy chybovosti diskriminátoru pro lambdy 1, 2 a 4.



Obrázek 5.6: Na levém obrázku je vidět průběh učení na cílových (cil) a adaptačních (adpt) datech pro lambdu 0. Na pravém obrázku je vidět chyba na cílové a adaptační sadě pro lambdu 1 a taky průběh chyby na diskriminátoru (disc).

Z výsledků v tabulce 5.2 je vidět, že chyba při lambdě 1, 2 a 4 (myšleno s hodnotou 1, 2 a 4 parametru lambda, tento zápis budu používat v textu dále) je při každém počtu řádků velmi blízko té s lambdou 0, až na případ počtu řádků 100, ve kterém lambda 1, 2 a 4 významně vylepšily rozpoznávání sítě na cílových datech. Hodnota 8 parametru lambda, při 100 řádcích, je již příliš velká a úspěšnost při použití této velikosti parametru lambda je nižší než při použití lambdy 0. Krom toho, že při použití lambd 1, 2 a 4 bylo docíleno lepší úspěšnosti na cílových datech, tak taky směrodatné odchylky při šesti opakováních experimentu s lambdou 1, 2 a 4 jsou menší než směrodatná odchylka s lambdou 0. Díky těmto výsledkům můžu konstatovat, že použití některé z lambdy 1, 2 a 4 mělo se souborem, ve kterém je 100 adaptačních řádků, lepší výsledek, než použití naivní metody adaptace se zapnutou augmentací jejíž výsledek je 6,43 %. U lambdy 1 se jedná o zlepšení 5,58 %, u lambdy 2 je to 4,04 % a u lambdy 4 3,5 % vůči lambdě 0. Vzhledem k zjištění, že hodnoty parametru lambda 1, 2 a 4 překonají lambdu 0, jsem se rozhodl dál prozkoumat průběhy adaptace s těmito hodnotami parametru lambda.

Průběhy chybovostí diskriminátorů při použití lambd 1, 2 a 4 jsou vidět na obrázku 5.5. Z obrázku je vidět, že diskriminátoru více osciluje při pozdějších iteracích. To je způsobeno tím, že gradient chyby od CTC je se zvětšujícím se počtem iterací menší a gradient od

diskriminátoru začne mít vyšší vliv na úpravu vah regularizované části sítě. Na obrázku 5.6 je vidět rozdíl mezi lambdou 0 a lambdou 1. Díky použití regularizátoru v podobě diskriminátoru je vidět, že nedochází k takovému přeučení na adaptačních datech, což má za následek zlepšení na cílových datech. Neschopnost sítě se přeučit na adaptačních datech je dána tím, že gradient diskriminátoru se zvětšuje vůči gradientu CTC, protože se zvětšujícím se počtem iterací klesá chybovost adaptační sítě a tím roste i regularizační efekt na adaptační síť, dochází k očekávané oscilaci na diskriminátoru. Jak je vidět z obrázku 5.6 pro lambdu 1, diskriminátoru velmi rychle klesne chybovost v prvních 50 iteracích z toho důvodu, že CTC gradienty významně změní vnitřní váhy adaptační sítě, což má za následek větší rozdíly ve vstupech do diskriminátoru a tedy pro diskriminátor je jednodušší rozpoznat rozdíl v přijatých datech. Jakmile však po 50. iteraci klesne gradient od CTC, tak gradient diskriminátoru bude mít vyšší vliv na váhy adaptační sítě a to způsobí opětovné drobné zhoršení úspěšnosti na adaptační i cílové sadě. Díky takovéto oscilaci mezi velikostmi gradientů je zabraňováno přetrénování a tím i umožnění zlepšení na cílových datech. Ze samotného obrázku pro lambdu 0 je také vidět ke konci velmi mírné zhoršení úspěšnosti na cílových datech, což je příznak přetrénování. Tato skutečnost ovšem na obrázku s lambdou 1 vidět není.

