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

STROJOVÝ PŘEKLAD POMOCÍ UMĚLÝCH NEURONOVÝCH SÍTÍ

MACHINE TRANSLATION USING ARTIFICIAL NEURAL NETWORKS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **JONÁŠ HOLCNER**
AUTHOR

VEDOUCÍ PRÁCE **Ing. IGOR SZŐKE, Ph.D.**
SUPERVISOR

BRNO 2018

Abstrakt

Cílem této práce je popsat a vytvořit systém pro strojový překlad textu mezi různými jazyky. Systém je postavený na rekurentních neuronových sítích, konkrétně na enkodér-dekodér architektuře. Výsledky jsou porovnány vůči systému postaveném na nástroji Moses.

Abstract

The goal of this thesis is to describe and build a system for neural machine translation. System is build using recurrent neural networks – encoder-decoder architecture in particular. Results are compared with system build with statistical tool Moses.

Klíčová slova

strojový překlad, neurální strojový překlad, neuronové sítě, rekurentní neuronové sítě, LSTM, enkodér, dekodér, model enkodér-dekodér, sekvence do sekvence, seq2seq, keras, moses, BLEU

Keywords

machine translation, neural machine translation, neural networks, recurrent neural networks, LSTM, encoder, decoder, encoder-decoder model, sequence to sequence, seq2seq, keras, moses, BLEU

Citace

HOLCNER, Jonáš. *Strojový překlad pomocí umělých neuronových sítí.* Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Igor Szöke, Ph.D.

Strojový překlad pomocí umělých neuronových sítí

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Igora Szökeho. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jonáš Holcner
30. března 2018

Poděkování

Tímto bych rád poděkoval panu Ing. Szőkemu, PhD. za vedení mé práce.

Obsah

Kapitola 1

Úvod

Přísloví praví „Kolik řečí znáš, tolikrát jsi člověkem“. Schopnost dorozumět se s ostatními lidmi na planetě je nesmírně důležitá a přitom jazyková bariéra je překážkou v mezilidské komunikace už od pradávných let. Proto vznikaly a vznikají jednoduché tištěné a následně digitální slovníky a vědci od počátků vzniku výpočetní techniky zkoumají jak vytvořit funkční překladový systém.

Ideálem je překlad tak jak ho známe ze science fiction materiálů. Dvě osoby, mluvící kompletně jiným jazykem si navzájem rozumí v reálném čase. S rozvojem který nastal v posledních letech, tedy intenzivní rozvoj strojového učení a s nástupem umělé inteligence používající hlubokých neuronových sítí se k tomuto ideálu blížíme mílovými kroky. Automatické rozpoznání mluvené řeči je již ve skvělé kvalitě dostupné v běžných spotřebitelských zařízeních a překlad se taky značně vylepšuje.

Obsahem této práce je návrh a realizace překladového systému schopného naučit se, za pomocí datasetů v různých zdrojových a cílových jazycích, překládat věty mezi těmito jazyky. A to pomocí nejnovějších metod, objevených a široce nasazovaných v posledních letech, používajících rekurentní neuronové sítě s architekturou enkodér-dekodér.

V kapitole 2 je neformálně nastíněn návrh a cíl této práce. V následující kapitole 3 jsou pak rozebrány důležité pojmy a teorie ze kterých je tato práce vystavěna. Kapitole 4 popisuje co bylo implementováno a kapitola 5 podává výsledky experimentů.

Kapitola 2

Neformální návrh systému

Cílem této práce práce je vytvořit systém pro strojový překlad textu pomocí umělých neuronových sítí. Pro snadnou představu, je to podobné jako to co dělá Google Translator[1] – blíže popsáno v článku [25]. Vezme se věta v původním jazyce a vytvoří se z ní co nejvěrnější překlad v jazyce cílovém a to za pomocí natrénované neuronové sítě. V této kapitole je vysvětleno, jak by takový systém mohl vypadat a co za komponenty potřebuje k tomu aby fungoval.

Dataset: Aby bylo možné nacvičit neuronovou síť pro překlad, je nejprve zapotřebí mít nějaký dataset. Jeden dataset obsahuje texty ve dvou jazycích mezi kterými se má překládat. Tyto texty musí být zarovnané, tak aby si jednotlivé věty v těchto jazycích navzájem odpovídaly. Obecně platí, že čím větší množství použitých dat a čím větší model, tím lepší bude výsledek (článek [11]). Ukázka datasetu je znázorněna na obrázku 4.1.

Tokenizer: Dataset a jeho jednotlivé věty před začátkem trénování sítě je nejprve potřeba připravit. Tokenizer rozdělí věty na jednotlivé tokeny (obrázek 4.2) a zahodí zvolené nepodstatné vlastnosti, které mohou být třeba velká písmena na počátku vět nebo interpunkce. To usnadňuje práci s datasety a také například snižuje velikost slovníků.

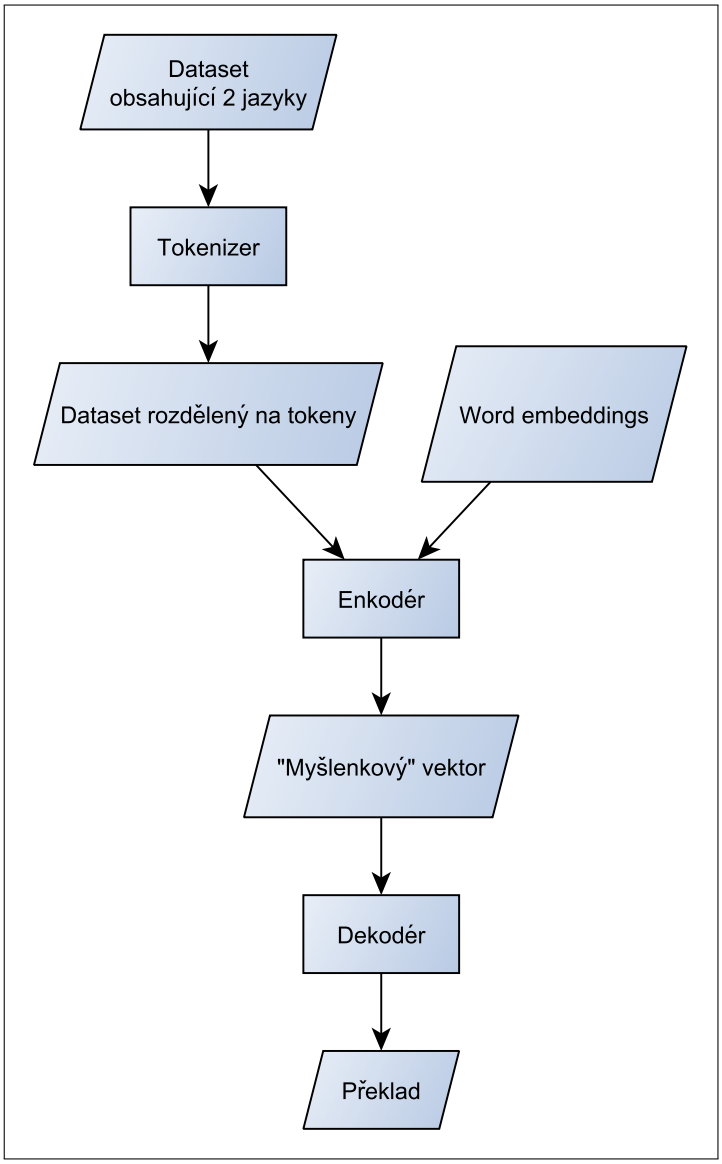
Slovník: Slovník se vytvoří jako seznam n nejčastějších slov v datasetu ve vstupním a cílovém jazyce. Čím je slovník menší, tím se zmenší výpočetní požadavky, ale na druhou stranu je potřeba vyřešit při trénování a překladu problém se slovy nevyskytujícími se ve slovníku (popsáno v sekci 3.1.4).

Word Embeddings: Obecně je možné vytvářet jazykové modely, které generují text po písmenech, částech slov nebo po slovech [15]. Word Embeddings je další forma předzpracování. Každý token ze vstupního slovníku se převede do vektoru reálných čísel, ve kterém jsou zakódovány některé syntaktické a sémantické vlastnosti daného tokenu, což umožní neuronové síti se učit lépe, než kdyby se použilo například jenom číslo označující pozici tokenu ve slovníku. Více v sekci 3.1.3.

Model: Pro překlad je nejvhodnějším způsobem sequence to sequence (dále seq2seq [24]) s použitím architektury enkodér-dekodér. Na rozdíl od starších statistických metod překladu, kde se překládalo po frázích, moderní překlad pomocí neuronových sítí probíhá po celých sekvencích (větách). Nejprve enkodér vezme word embedding na

[1] translate.google.cz

vstupu a pomocí rekurentní neuronové sítě (sekce 3.2) převede větu na vstupu do velkého vektoru reprezentující její význam (tzv. myšlenkový vektor – intuice je taková, že když člověk překládá větu, také nejprve pochopí její význam a poté ji až začne překládat). Dekodér – taky rekurentní neuronová síť – následně z tohoto vektoru slovo po slovu vygeneruje výslednou přeloženou větu. Dekodér tedy funguje jako jazykový model (sekce 3.1), který je na inicializovaný na jednu konkrétní větu.



Obrázek 2.1: Schéma návrhu systému pro překlad. Dataset se předzpracuje pomocí tokenizeru. Do enkodéru vstupují tokeny převedené na embeddings. Enkodér větu zakóduje do velkého "myšlenkového"vektoru ze kterého dekodér generuje překlad.

Kapitola 3

Související teorie a pojmy

Účelem této kapitoly je blíže vysvětlit a rozebrat jednotlivé pojmy a komponenty potřebné pro vytvoření překladového systému.

3.1 Jazykové modely

Zatímco u programovacích jazyků existuje jejich formální definice přesně popisující jejich syntaxy a význam, u přirozených jazyků to tak není. Přirozený jazyk vznikl náhodným způsobem v průběhu staletí a tisíciletí narozdíl od formálně definovaných jazyků, které byly navrženy. Přestože běžný jazyk se řídí nějakými pravidly, existuje značné množství výjimek a odchylek. I napříč tomu si však lidé navzájem rozumí. Problém však je tyto pravidla převést do formálních pravidel, tak aby jim rozuměl počítač. Řešením pro tento problém mohou být jazykové modely, které nevznikají nadefinováním formálních pravidel, ale nacvičením modelu z příkladů. Sekce vychází z práce [13] a článku [17].

Jazykový model udává pro každou větu w jaká je její pravděpodobnost. Respektive pro sekvenci slov $w = w_1, w_2..., w_m$ získá pravděpodobnost podle rovnice 3.1.

$$p(w) = \prod_{i=1}^{m} p(w_i|w_{<i}) \tag{3.1}$$

Pro každé slovo w_i ze sekvence w určí jaká je jeho podmíněná pravděpodobnost v případě, že se před ním nachází slova $w_{<i}$.

3.1.1 N-gram modely

Ve výsledku je pro překladový systém potřeba získat model, který pro zdrojovou větu F vrátí přeloženou větu E, tak že $P(E|F)$. N-gram model je však jazykový model, který udává jen pro pravděpodobnost věty P(E) (pro nějaký daný kontext nad kterým se model nacvičil).

Takovýto model umožní zhodnotit přirozenost věty a generovat text podobný tomu, na kterém byl model nacvičen.

Zhodnocení přirozenosti: Pomocí jazykového modelu je možné pro větu w zhodnotit, jak moc je přirozená nebo-li jak moc je pravděpodobné, že by takováto věta mohla existovat v textu na kterém byl model natrénován.

Generování textu: Protože model umožňuje pro každé slovo w_i získat pravděpodobnost následujícího slova w_{i+1}, je takto možné generovat náhodný, přirozeně (vůči zdrojovému textu) vypadající text. Přesně tato vlastnost je potřeba pro generování překladů.

N-gram modely umožňují určit pravděpodobnost následujícího slova ve větě v případě, že se před ním nacházelo n nějakých slov (rovnice 3.2).

$$P(x_i \mid x_{i-(n-1)}, \ldots, x_{i-1}) \tag{3.2}$$

Se zvětšujícím se n se výrazně zvětšuje náročnost výpočtu. Tímto způsobem tak není snadné zachytit závislosti mezi slovy vzdálenými od sebe více než pár míst.

3.1.2 Log-lineární modely

Stejně jako v případě n-gram modelů (sekce 3.1.1), tyto modely počítají pravděpodobnost následujícího slova w_i při kontextu $w_{<i}$. N-gram model počítá pouze s výskytem (identitou) slova. Log-lineární modely pracují s **rysy** (z anglického features). Rys je něco užitečného ohledně daného slova, co se dá použít pro zapamatování a pro předpověď slova dalšího. Jak už bylo řečeno, u n-gram modelů to je pouze identita minulého slova. Formálněji je rys funkce $\phi(e_{t-n+1}^{t-1})$, která dostane na vstupu aktuální kontext a jako výsledek vrátí reálnou hodnotu – vektor rysů $x \in \mathbb{R}^N$ popisující kontext při použití N různých rysů.

Stejně jako u n-gram modelů, nastává problém když je potřeba zaznamenat vzdálenější závislosti. Například u věty „farmář jí steak“ je potřeba zaznamenat pro předpovězení slova „steak“ jak jeho předcházející slovo $w_{t_1} = jí$, tak $w_{t_2} = farmář$. V případě, že by se použil pouze rys w_{t_1}, mohl by model předpovídat i věty, které nedávají smysl. Jako je například „kráva jí steak“. Při použití většího množství rysů vznikají mnohem větší nároky na paměť a výkon a taky na velikost trénovacího datasetu. Řešením těchto problémů může být použití neuronových sítí (sekce 3.1.3).

3.1.3 Neuronové sítě a word embeddings

Stejně jako předchozí modely i NLM (neural language model) je trénován tak aby předpovídal rozložení pravděpodobností přes slova v cílovém slovníku na základě aktuálního kontextu (rovnice 3.1).

Předchozí modely při použití většího datasetu a tím pádem většího slovníku čelí „prokletí“ dimenzionality. Jednotlivá slova jsou běžně reprezentována jako **one-hot vektor** (obrázek 3.1). Pro reprezentaci jednoho slova je tak použit rozsáhlý vektor $x_i \in \mathbb{R}^V$, kde V je použitý slovník daného jazyka. Většina hodnot, až na hodnotu označující dané slovo, je nulová (řídký vektor nebo-li sparse vector).

$$V = [\text{farmář, jí, steak, kráva}] \qquad oneHot_{steak} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Obrázek 3.1: One-hot vektor pro slovo „steak“ ze slovníku V. Slovo je znázorněno jedničkou na třetí pozici, což odpovídá jeho pozici ve slovníku. Všechny ostatní pozice vyplňují pouze nuly (řídký vektor). Pro velký slovník to znamená, že každé slovo zabere značné množství paměti.

NLM se s tímto problémem vypořádává za pomocí takzvaných **word embeddings**. Word embeddings, jsou na rozdíl od one-hot vektoru vektory reálných čísel (husté nebo-li dense vektory). Ke každému slovu ze slovníku se přiřadí takovýto vektor. Výhodou je, že může nést, narozdíl od pouhé pozice slova ve slovníku, další různé užitečné významy. Třeba pro slovo „kráva“, by ve vektoru mohly být zakódované významy jako podstatné jméno, velký savec atd. Díky tomu může model lépe generalizovat, a slova, která jsou sobě blízká v tomto prostoru, může model brát například jako synonyma. Nejznámější ukázkou vlastností word embeddings je ukázka 3.2 z článku [16].

$$v(král) - v(muž) + v(žena) \approx v(královna)$$

Obrázek 3.2: Ukázka vlastností word embeddings. ≈ udává nejbližšího souseda v prostoru. Je vidět, že vektory v sobě nesou určitý sémantický význam. Odečtením hodnoty vektoru slova „muž“ se získá jakási podstata slova „král“ nebo „kralovat“. Přičtením hodnoty slova „žena“ k této dočasné hodnotě se pak získá ženská varianta krále – královna.

Existuje několik variant výpočtů word embeddings – word2vec [14], glove [20] a fasttext [3].

Embbedings jsou vhodné pro **transfer learning**. To je způsob využití znalostí získaných někde jinde pro jiný problém. Word embeddings je možné buďto získat v průběhu učení modelu nebo použít už předtrénované, připravené pro tento účel. Díky tomu může model získat více znalostí o jednotlivých slovech a celkový výsledek může být výrazně lepší.

3.1.4 Zpracování neznámých slov

Jazykový model typicky pracuje s pevně danou velikostí slovníku (počtu slov, co se mohou vyskytovat), což je problém, protože překlad je obecně problém s otevřenou slovní zásobou. Existuje-li dataset ε_{train} obsahující texty na kterých se model bude učit a dataset ε_{test}, který bude sloužit k ověření výkonosti a generalizace modelu, je více než pravděpodobně, že v testovacím setu se budou nacházet slova, která se v trénovacím nenacházela. To znamená, že se v testovacím datasetu budou vyskytovat **neznámá slova**. Pro pokrytí co největšího množství slov by tak bylo potřeba učit model s co největším slovníkem, to by ale bylo neefektivní a tak naopak může být vhodné omezit celkový počet slov se kterými se bude model trénovat pro zlepšení výkonu a tím pádem ale zvýšit výskyt neznámých slov. Práce [17] uvádí tři běžné způsoby jak se vypořádat s neznámými slovy.

Předpokládat že slovník je konečně velký: V některých případech se dá počítat s tím, že slovník je omezený. Tím pádem se neznámá slova nebo znaky nemohou vyskytovat. Například, když by se trénoval model pouze na znacích ASCII, tak při dostatečně velkém vstupním datasetu by bylo rozumné předpokládat, že se v něm vyskytly všechny znaky a model se je tedy mohl všechny naučit.

Interpolovat pravděpodobnost pro neznámá slova: Je možné interpolovat rozložení pravděpodobnosti i přes neznámá slova. Lze natrénovat jazykový model co by po písmenech odhadoval neznámá slova nebo lze odhadnout celkový počet slov ve slovníku a pravděpodobnost P_{unk} pak počítat jako $P_{unk}(e_t) = 1/|V_{all}|$.

Přidáním speciálního slova <unk>: V případě, že se v trénovacím setu ε_{train} některá slova vyskytují málo nebo jenom jednou, mohou se nahradit speciálním slovem *<unk>*. S tímto slovem se pak pracuje stejně jako s ostatními. Díky tomu se zredukuje počet slov ve slovníku a tedy náročnost výpočtu. Má však přiřazenou svoji pravděpodobnost a může se tak vyskytnout v předpovědi modelu při generování textu.

Dalším uváděním řešením, je místo slov jako nejmenší jednotky se kterou se pracuje (tokenem), je použití takzvaných **subword units** nebo-li jednotek menších než slovo [22].