Shrnutí experimentu nad datasetem IMPACT. V rámci tohoto experimentu jsem zjistil, že je lepší pro vstup do diskriminátoru předávat aktivace agregační vrstvy, než aktivace za prvním blokem, i když rozdíl není příliš velký. Dále se díky experimentu ukázalo, že použití lambdy 1, 2 a 4 má na výsledky úspěšnosti na cílových datech při počtech řádků 20, 50 a 200 velmi podobný efekt jako použití samotné lambdy 0, ale u počtu řádků 100 je výsledek pro tyto lambdy podstatně lepší, než použití lambdy 0 a to znamená, že se pro adaptaci na podobných datech, jaké jsou v datasetu IMPACT, vyplatí použít lambdu 1, 2 a 4, jelikož jsou výsledky testů s nimi velmi podobné výsledkům testů s lambdou 0, avšak u případu 100 řádků jsou mnohem lepší. Z experimentů také vyšlo najevo, že očekávané oscilační chování diskriminátoru má pozitivní efekt na úspěšnost na cílových datech.

Obrázek 5.7: Na levém snímku jsou vidět data, na kterých se původní síť trénovala. Prostřední obrázek je ukázka dat z deníku, na kterém měla původní síť chybu 1,82 %. Na pravém obrázku je deník, s nímž měla síť problém na 10,82 % znacích.

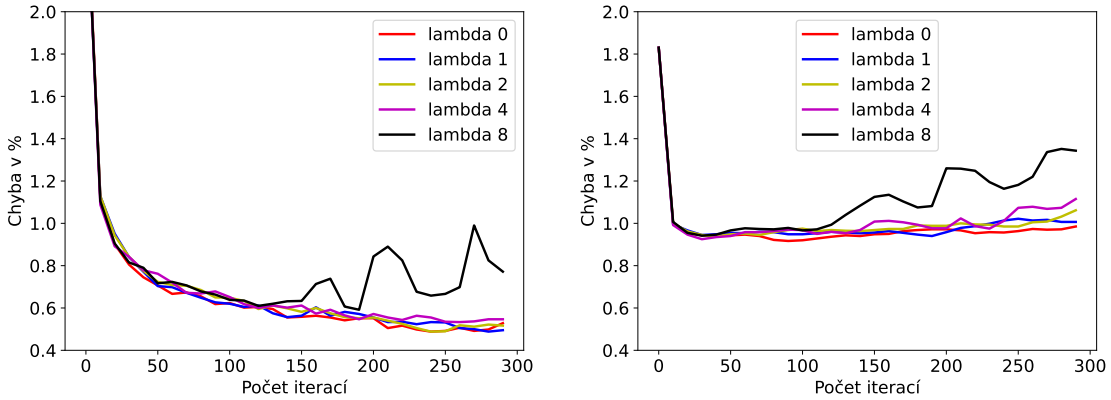
5.3 Experimenty nad datasetem ručně psaných textů

Při experimentech s datasetem ručně psaných textů jsem se zaměřil na adaptaci na česky psané vojenské deníky. V rámci tohoto datasetu je několik starých vojenských deníků, které jsou vždy napsané jedním člověkem. Toto odpovídá scénáři, kvůli kterému řeším i svoji práci. Jedná se totiž přímo o adaptaci na jednoho konkrétního pisatele.

Původní síť byla natrénovaná na celkové množině dat z datasetu ručně psaných textů, krom devíti vojenských deníků. Za použití sítě, jejíž architektura je na obrázku 5.2, se mi podařilo dosáhnout chybovosti na 2 denících, na které se chci adaptovat, 1,82 % chyby na prvním a na druhém vojenském deníku chyby 10,82 % (deníky byly součástí testovací sady). Chyba na celé testovací sadě sítě byla 7 %. Ukázky dat, na kterých se původní síť trénovala, a data, na kterých jsem prováděl adaptace, jsou vidět na obrázku 5.7. Jako vrstvu sítě, po které předávám aktivace diskriminátoru, jsem opět zvolil vrstvu agregace. I u tohoto datasetu jsem totiž zkoušel předávat aktivace po prvním bloku sítě, nicméně mělo to velmi podobný efekt jako u datasetu IMPACT a ten je, že mezi těmito aktivacemi není velký rozdíl a pro aktivace po agregační vrstvě jsem dostal lepší výsledky.