Subword units: Kvůli velkému množství slov a jejich tvarů, se obvykle používá velikost slovníků kolem 30 000-50 000 tisíců slov. Kvůli tomu je trénink a překlad náročnější a také to nezaručuje kvalitní překlad pro vzácná slova ani že se nevyskytnou slova neznámá. Značné množství slov je tvořeno několika pod částmi, předložkami a spojeninami (například „kladkostroj“). Ukázalo se, že v případě, že se slova rozdělí na takovéto podčásti tak se:

1. Zmenší velikost slovníku - když by se místo slov pro trénování sítě používaly jednotlivá písmena, velikost slovníku by byla jen samozřejmě nejmenší, ale trénink modelu by byl složitý. Při použití částí slov je možné z nich skládat větší celky a nemusí se tak ve slovníku vyskytovat tolik slov. Je to vhodná kombinace mezi velikostí slovníku a množstvím závislostí co model musí natrénovat.
2. Zredukuje se výskyt neznámých slov - protože se slovník skládá z jednotek menších jak slovo, je možné z nich seskládat všechna slova z trénovacího datasetu a tím v něm zrušit výskyt neznámých slov. V případě výskytu neznámých slov v době překladu je možné z menších jednotek slovo přeložit po částech.

Pro rozložení na menší jednotky se používá *byte pair encoding* (BPE). BPE je jednoduchá kompresní metoda. V práci [22] je použita pro spojování písmen nebo sekvencí ve větší celky. Vybere se počet, kolikrát má proběhnout spojování, v každé iteraci se vyberou nejčastěji se vyskytující znaky nebo sekvence ve slovníku a ty se spojí. Nejčastější slova tak zůstanou zachována a ostatní budou rozdělena na různé množství n-gramů. Díky zachování nejčastějších slov, nevzniká výrazný problém s nárůstem délek sekvencí, která by zapříčinila horší učení modelu, protože delší sekvence znamená větší vzdálenost, přes kterou musí model přenést informace.

3.2 Rekurentní neuronové sítě

V této kapitole je popsán základní koncept rekurentních neuronových sítí (RNN[1]), jejich srovnání s běžnými neuronovými sítěmi a dále pak popis upravených variant rekurentních sítí – LSTM (sekce 3.2.3) a GRU (sekce 3.2.4). Sekce vychází z práce [13], práce [17] a článku [18].

RNN (článek [6]) jsou známé již přes dvě desítky let. Úspěšně jsou však používány až v posledních letech. A to hlavně díky vyššímu výpočetnímu výkonu a většímu objemu trénovacích dat, který je v současné době dostupný a také zpracovatelný. Tento druh neuronových sítí je obzvlášť vhodný například pro rozpoznávání psaného písma, rozpoznávání řeči, v kombinaci s konvolučními neuronovými sítěmi pro generování popisků obrázků a co

[1] z anglického recurrent neural network

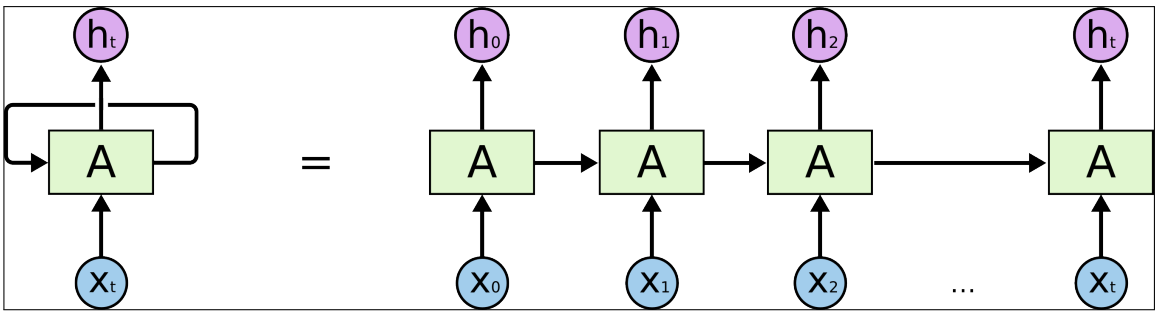
je nejvíce zajímavé pro tuto práci, pro tvorbu jazykových modelů, generátorů textu a tím pádem i pro překlad.

Jejich hlavních výhodou oproti jednoduchým dopředným neuronovým sítím je jejich schopnost držet si vnitřní stav napříč časem. Základní neuronová síť pracuje vždy s aktuální hodnotou x na vstupu, pro kterou pomocí vah W získá výstup y (rovnice 3.3).

$$y = f(x, W) \tag{3.3}$$

Pokud pak takováto síť pracuje s nějakou sekvencí měnící se v čase, například se slovy v rámci jedné věty, pro každé slovo na vstupu x_t, kde t znázorňuje čas (pozici) slova ve větě, použije stejné váhy pro získání výstupu y_t a nezjistí ani nezachová žádnou úvahu o vzájemném vztahu těchto slov.

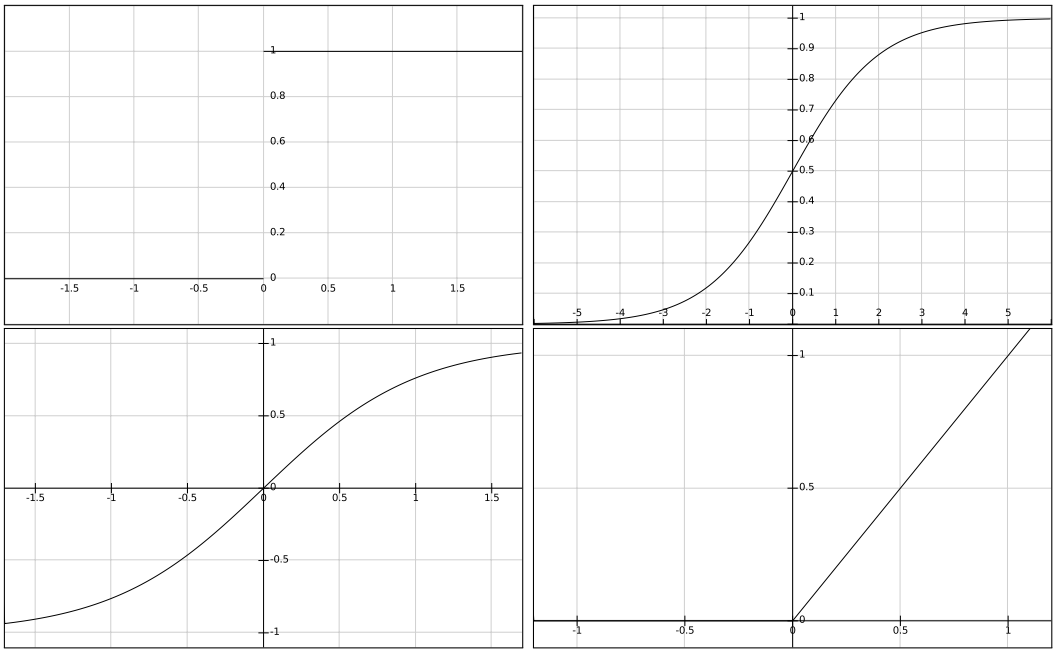
RNN tento problém řeší zavedením skrytého stavu h_t a zpětné smyčky (obrázek 3.3). Vstupem dalšího stavu je kromě nového vstupu vždycky také výstup ze stavu minulého. Pro každé x_t ze sekvence se tedy nyní může získat výstup y_t pomocí vnitřního stavu h_t z předchozího kroku t (rovnice 3.4). Přičemž počáteční stav h_0 je obvykle nastaven na nulu.



Obrázek 3.3: Znázornění RNN – x_t je vstup, A zastupuje vnitřní chování RNN a h_t je skrytý stav. Rozdíl oproti běžné dopředné neuronové síti je zpětná smyčka a skrytý stav. Pravá část obrázku ukazuje pro lepší představu místo zpětné smyčky rozbalenou strukturu přes jednotlivé časy t. Intuitivně se pak dá odhadnout, že RNN umí dobře pracovat s podobnými strukturami jako jsou sekvence a seznamy. Obrázek převzat z [18].

$$h_t = \begin{cases} f(W_{xh}x_t + W_{hh}h_{t-1} + b_h) & \text{pokud t} \geq 1, \\ 0 & \text{jinak.} \end{cases} \tag{3.4}$$

W_{xh} znázorňuje váhy pro aktuální vstup, W_{hh} jsou váhy pro skrytý stav z minulého kroku a b_h je bias. Funkce f z rovnice 3.4 je nelineární funkcí a nejčastěji se používá jedna z funkcí *step*, *sigmoida*, *tanh* nebo *relu* (obrázek 3.4).



Obrázek 3.4: Funkce *step*, *sigmoida*, *tanh* a *relu*.

Rovnice pro RNN jazykový model jsou následující:

$$m_t = M_{e_{t-1}} \tag{3.5}$$
$$h_t = RNN(m_t, h_{t-1}) \tag{3.6}$$
$$s_t = W_{hs} h_t + b_s \tag{3.7}$$
$$p_t = softmax(s_t) \tag{3.8}$$

Kde 3.5 je aktuální kontext, 3.6 je zjednodušený přepis rovnice RNN 3.4 a rovnice 3.8 je funkce softmax (rovnice 3.9), která vezme všechny hodnoty skóre pro jednotlivá slova a transformuje je do pravděpodobnostního rozložení p_t. Díky tomu pak lze již snadno určit, které slovo bude vygenerováno s největší pravděpodobností.

$$p_t(y) = \frac{e^{p_t(y)}}{\sum_{k=1}^{K} e^{p_{t_k}}} \tag{3.9}$$

Protože vektor m z rovnice 3.5 je konkatenací všech předchozích slov (a tedy je to aktuální kontext), model se může naučit kombinaci různých vlastností napříč několika různými slovy z kontextu. V sekci 3.1.2 byl jako problém uveden příklad „Farmář jí maso“ a „Kráva jí maso“, kde druhá věta nedává smysl. Při použití RNN by se pro kontext M_f {farmář, jí} mohla naučit jedna z jednotek skryté vrstvy h rozpoznat vlastnost "věci které farmář jí"a správně se aktivovat a pak nabízet slova jako „maso“ nebo „brambory“. Zatímco pro kontext M_k {kráva, jí} by se naučila zase jiná jednotka. RNN je tedy schopná zachytit tyto vzdálenější závislosti. Základní verze RNN je však schopná zachytit závislosti jen do určité vzdálenosti viz 3.2.2.