Počty řádků v souboru								
	20		50		100		200	
Chyba	1,82 %							
	μ	σ	μ	σ	μ	σ	μ	σ
λ=0	**1,38**	**0,04**	**0,91**	**0,02**	**0,91**	**0,02**	**0,77**	0,02
λ=1	1,40	**0,04**	0,96	**0,02**	0,95	**0,02**	0,79	0,03
λ=2	1,45	0,07	0,93	**0,02**	0,97	**0,02**	0,79	0,13
λ=4	1,48	0,10	0,97	0,04	0,97	0,03	0,80	**0,01**
λ=8	1,57	0,10	0,98	**0,02**	0,97	**0,02**	0,80	**0,01**

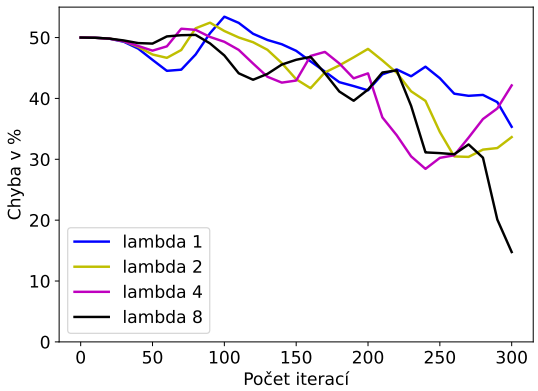
Tabulka 5.3: Tabulka výsledků adaptace na deníku s chybou na začátku adaptace 1,82 %. Sloupec výsledků pro každý počet řádků je rozdělen na dvě části. V levé části (označené jako μ v %) je chyba na cílových datech po 100 iteracích a v pravé části je vypočítána směrodatná odchylka (σ) výsledků spuštění šesti opakování pro každou hodnotu lambda. Nejlepší výsledky v jednotlivých sloupcích jsou označeny tučně



Obrázek 5.8: Na levém obrázku je ukázka chybovosti na adaptačních datech. Na pravém pak chybovost na cílových datech.

Adaptace na deník s chybou 1,82 % Výsledky adaptace pro tento deník jsou vidět v tabulce 5.3. Z těchto výsledků je vidět, že s žádnou hodnotou parametru lambda vyšší než 0 jsem nebyl schopný získat na cílové sadě vyšší úspěšnost, než při použití lambdy 0. Zajímavé taky je, že rozdíl v úspěšnosti mezi 50 a 100 řádky je velmi malý a dokonce u jedné hodnoty parametru lambda je výsledek lepší pro 50 řádků, než pro tu samou lambdu při vyšším počtu řádků. Toto však je pravděpodobně způsobeno informacemi, které je síť schopna se z řádků naučit. Je pravděpodobné, že soubor se 100 řádky obsahuje mnoho řádků, které mají nízký informativní charakter pro síť a nebo se tyto informace dokáže síť naučit z 50 řádků, které pak zůstanou pro adaptační soubor s 50 řádky.

Průběh adaptace ukazuje poměrně významné problémy. Na obrázcích 5.8 je vidět, že chyba na adaptačních datech se prvně rapidně zmenšuje, ale kolem iterace 50 se začne snižování chyby zpomalovat. Po prvotním prudkém snížení chyby na adaptačních datech, začne růst chyba na cílových datech. Dochází tady k podobnému efektu, jaký má přetrénování. Problémem u tohoto deníku je, že CTC v prvních 25 iteracích sníží chybu na adaptačních i cílových datech, nicméně to, jakým způsobem upraví váhy adaptační sítě, není dostatečné,