3.2.1 Trénování

Cílem trénování sítě je nalézt takové parametry θ (kombinace vah W a biasu b) aby byla co nejmenší hodnota takzvané *loss funkce*. Loss funkce vyjadřuje jak moc špatně výstupy sítě odpovídají datům na kterých se síť učí. Průchod sítí a následné vypočítání loss funkce se nazývá dopřednou propagací.

[[jakej pouzit vzorec]]

$$a \tag{3.10}$$

K optimalizaci parametrů pro najití minima loss funkce se používá zpětná propagace. Vypočte se přírůstek pro každý parametr, tak aby síť s novými váhy o něco lépe pracovala a loss funkce se snížila. Existuje více různých metod optimalizace pro tento výpočet, lépe popsaných v práci [21].

Úprava parametrů může probíhat po každém jednom průchodu dat (jedné sekevence) sítí. Takovýto přístup se nazývá **online** učení. Dalším přístupem je **učení po dávkách**. V takovém to případě probíhá přepočet parametrů až po průchodu přes n sekvencí. Toto číslo n se nazývá **batch size**, tedy počet dat v jedné dávce.

[[epoch, batchsize....loss, back propagation (over time)]] [[z tutorialu, sekce 6.5 strana 35..nejak popsat batch, minibatch, online batch, sentence padding a masking]]

[[trénování rnn, back propagation through time, have difficulties (http://proceedings.mlr.press/v28/pascanu13.pdf) learning long term dependencies]] [[loss computing]] [[gradient computing]]

Gradient descent methods

- a

http://ruder.io/optimizing-gradient-descent/index.html#stochasticgradientdescent **[[jak prelozit gradient descent? popsat co to je, ke cemu to je a pak vypsat ty jednotlivy metody a trochu je popsat]]** https://arxiv.org/abs/1609.04747

3.2.2 Mizející a explodující gradient

RNN jsou oproti základním neuronovým sítím schopné zachytit různé závislosti mezi slovy na delší vzdálenosti. I tato schopnost je však velmi limitovaná. Hlavními zdroji problémů jsou **mizející a explodující gradient** (článek [2]).

[[patrne v nejake predchozi casti zhruba popsat LOSS, BACKPROPAGATION aby se tady na to pak dalo navazat]] v sekci 3.2.1

Při průběhu učení RNN průběžně vznikají predikce a počítá se *loss* funkce. Následně je potřeba zpětně zpropagovat tuto hodnotu přes všechny (časové) kroky sítě (Back propagation over time – BPTT). Pokud však není gradient rovný 1, tak se v každém zpětném kroku buďto zmenší a tím pádem se blíží k nule, nebo se naopak zvětší a blíží s k nekonečnu. Ve výsledku je tak gradient buďto příliš malý a nemá tak žádný efekt na úpravu vah nebo jimi pohne příliš a tak zaviní špatné učení se sítě.

Jako možné řešení těchto problémů vznikla varianta rekurentní sítě LSTM (sekce 3.2.3).

3.2.3 LSTM

Long short term memory, dále LSTM, (původní článek [9] a varianta LSTM s forget gate, která se zde používá [7]) nebo-li dlouhá krátkodobá paměť, je varianta RNN navržená jako řešení problémů mizejícího/explodujícího gradientu a vzdálených závislostí.

Stejně jako základní RNN (sekce 3.2), se dá LSTM představit jako opakující se modul v řetězové struktuře (viz obrázek 3.3). Rozdíl je ve vnitřku modulu A. Zatímco RNN používá pouze jednu nelineární funkci (rovnice 3.4), struktura LSTM je složitější (obrázek 3.5 a následující rovnice).

$$u_t = tanh(W_{xu}x_t + W_{hu}h_{t-1} + b_u) \tag{3.11}$$
$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \tag{3.12}$$
$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \tag{3.13}$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \tag{3.14}$$
$$c_t = i_t \odot u_t + f_t \odot c_{t-1} \tag{3.15}$$
$$h_t = o_t \odot tanh(c_t) \tag{3.16}$$

RNN má pouze skrytý stav h. LSTM má navíc ještě paměťovou buňku c (rovnice 3.15). Protože gradient této buňky je právě jedna, netrpí tak LSTM problémy ze sekce 3.2.2 a mohou tak v ní být zachyceny i vzdálené závislosti.

Rovnice 3.11 je update funkcí a je ekvivalentem rovnice 3.4 z RNN. Dále LSTM obsahuje tři různé **brány**. **Zapomínací**, **vstupní** a **výstupní**. Tyto brány určují a kontrolují co se nachází v paměti c_t.

Nejdříve se LSTM rozhodne, jaké informace se vyhodí z paměti. K tomuto slouží již zmíněna zapomínací brána nebo-li forget gate (rovnice 3.12). Například v případě, že síť narazí na vstupu na podstatné jméno, mohla by chtít zapomenout rod posledního podstatného jména, který by si mohla uchovávat pro správné generování sloves v minulém čase.

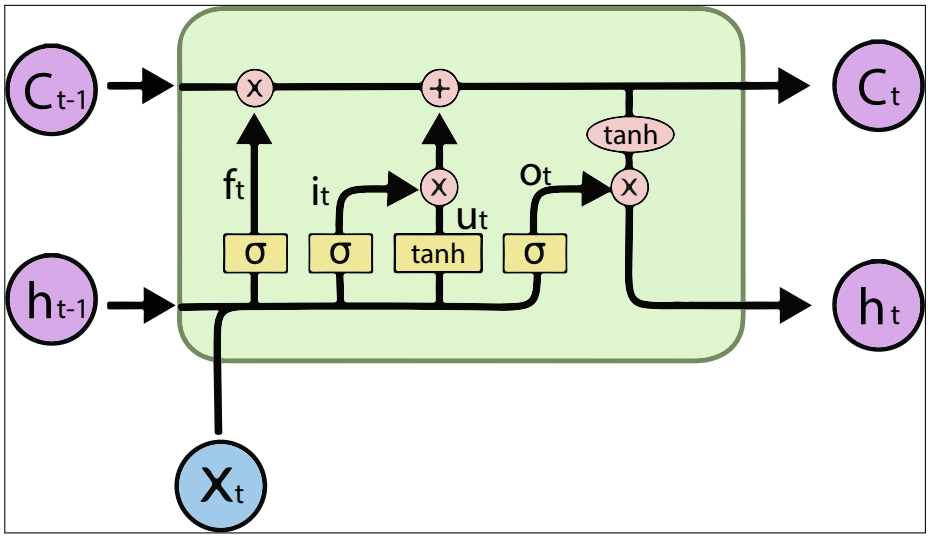
Dalším krokem je vyhodnocení toho co se má přidat do paměti. Nejdříve vstupní brána nebo-li input gate (rovnice 3.13) rozhodne, které hodnoty se změní nebo přidají. V návaznosti na minulý příklad by síť mohla chtít uložit aktuální rod nalezeného podstatného jména. Update funkce (rovnice 3.11) vyhodnotí jaké hodnoty se mají přidat.

Následuje aktualizace paměti c_t (rovnice 3.15). V kontextu příkladu by se zahodil rod jak určila zapomínací brána a uložil se nový rod podle vstupní brány.

Posledním krokem je určení toho co vydá LSTM na výstupu (skrytý stav h_t). Výstupní brána určí co z paměti c_t má projít (rovnice 3.14) a v rovnici 3.16 se získá výsledek.

Pravděpodobnosti jazykového modelu se získají rovnicí:

$$p_t = softmax(W_{hs}h_t + b_s) \tag{3.17}$$



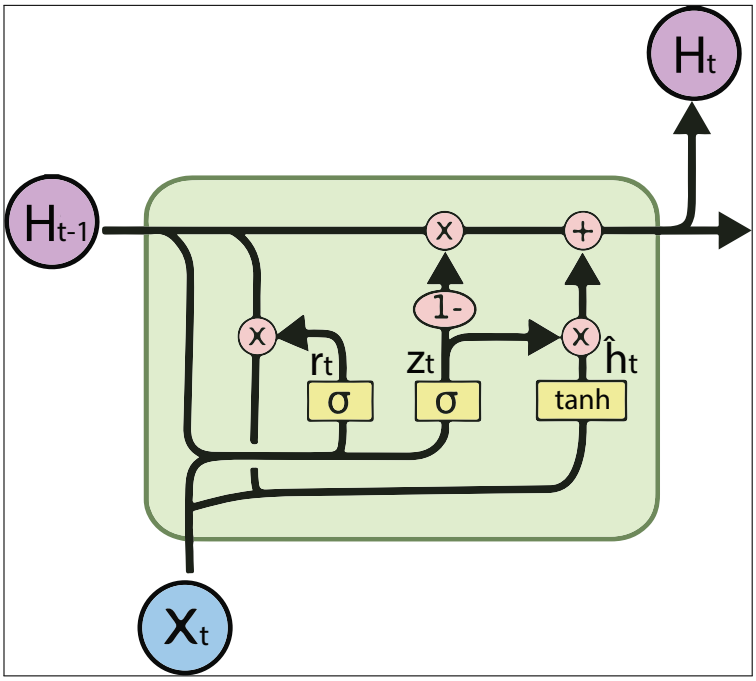
Obrázek 3.5: Jeden časový úsek LSTM. h je skrytý stav, c je paměťová buňka a x je vstup. Vnitřní struktura koresponduje s rovnicemi 3.11 až 3.16. Obrázek převzat z [18], upraven.