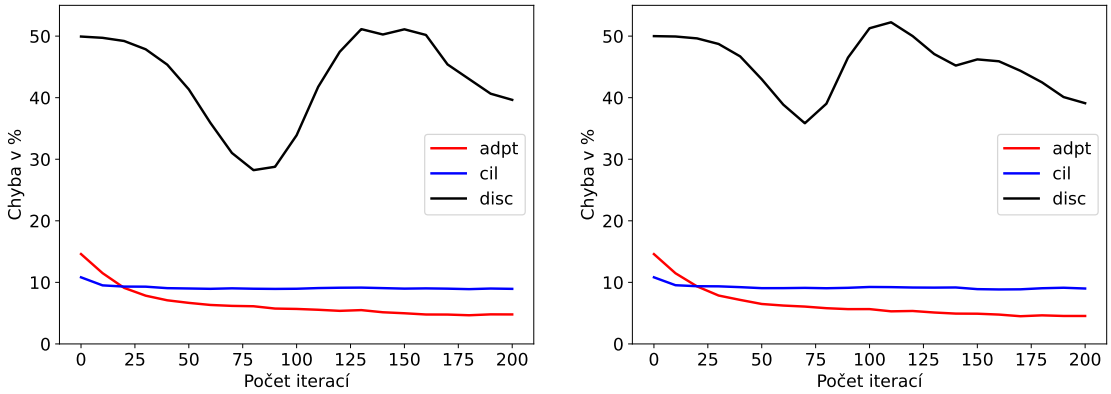
Obrázek 5.9: Průběh chybovosti na diskriminační síti.

aby diskriminátor dokázal rozlišit vstupy. Díky tomu, že diskriminátor není schopný rozeznat adaptační vstup od vstupu původní sítě (obrázek 5.9), posílá velké gradienty chyby do adaptační sítě, které mnohonásobně změní gradient v této sítí a tím i váhy. Nicméně gradient, který se propaguje od diskriminátoru, nemá logický smysl. To je z toho důvodu, že diskriminátor pouze hádá odkud vstup došel, protože vstupy jsou si velmi podobné. Díky tomuto pak gradient chyby nemá požadovaný účinek regularizace, ale v tomto případě funguje spíš jako katalyzátor chybovosti na cílových datech. Na obrázku 5.9 je vidět, že diskriminátor začne, kolem 200. iterace rozpoznávat vstupy, jenže to je pravděpodobně dáno tím, že předtím tento diskriminátor změnil váhy adaptační sítě nevhodným způsobem, což mělo za následek zhoršení rozpoznávací schopnosti diskriminátoru.

Shrnutí experimentu nad deníkem ručně psaného textu. U tohoto případu adaptace, se mi nepodařilo překonat naivní metodu adaptace se zapnutou augmentací. U tohoto experimentu je problém diskriminátor samotný. Díky tomu, že nedokáže rozpoznávat vstupy, tak svým gradientem výrazně mění strukturu vah adaptační sítě po dlouhou dobu a to má za následek zhoršování úspěšností nad daty.

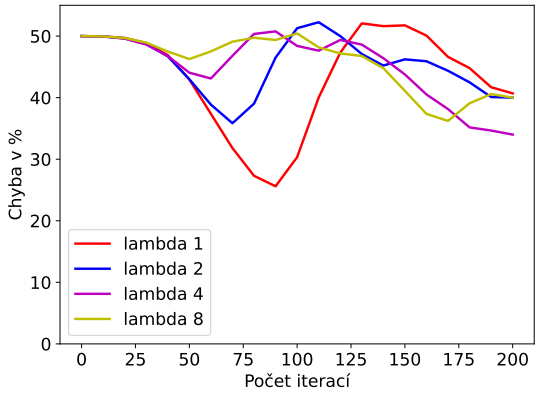
Počty řádků v souboru								
	20		50		100		200	
Chyba	10,82 %							
	μ	σ	μ	σ	μ	σ	μ	σ
λ=0	**11,25**	**0,28**	9,92	0,15	8,97	0,19	**8,48**	0,06
λ=1	11,33	0,42	9,99	0,17	9,02	0,18	8,52	**0,02**
λ=2	11,59	0,36	**9,73**	**0,12**	**8,85**	**0,10**	8,62	0,05
λ=4	11,79	0,33	10,09	0,47	9,13	0,14	8,53	0,04
λ=8	12,62	0,96	10,16	0,20	9,32	0,37	8,65	0,07