3.2.4 GRU

LSTM (sekce 3.2.3) je dobrým řešením pro problémy ze sekce 3.2.2. Její struktura je ale dosti komplikovaná a tím pádem i náročná na výpočetní výkon. To podnítilo vznik další varianty RNN – GRU, nebo-li gated recurrent unit (článek [5]), která je o něco jednodušší a proto je možné ji použít pro úsporu výkonu.

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \tag{3.18}$$
$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \tag{3.19}$$
$$\tilde{h}_t = tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h) \tag{3.20}$$
$$h_t = (1 - z_t)h_{t-1} + z_t\tilde{h}_t) \tag{3.21}$$

GRU má pouze dvě brány a skrytý stav h. Nový stav se počítá v rovnici 3.21 interpolací mezi minulým stavem h_{t-1} a kandidátem na nový stav $\tilde{h}_t$ upravený hodnotou **update** brány (rovnice 3.19). Kandidát se získá v rovnici 3.20, která je podobná update funkci z RNN (rovnice 3.4), ale je upravena hodnotou **resetovací** brány (rovnice 3.18). Struktura je vyzobrazená na obrázku 3.6.



Obrázek 3.6: Jeden časový úsek GRU. h je skrytý stav a x je vstup. Vnitřní struktura koresponduje s rovnicemi 3.18 až 3.21. Obrázek převzat z [18], upraven.

3.3 Seq2seq model s architekturou enkodér-dekodér

V předchozích sekcích se práce zabývá rekurentními neuronovými sítěmi a jazykovými modely na nich postavených. V této sekci bude popsáno jak tyto sítě vzít a poskládat je vhodným způsobem pro překlad vět. Sekce vychází z práce [17].

Seq2seq (článek [24]) nebo-li sequence to sequence je způsob překladu po celých větách. Jde o modelování pravděpodobnosti $P(E|F)$ tedy pravděpodobnost výstupu E na základě vstupu F (obrázek 3.7).

$$\boxed{W_{in} = \text{„Ahoj světe“}} \Longrightarrow \boxed{W_{out} = \text{„Hello world“}}$$

$$P(W_{out}|W_{in})$$

Obrázek 3.7: Seq2seq modeluje pravděpodobnost $P(W_{out}|W_{in})$. Znamená to, že se naučí předpovídat větu W_{out} na základě věty W_{in} a tím pádem překládat.

Pro tento druh překladu celých vět za pomocí rekurentních neuronových sítí se používá model s architekturou **enkodér-dekodér**. Enkodér i dekodér jsou RNN modely. Enkodér dostane na vstupu větu určenou pro překlad a převede ji (enkóduje) do vektoru reálných čísel – skrytý stav, takzvaný myšlenkový vektor, vyjadřující význam dané věty. Dekodér inicializovaný tímto stavem generuje (dekóduje z myšlenkového vektoru) přeloženou větu. Díky tomu, že dekodér generuje z významového vektoru, nemusí být vstupní věta stejně dlouhá jako výstupní.

$$m_t^{(f)} = M_{f_t}^{(f)} \tag{3.22}$$

$$h_t^f = \begin{cases} RNN^{(f)}(m_t^{(f)}, h_{t-1}^{(f)}) & \text{pokud t} \geq 1, \\ 0 & \text{jinak.} \end{cases} \tag{3.23}$$

$$m_t^{(e)} = M_{e_{t-1}}^{(e)} \tag{3.24}$$

$$h_t^e = \begin{cases} RNN^{(e)}(m_t^{(e)}, h_{t-1}^{(e)}) & \text{pokud t} \geq 1, \\ h_{|F|}^f & \text{jinak.} \end{cases} \tag{3.25}$$

$$p_t^{(e)} = \text{softmax}(W_{hs} h_t^{(e)} + b_s) \tag{3.26}$$

Pro každé slovo v čase t ze vstupní sequence F se vyhledá jeho embedding (rovnice 3.22). Následně se v rovnici 3.23 spočítá skrytý stav enkodéru. Po projití přes celou vstupní větu by měly uvnitř být uloženy všechny informace potřebné pro inicializaci dekodéru. I pro dekodér se nejprve vyhledá pro vstupní slovo jeho embedding (rovnice 3.24). Použité slovo není z času t, ale z času $t-1$, protože dekodér generuje následující slovo vždy na základě předchozího. V čase t_0 se jako vstupní slovo používá **startovací** symbol <s>. Rovnice pro výpočet skrytého stavu dekodéru (3.25) je prakticky stejná jak u encodéru. Pouze v čase t_0 se použije koncový stav enkodéru jako inicializace ze které může dekodér vycházet při překladu – ve vnitřním stavu je zachycen význam věty, kterou má přeložit. Pravděpodobnostní rozložení se pak jako u všech jazykových modelů spočítá pomocí funkce *softmax* (rovnice 3.26).

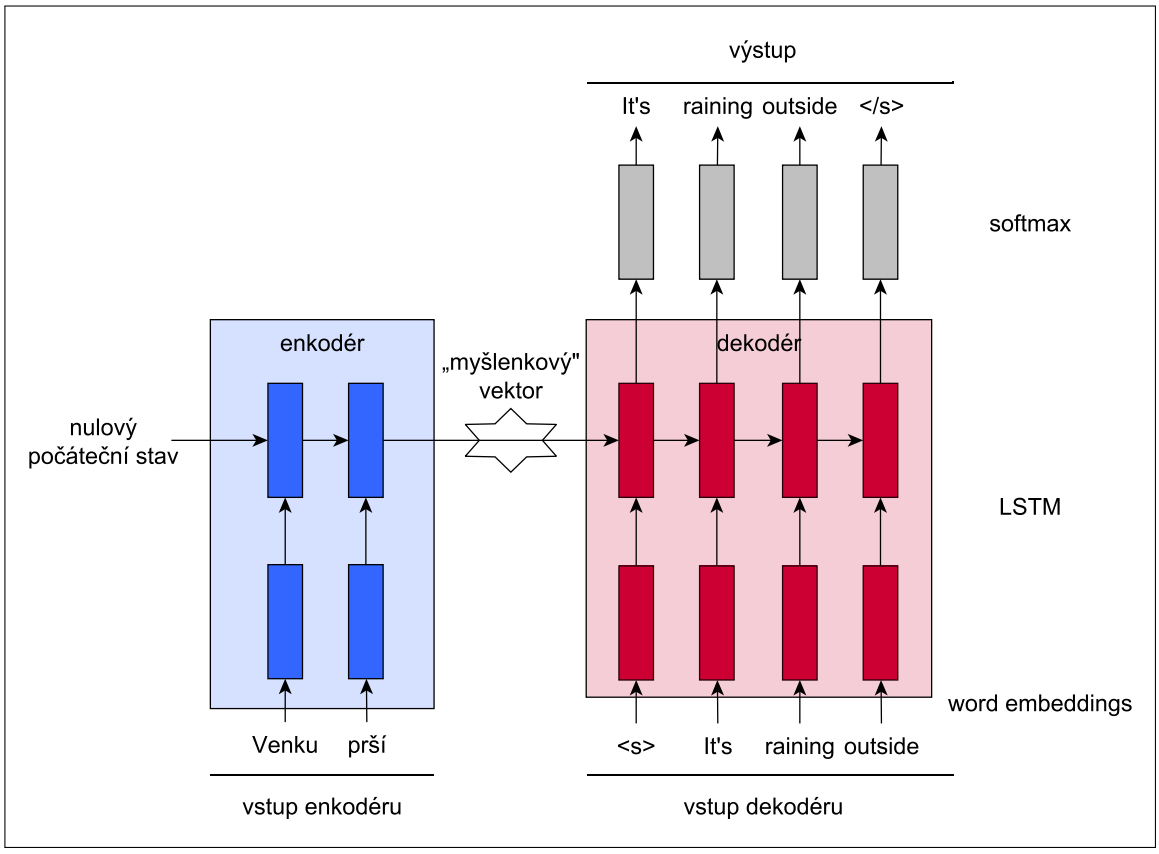
3.3.1 Průběh trénování a generování

Cílem jazykového modelu je předpovídat následující slovo ve větě. Při trénování se nejprve do enkodéru pošle výchozí věta, aby se získal inicializační stav pro dekodér. Do dekodéru, inicializovaného získaným stavem, se po jednotlivých slovech pošle správně přeložená věta, které předchází **startovací** symbol. Startovací symbol dekodéru říká, že má začít překládat. Při trénování se mu pak následně posílají korektní další slova, aby se zrychlilo učení. Tato metoda se nazývá „teacher forcing“ ([8]). Po naučení modelu, ve fázi generování, se pak do dekodéru posílají slova, která již sám vygeneroval. Proces je znázorněný na obrázku 3.8. Dekodér generuje tak dlouho, dokud nenarazí na **koncový** symbol, kterým je v době trénování zakončená každá očekávaná věta. Ve skutečnosti však výstupem dekodéru není přímo slovo, ale rozložení pravděpodobnosti přes všechna slova cílového slovníku získaného funkcí *softmax* v rovnici 3.26. Je několik možností jak z tohoto rozložení vybrat konkrétní slovo:

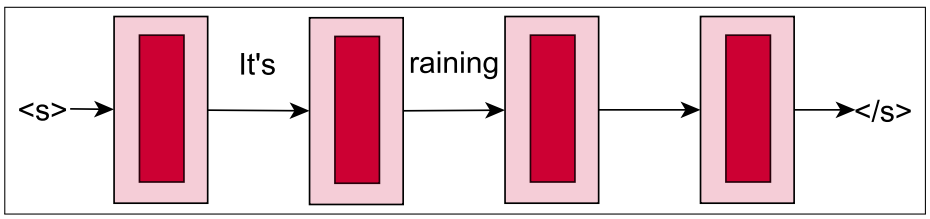
Náhodný výběr: Z rozložení pravděpodobnosti $P(E|F)$ se slovo vybere náhodně.