Tabulka 5.4: Tabulka výsledků adaptace na deníku s chybou na začátku adaptace 10,82 %. Sloupec výsledků pro každý počet řádků je rozdělen na dvě části. V levé části je zprůměrovaná chyba na cílových datech po 120 iteracích (označená jako μ v %) a v pravé části je vypočítána směrodatná odchylka výsledků spuštění šesti opakování pro každou hodnotu lambda (označená jako σ).



Obrázek 5.10: V obou obrázcích jsou vidět průběhy chybovosti na adaptačních (adpt) a cílových (cil) datech, včetně průběhů chybovosti na diskriminátoru (disc). Na levém obrázku jsou průběhy pro lambdu 1 a na pravém pro lambdu 2.

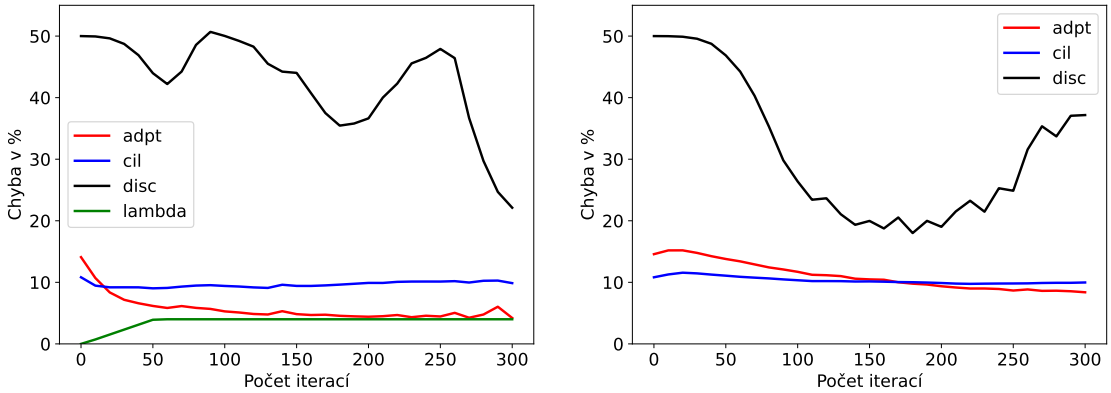
Adaptace na deník s chybou 10,82 % Výsledky tohoto druhého pokusu adaptace na deníku jsou vidět v tabulce 5.4. Z tabulky je vidět, že překonat lambdu 0 se podařilo pouze v dvou případech adaptace. Těmito případy jsou adaptace na 100 adaptačních řádcích, kdy jsem dostal zprůměrovanou chybu při lambdě 2 8,85 % oproti lambdě 0, která měla chybu 8,97 % a na 50 řádcích při lambdě 2 s chybou 9,73 % oproti chybě s lambdou 0 9,92 %. Jedná se o zlepšení 1,3 % u 100 řádků a 1,9 % u 50 řádků vůči lambdě 0. Abych zjistil, proč právě lambda 2 dosahovala zlepšení, rozhodl jsem se podívat na průběh adaptace lambdy 2 u 100 řádkového souboru vůči lambdě 1. Tyto grafy porovnání jsou vidět na obrázku 5.10. Tyto dvě lambdy mezi sebou dělí úspěšnost 1,9 %. Průběhy úspěšností jsou v obou případech velmi podobné. Viditelné rozdíly jsou v průběhu diskriminátorů, kdy diskriminátor s lambdou 1 se dokáže lépe naučit rozpoznávat vstupy, než diskriminátor s lambdou 2. Nicméně co je zajímavé u obou průběhů je, že diskriminátor se oproti experimentu nad datasetem IMPACT nikdy nedokáže kvalitně naučit rozpoznávat původ vstupů. Toto je pravděpodobně způsobeno tím, že počáteční chyba na datech je relativně malá, tím pá-