Chamtivý výběr: Chamtivý (greedy) výběr spočívá ve vybrání slova, které získalo největší pravděpodobnost – $\text{argmax}(P(E|F))$.

Paprskové prohledávání: Z anglického beam search, paprskové prohledávání najde n výstupů s největší pravděpodobností $P(E|F)$, které drží jako n možných výsledků nebo-li hypotéz. V každém kroku t se každá hypotéza rozšíří o další slovo a ze všech aktuálních hypotéz se zase vybere n nejslibnějších. Až jsou všechny hypotézy ukončeny koncovým symbolem <eos>, vybere se z nich ta s největší pravděpodobností, jako výsledek.



Obrázek 3.8: Enkodér-dekodér architektura. Enkodér obdrží větu na vstupu a vytvoří inicializační stav pro dekodér. Ten v době trénování dostává na vstupu správně přeloženou větu. V době generování na vstup dostává po jednom slova co sám vygeneroval a tím tak získává výslednou větu, dokud nenarazí na koncový symbol </s>. **[[líp popsat rozdíl inference a trénování. udělat tenhle odstavec obsáhlej a smysluplnej, kdyžtak doladit obrázek]]**



Obrázek 3.9: Inference

[[asi bych zkusil udelat primo dva obrazky, jeden na uceni a jeden na inference kde dekoder dostava svuj vlastni output]] [[lip popsat vsechny rovnice! vysvetlit jednotlivy pismenka]]

3.3.2 Metody optimalizace

V této sekci jsou popsány způsoby jakými lze zlepšit výkon seq2seq modelu.

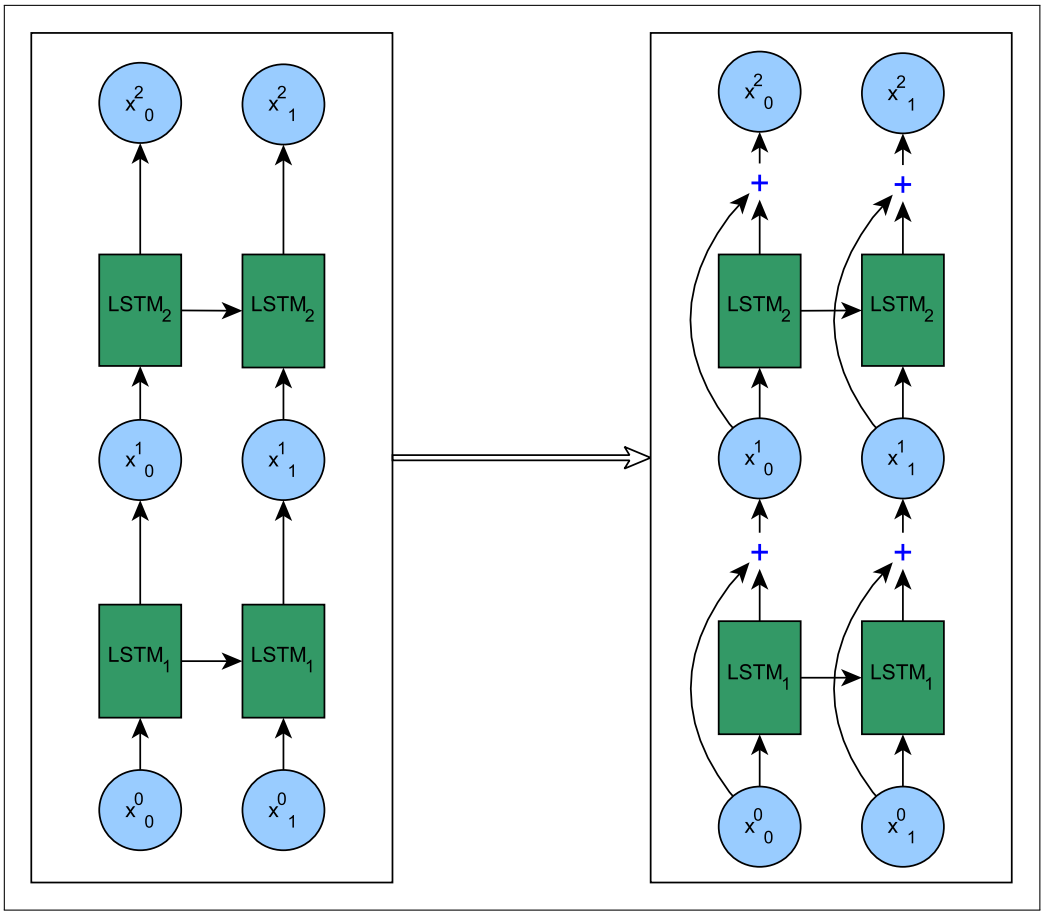
Převrácení vstupu: Článek [24] udává, že výrazným způsobem pomůže, když se slova ve vstupní sekvenci převrátí a do enkodéru se věta předává pozpátku. Pravděpodobně je to díky tomu, že závislosti co by běžně byly vzdálené – typicky poslední slovo ve vstupní větě a jeho přeložená varianta v přeložené větě – jsou si takhle blíž. Díky tomu se může model snáz a rychleji učit.

Obousměrný enkodér: Zatímco převrácení vstupu pomůže jen pokud jsou slova ve větách překládaných jazyků na podobných místech (což není pravda napříč všemi jazyky), tato varianta je spolehlivější. Místo jednoho enkodéru se použijí dva a každý z nich projde větu jedním směrem. Jejich výsledky se pak spojí do jednoho skrytého stavu h, kterým se již běžně inicializuje dekodér.

Hloubka sítí: Enkodér i dekodér jsou RNN a mohou obsahovat více skrytých vrstev (ať již základní varianty, LSTM nebo GRU). Článek [25] udává, že více vrstev může do určité hloubky pomoci. V práci jich použili 8 jak pro enkodér tak dekodér. Při použití většího množství již má model problém se úspěšně učit.

Dropout: Při trénování neuronových sítí může dojít k přetrénování – síť se naučí podávat správné výsledky pro trénovací data, ale bude mít malou nebo žádnou schopnost generalizce, tedy nebude fungovat nad jinými než trénovacími daty. Dropout ([23]), je regulační metoda používaná k předejití přetrénovaní. Metoda spočívá v náhodném zahazování výsledků některých neuronů v době trénovaní, čímž se snaží síť udělat více robustní, protože se síť nemůže spoléhat na výstupy konkrétních neuronů.

Reziduální propojení: V práci [25] doporučují při použití více vrstev LSTM použít takzvané reziduální propojení nebo zapojení mezi vrstvy. Podle jejich experimentů při použití většího počtu vrstev a běžném zapojení, začne být učení pomalé a složité, pravděpodobně kvůli explodujícímu a mizejícímu gradientu (3.2.2). Řešením je použití reziduálního propojení vrstev. Běžně by do vrstvy $LSTM_0$ vstupoval vstup x_0 a do vrstvy $LSTM_1$ vstup x_1, který je výstupem vrstvy $LSTM_0$. Při reziduálním propojení vstupuje da každé následující vrstvy výstup z vrstvy minulé sečtený dohromady se vstupem minulé vrstvy. Tedy do $LSTM_1$ vchází $x_1 + x_0$ (obrázek 3.10).

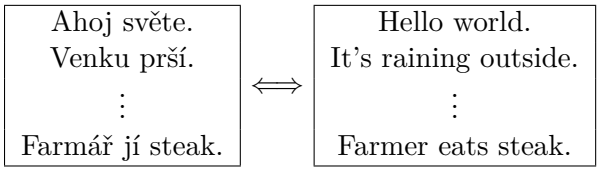


Obrázek 3.10: Rozdíl mezi normálním a reziduálním propojení pro zlepšení učení více vrstvých LSTM. Levý obrázek znázorňuje běžné zapojení zatímco na pravém je ukázka reziduálního propojení. Do každé (kromě první) vrstvy $LSTM_i$ vstupuje součet výstupu vrstvy minulé x_i sečtený se vstupem minulé vrstvy x_{i-1}.

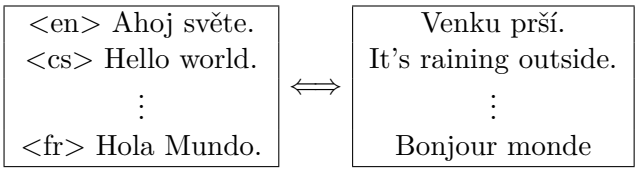
3.3.3 Překlad s jedním modelem mezi více jazyky

Doposud udávané informace se týkaly překladu z jednoho jazyka do druhého, tedy existují páry vět mezi dvěmi jazyky, enkodér se natrénuje na větách ve zdrojovém jazyce a dekodér se naučí překládat pomocí vět v cílovém jazyce.