Obrázek 5.11: Chybovost rozpoznání vstupů diskriminační jednotkou, při rozdílných hodnotách parametru lambda.

dem úpravy od CTC funkce jsou menší, čímž se upraví váhy adaptační sítě jen drobně a to má za následek to, že diskriminátor přijímá jen lehce upravené vstupy od adaptační sítě, ale ty dokáže regularizovat tak, že se vstupy od adaptační sítě skutečně příliš neliší od vstupů původní sítě. Kvůli tomuto pak diskriminátor nezpůsobí žádnou větší úpravu, která by výrazněji zoscilovala jeho průběh. Oscilace úspěšnosti na diskriminátoru je vidět i na průbězích na obrázcích 5.10, ale není tak výrazná jako u experimentu nad datasetem IMPACT. Na obrázcích 5.10 je vidět, že prvních 50 iterací se podstatně snižuje chyba na adaptačních datech, což má za následek větší úpravy vah adaptační sítě a tyto větší úpravy pomáhají lepšímu rozpoznávání diskriminátoru, jelikož se distribuce vah adaptační sítě výrazněji oddaluje od distribuce vah původní sítě a tím se více liší i vstupy diskriminátoru. Jakmile se ale po 50. iteraci začne snižování chyby na adaptačních datech zpomalovat, tak gradient chyby od diskriminační sítě začne víc regularizovat váhy adaptační sítě, což má za následek opětovné přiblížení vstupů diskriminační sítě. Efekt diskriminátoru, pro různé lambdy, je dobře viditelný na obrázku 5.11. Z obrázku je patrné, že oscilační efekt diskriminátoru zde je. Nicméně tento efekt není tak výrazný jako u experimentu nad datasetem IMPACT. Z tohoto obrázku je taky dobře pozorovatelný vliv regularizačního parametru lambda. Čím vyšší je hodnota parametru lambda, tím silnější je regularizace a to má za následek horší schopnost diskriminátoru poznat rozdíl ve vstupech.

Abych s pomocí parametru lambda většího než nula dostal lepší výsledky, pokusil jsem se o zvýšení oscialce na diskriminátoru tím, že jsem v průběhu adaptace hodnotu lambda progresivně zvětšoval až do chvíle, kdy byla na požadované hodnotě. Příkladem je spuštění programu pro lambdu 4 a v závislosti na počtu iterací zvětšování hodnoty parametru lambda od 0 až do požadovaných 4. Tento přístup měl však spíš negativní dopad na úspěšnost na cílových datech. Dalším pokusem jak zlepšit výsledek adaptace byl pokus, kdy jsem vytvořil separátní optimalizátor pro adaptační síť a další pro diskriminační síť. V průběhu adaptace jsem optimalizoval diskriminátor každou iteraci, přičemž CTC jsem optimalizoval každou 10. iteraci. Toto řešení se však ukázalo taky jako neefektivní. Výsledky pokusu s dvěma optimalizátory a i s progresivním zvyšováním hodnoty parametru lambda jsou vidět na obrázcích 5.12. Při zvyšování parametru lambda na 100 adaptačních řádcích jsem dosáhl zprůměrované chyby na cílové sadě, po 300 iteracích, 9,87 %, což je horší výsledek než při použití neupravené metody a lambdy 8. V případě dvou optimalizátorů jsem dostal chybu 9,97 %, což je opět horší výsledek.



Obrázek 5.12: Na levém obrázku je vidět chybovost na adaptačních (adpt), cílových (cil) datech při použití progresivního zvyšování parametru lambda. Na právem obrázku je průběh chybovosti na adaptačních datech, cílových datech a diskriminátoru (disc) při použití dvou optimalizátorů. Diskriminátor je označen jako disc. Hodnota parametru lambda je v obrázku zaznačena zelenou barvou.