Google ve své práci [10] ukazuje, že lze se stejným enkodér-dekodér modelem, bez jeho úprav, lze překládat mezi několika jazyky. Jediným rozdílem je přidání speciálního tokenu před zdrojové věty, který udává, do jakého jazyka se má daná věta překládat, znázorněno na obrázku 3.11. V práci se používá slovník wordpieces (3.1.4) natrénovaný a sdílený přes všechny použité jazyky.



(a) Příklad párů vět mezi dvěmi jazyky.



(b) Příklad párů vět se speciálním tokenem, který odlišuje do jakého jazyka má být věta přeložena. Model může překládat z vícero do vícero jazyků.

Obrázek 3.11: Obrázek ukazuje rozdíl mezi páry vět použitými při běžném překladu mezi dvěmi jazyky (3.11a) a při použití stejného modelu pro překlad mezi více jazyky (3.11b). Je potřeba přidat počáteční token, který pro enkodér označuje jazyk, do kterého se věta má přeložit.

Model může překládat v různých kombinacích, seřazeno od nejjednodušší po nejsložitější:

N:1 z více jazyků do jednoho

1:N z jednoho jazyka do vícero

M:N z více jazyků do vícero jazyků

Zajímavým zjištěním je, že model je schopný naučit se generovat překlady mezi kombinacemi jazyků, pro které nebyl explicitně natrénován, takzvané *zero-shot* překlady. Pokud se model tranzitivně natrénuje například na překlad jazykových kombinací Čeština⇒Angličtina a Angličtina⇒Francouzština, je následně schopný generovat relativně rozumné překlady mezi párem Čeština⇒Francouzština.

3.4 Vyhodnocování vlastností překladových systémů

[[https://en.wikipedia.org/wiki/Evaluation_of_machine_translation vsechny nebo jenom BLEU? muze to byt kdyztak nekde dole, pokud jenom BLEU?]]

3.4.1 BLEU

BLEU [19] nebo-li bilingual evaluation understudy, je algoritmus pro automatické hodnocení kvality překladu získané strojovým překladem vůči původnímu zdroji. BLEU je jedním z nejpopulárnějších způsobů hodnocení kvality systému mezi výzkumníky. Snaží se hodnotit takovým způsobem, aby čím větší skóre znamenalo, tím blíže se překlad blíží k něčemu co by přeložil profesionální lidský překladatel.

BLEU je počítáno pro jednotlivé věty, které porovnává s jedním nebo více referenčními překlady. Skóre se pak zprůměruje přes celý dataset. Algoritmus porovnává shody v n-gramech a výsledkem je skóre 0–1 respektive 0–100%.

3.5 Attention? nebo ne kdyz ji nepouziju

Kapitola 4

Implementace

Tato kapitola popisuje všechny autorem vytvořené a použité části. Sekce 4.1 je o výběru a předzpracování datasetů. Následující sekce 4.2 se zabývá vytvořením baseline systému, vůči kterému se porovnávají výsledky v kapitole 5. Poslední sekce této kapitoly (4.3) popisuje vytvořený překladový systém.

4.1 Datasety

Jako dataset (nebo korpus) se v této práci považují dva soubory. Každý ze souborů obsahuje věty v jednom jazyce. Na každém řádku souboru je jedna věta a ta svým významem odpovídá větě na stejném řádku v jazyce druhém. Dataset nese nějaký název (název souboru stejný pro oba jazykové soubory) a jako koncovku používá dvou písmenou zkratku jazyka. Pro lepší představu je přiložen obrázek 4.1.

exampleDataset.cs		exampleDataset.en
Ahoj světe.		Hello world.
Venku prší.	$\Longleftrightarrow$	It's raining outside.
⋮		⋮
Farmář jí steak.		Farmer eats steak.

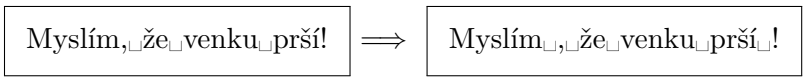
Obrázek 4.1: Ukázka datasetu. Dataset se jmenuje „exampleDataset“ a je rozdělen na český seznam vět (koncovka „cs“) a anglický seznam vět (koncovka „en“). Na každém řádku seznamu vět jednoho jazyka je jedna věta odpovídající si s větou na stejném řádku v jazyce druhém.

4.1.1 Předzpracování

Datasety obsahující zarovnané řádky vět v různých jazycích lze pořídit online[1]. Před použitím na trénování a vyhodnocování překládacího systému je však potřeba je ještě vhodným způsobem předpřipravit. Všechny použíte skripty jsou dostupné na githubu programu Moses [2]. Cílem je snížit velikost výsledných slovníků a zbavit se nevhodných vět.

[1] například na statmt.org/wmt17/translation-task.html a opus.lingfil.uu.se
[2] https://github.com/moses-smt/mosesdecoder

Tokenizace: Věty se rozdělí na jednotlivé tokeny oddělené mezerou. V případě běžných slov to znamená, že se nic nezmění. Oddělí se však například interpunkce. K tokenizaci se používá skript z nástroje Moses *tokenizer.perl*. Každý jeden token je ve výsledku jedno slovo ze slovníku a musí tak pro něj existovat jeho embedding nebo se převede na <unk> symbol.



Obrázek 4.2: Ukázka tokenizace. Věty se rozdělí po jednotlivých tokenech a každý z nich je oddělen mezerou. Pro lepší znázornění je v ukázce jako mezera použit znak „␣“.

Truecasing: Velká písmena na začátku vět se převedou na malá nebo se zachovají podle toho v jaké formě se slovo nejčastěji vyskytuje v celém datasetu. Velká písmena tak zůstanou jen tam kde je to běžná podoba slova (například u jmen). Díky tomu se sníží počet slov ve slovníku. Pro truecasing se používají skripty z nástroje Moses *train-truecaser.perl* a *truecase.perl*.

Vyčištění: Zahodí se prázdné či špatně zarovnané řádky. Dále se zkrátí věty na maximální délku 15 tokenů. Příliš dlouhé sekvence by znamenaly značnou zátěž na paměť a rychlost trénování překladového systému. Pro vyčištění je použit skript z nástroje Moses *clean-corpus-n.perl*.

Rozdělení slov na menší jednotky: Jak je popsáno v sekci 3.1.4, může být pro trénování výhodné nepoužívat jako nejmenší jednotku slova, ale jejich části. Pro aplikování BPE je použita knihovna *subword-nmt*[3]. Podle doporučení se BPE vytváří za dohromady nad datasety zdrojového i cílového jazyka. Zvolený počet *merge* operací je 15000. V sekci 5 je blíže popsáno na které datasety toto bylo aplikováno a jak se podle toho liší výsledky.

4.2 Baseline systém v Moses

Moses [12] je nástroj na vytváření statistických strojových překladových systémů. Vzniklý model bude sloužit jako baseline vůči kterému se porovnají výsledky implementovaného překladového systému (sekce 4.3). Kromě toho se také používají některé skripty z tohoto nástroje pro přípravu datasetů (sekce 4.1.1) a získání skóre BLEU. Konkrétní postup jeho přípravy je dostupný na stránkách Moses [4], byly použity výchozí nastavení (viz soubor *run_moses.sh*). **[[muzu odkazovat na prilozene soubory?]]**

[[líp popsat na čem jsem natrénoval baseline, možná až v experimentech a hodnocení]]

4.3 Překladový systém

Pro implementaci překladového systému byl zvolen jazyk Python[5] v jeho verzi 3.6. Na výběr bylo z několika vhodných knihoven/frameworků pro práci se strojovým učením:

[3] https://github.com/rsennrich/subword-nmt
[4] statmt.org/moses/?n=Moses.Baseline
[5] python.org

- Tensorflow – je open source knihovna, která původně vznikla v rámci výzkumného týmu Google Brain uvnitř společnosti Google. Tensorflow používá pro výpočty graf, kde jednotlivé uzly reprezentují operace a hrany reprezentují datové struktury (tensory). Tensorflow se stala velice populární v oblasti vývoje neuronových sítí.
- Theano – knihovna pro efektivní práci s mnoho rozměrnými poli. Využívá pole z hojně používané pythoní knihovny Numpy. Nedávno se knihovna dostala na verzi 1.0 a naráz s tím se ukončil její vývoj.
- CNTK – Cognitive Toolik je open-source nástroj deep learning od firmy Microsoft. Poskytuje API pro jazyky C#, C++ i Python. Pro výpočty také používá graf, kde listy reprezentují vstupní hodnoty nebo parametry a ostatní uzly reprezentují maticové operace.
- Keras – je pythoní knihovna poskytující vysoko úrovňové API pro deep learning. Je vysoce modulární a určená pro snadné prototypování. Knihovna běží nad backendem, který používá pro výpočty. Backend může být jedna z předchozích knihoven – Tensorflow, Theano nebo CNTK.