Shrnutí experimentu nad 2. deníkem ručně psaného textu. I přes to, že chybovost diskriminátoru v rámci experimentu osciluje tak, jak jsem očekával, tak výsledky adaptace při použití lambd větších než nula, jsou lepší pouze ve dvou případech. Nicméně i přes to, že výsledky s lambdami většími než nula nejsou často lepší, tak se ani nijak výrazněji neliší, krom lambdy 8, od výsledků s lambdou nula. Druhým zjištěním, které z tohoto experimentu vyplývá je, že samotný efekt oscilace na diskriminátoru ještě nezaručuje lepší úspěšnost při použití regularizace z tohoto diskriminátoru.

Kapitola 6

Závěr

Cílem práce bylo vybrat metody sloužící k adaptaci neuronových sítí na pisatele. Nejvhodnější metodu pak naimplementovat a otestovat její přínos na obstaraných datových sadách. Při výběru vhodné metody jsem se rozhodl použít metodu z článku Adversarial Speaker Adaptation. V tomto článku jsou uvedeny dvě možné metody adaptace. Jednou z nich je metoda učení bez učitele a druhou metoda učení s učitelem. Na základě výsledků prezentovaných v tomto článku jsem se rozhodl implementovat metodu učení s učitelem, protože v rámci článku vykazovala lepší výsledky. Při řešení adaptace pomocí tohoto modelu jsem se snažil překonat výsledky, kterých jsem dosáhl při adaptaci naivní metodou s augmentací. Touto metodou je dotrénovat kvalitně natrénovanou síť na adaptační množině dat.

Pro samotné testování implementované metody jsem využíval dva datasety, z nichž jeden je dataset tištěných dokumentů IMPACT a druhým je dataset ručně psaných textů. Při pokusech jsem vždy prováděl experimenty nad čtyřmi množinami adaptačních dat, které obsahovaly 200, 100, 50 a 20 řádků. V rámci adaptační metody prezentované v článku se využívá regularizace pomocí diskriminační sítě. Velikost této regularizace je upravována pomocí parametru lambda. V rámci experimentu nad datsetem IMPACT se mi v několika případech podařilo dosáhnout zlepšení. Nejzajímavějšího zlepšení úspěšnosti se mi podařilo dosáhnout při souboru se 100 adaptačními řádky a při použití lambdy 1 a to o 5,58 % vůči naivní metodě. V případě jiných počtů řádků měla adaptační metoda podobné úspěšnosti jako naivní přístup s augmentací.

Nad datasetem ručně psaných textů jsem prováděl pokusy nad dvěma adaptačními vojenskými deníky. U deníku, s předadaptační chybou 10,82 %, se mi podařilo při adaptačním souboru se 100 řádky dat dosáhnout marginálního zlepšení oproti naivní metodě. Toto zlepšení na cílových datech bylo 1,3 %. Podobné zlepšení bylo i na souboru s 50 řádky a stejnou velikostí lambdy. V ostatních případech, včetně druhého adaptačního deníku, jsem nebyl schopný získat nižší chybu při použití metody vycházející z článku oproti naivnímu řešení. V rámci své práce jsem se pokoušel metodu adaptace dál upravovat, ale to nevedlo k lepším výsledkům na cílových datech. Metoda z článku Adversarial Speaker Adaptation, převedená na rozpoznávání textu, se ukázala jako účinná pouze v některých případech. Z pozorování také vyplývá, že pokud diskriminátor osciluje, tak dochází k regularizaci adaptační sítě, nicméně samotná oscilace není zárukou zlepšení adaptace vůči naivní metodě s augmentací.

V práci bych dále pokračoval tak, že bych naimplementoval do této adaptační metody část, která by využívala učení bez učitele. Při tomto přístupu, bych mohl využit i obrázky cílových dat.

Literatura

[1] Shi, B., Bai, X. a Yao, C. An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition. *CoRR*. 1. vyd. 2015, abs/1507.05717. Dostupné z: http://arxiv.org/abs/1507.05717.

[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. DOI: 10.1109/72.279181.

[3] Clausner, C., Pletschacher, S. a Antonacopoulos, A. Aletheia - An Advanced Document Layout and Text Ground-Truthing System for Production Environments. In: *2011 International Conference on Document Analysis and Recognition*. 2011, s. 48–52. DOI: 10.1109/ICDAR.2011.19. ISBN 978-0-7695-4520-2.

[4] Kingma, D. P. a Ba, J. *Adam: A Method for Stochastic Optimization*. 2017.

[5] Ganin, Y. a Lempitsky, V. *Unsupervised Domain Adaptation by Backpropagation*. 2015.

[6] Gatos, B., Louloudis, G., Causer, T., Grint, K., Romero, V. et al. Ground-Truth Production in the Transcriptorium Project. In: *2014 11th IAPR International Workshop on Document Analysis Systems*. 2014, s. 237–241. DOI: 10.1109/DAS.2014.23. ISBN 978-1-4799-3243-6.

[7] Graves, A., Fernández, S. a Gomez, F. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In: *In Proceedings of the International Conference on Machine Learning, ICML 2006*. 2006, s. 369–376.

[8] Hladká, Z. *111 let českého dopisu v korpusovém zpracování*. Masarykova univerzita, 2013.

[9] Hochreiter, S. a Schmidhuber, J. Long Short-term Memory. *Neural computation*. Prosinec 1997, sv. 9, s. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[10] Kodym, O. a Hradiš, M. *Page Layout Analysis System for Unconstrained Historic Documents*. 2021.

[11] MathWorks. *Convolutional Neural Network 3 things you need to know* [online]. 2020 [cit. 2011-04-23]. Dostupné z: https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html.

[12] Meng, Z., Li, J. a Gong, Y. Adversarial speaker adaptation. In: IEEE. *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2019, s. 5721–5725.

[13] Olah, C. *Understanding LSTM Networks*. 2015 [cit. 2021-04-23]. Dostupné z: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[14] Papadopoulos, C., Pletschacher, S., Clausner, C. a Antonacopoulos, A. The IMPACT dataset of historical document images. In:. Srpen 2013, s. 123–130. DOI: 10.1145/2501115.2501130.

[15] Phi, M. *Illustrated Guide to LSTM's and GRU's: A step by step explanation* [online]. 2018 [cit. 2021-04-23]. Dostupné z: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.

[16] Rao, N., Sastry, A., Chakravarthy, A. a Kalyanchakravarthi, P. Optical character recognition technique algorithms. Leden 2016, sv. 83, s. 275–282.

[17] Sánchez, J. A., Mühlberger, G., Gatos, B., Schofield, P., Depuydt, K. et al. TranScriptorium: A European Project on Handwritten Text Recognition. In: *Proceedings of the 2013 ACM Symposium on Document Engineering*. New York, NY, USA: Association for Computing Machinery, 2013, s. 227–228. DocEng '13. DOI: 10.1145/2494266.2494294. ISBN 9781450317894. Dostupné z: https://doi.org/10.1145/2494266.2494294.

[18] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. a Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*. Červen 2014, sv. 15, s. 1929–1958.

[19] Sánchez, J.-A., Romero, V., Toselli, A. a Vidal, E. ICFHR2014 Competition on Handwritten Text Recognition on Transcriptorium Datasets (HTRtS). In:. Září 2014, sv. 2014, s. 785–790. DOI: 10.1109/ICFHR.2014.137.

[20] Sánchez, J. A., Romero, V., Toselli, A. H., Villegas, M. a Vidal, E. ICDAR2017 Competition on Handwritten Text Recognition on the READ Dataset. In: *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*. 2017, sv. 01, s. 1383–1388. DOI: 10.1109/ICDAR.2017.226.

Příloha A

Obsah přiloženého paměťového média

- `src` – mnou vytvořené zdrojové soubory
- `text` – zdrojové soubory textu této práce
- `video.mp4` – video prezentující mou bakalářskou práci
- `readme.md`