4.3.1 Balíček nmt

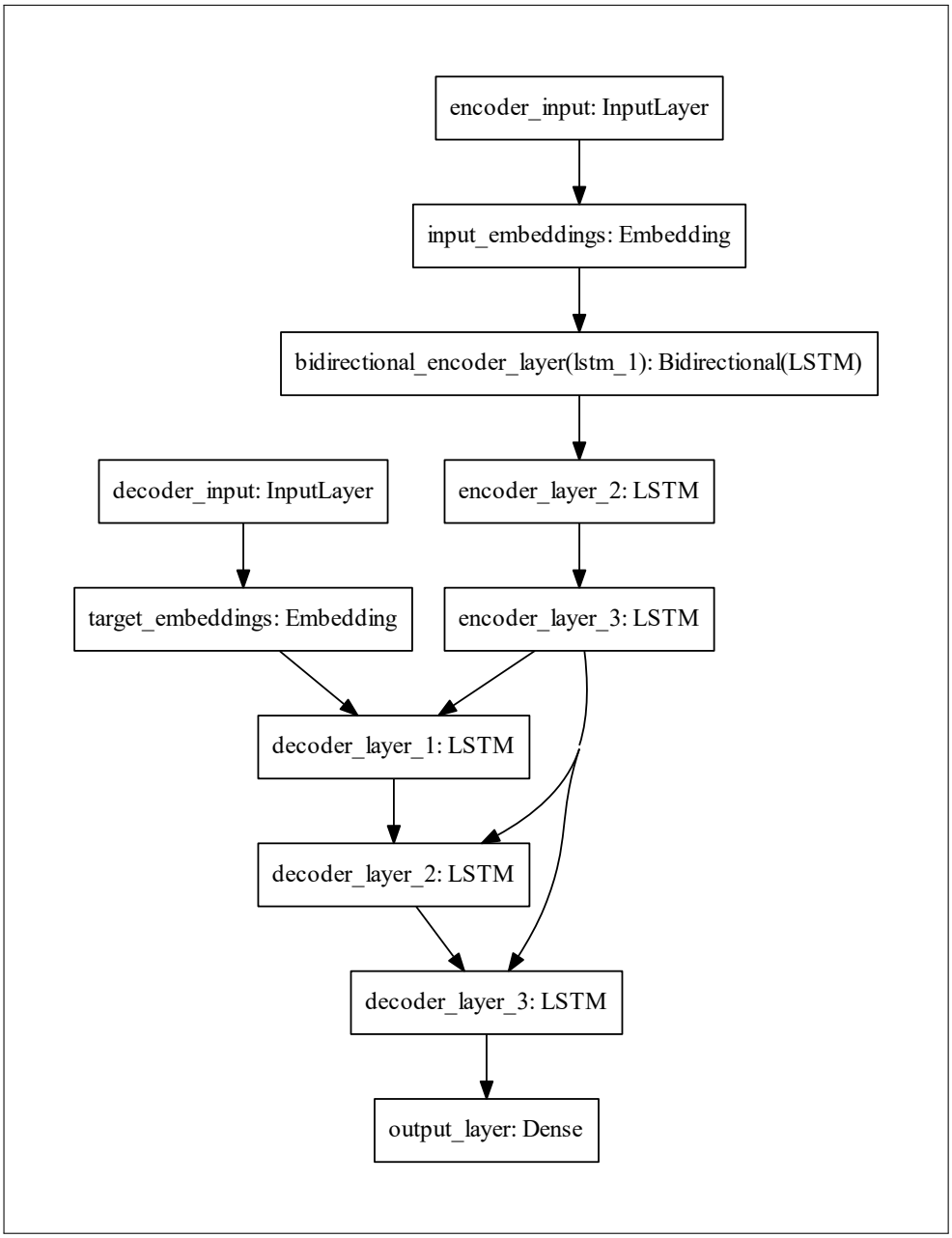
Překladový systém je naimplementován formou pythoního balíčku (knihovny), který je zveřejněn na githubu [6]. Pro jeho implementaci byla zvolena knihovna Keras [4] pro svůj jednoduchý a více intuitivní přístup a také pro množství návodů, které pro tuto knihovnu vznikají. Jako backend pro Keras je použit framework Tensorflow [1].

Slovníky výchozího a cílového jazyku se omezují na zvolenou maximální velikost a neznámá slova jsou nahrazeny symbolem <unk> (viz sekce 3.1.4). Knihovna umožňuje použití před trénovaných word embeddings (3.1.3) ve formátu *fastText*. Jsou implementovány a použity optimalizace popsané v sekci 3.3.2 – používá se obousměrný enkodér, je možné vytvořit model s různou hloubkou sítí a při trénování se používá dropout a teacher forcing (3.3.1). Pro co nejpřesnější generování předpovídaných vět je použito paprskové prohledávání (3.3.1).

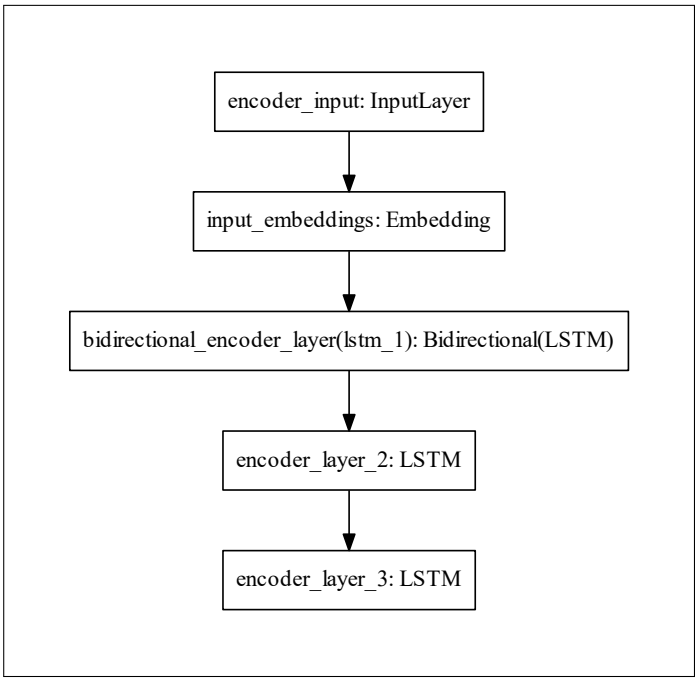
Pro potřeby trénování a překladu jsou pomocí knihovny Keras vystavěny tři modely, které spolu sdílejí vrstvy a jejich natrénované váhy – úplný model enkodér-dekodér použitý při trénovaní (obrázek 4.3), model enkodéru použitý při překladu pro získání inicializačních hodnot pro dekodér (obrázek 4.4) a model dekodéru použitý při překladu pro generování vět (obrázek 4.5).

[[udelat na to i novej model obrazek s residuals..muzu udelat bez vrstev kompletni a s vrsvama zvlast encoder/decoder at to nezabira tak moc mista]]

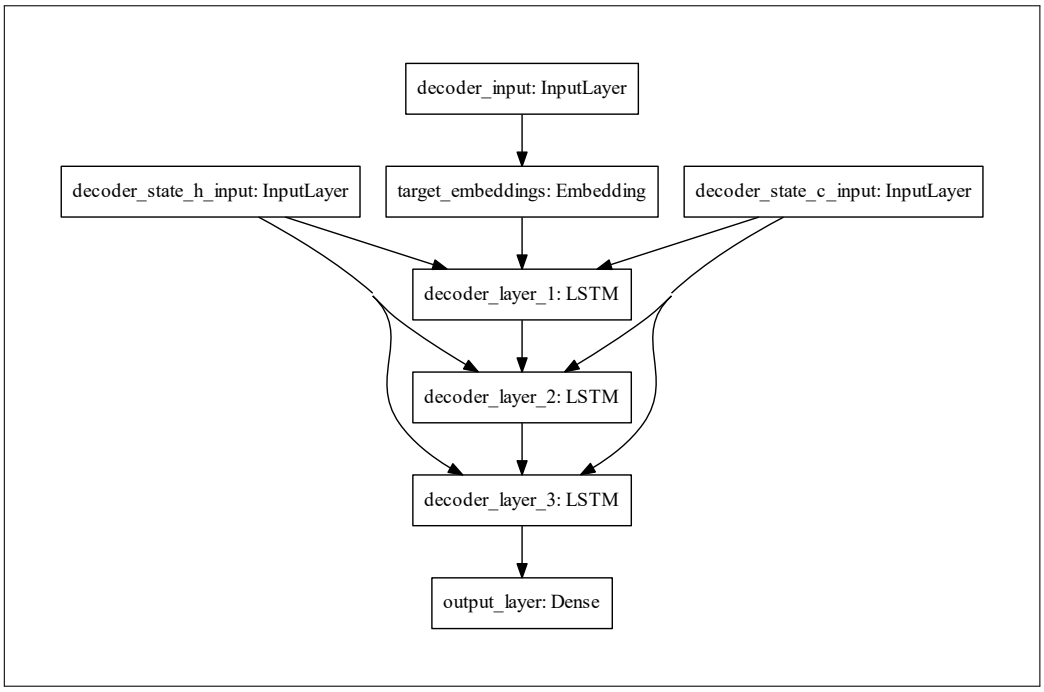
[6] https://github.com/jojkos/neural-machine-translation



Obrázek 4.3: Vrstvy a jejich propojení v enkodér-dekodér modelu použitém při trénovaní při použití 3 vrstev LSTM pro enkodér a 3 vrstev LSTM pro dekodér. Do *encoder_input* přicházejí sekvence ve výchozím jazyce. V *input_embeddings* se převedou do podoby embeddings vektoru. Následuje první vrstva enkodéru, která je obousměrná. Další vrstvy enkodéru jsou již jen dopředné. Do *decoder_input* přichází sekvence začínající startovacím tokenem <s> následované korektním překladem v cílovém jazyce, protože se používá teacher forcing. V *target_embeddings* se převedou do podoby embeddings vektoru. Následují vrstvy dekodéru, které jsou všechny inicializovány výsledným stavem enkodéru. Poslední vrstva *output_layer* s aktivační funkcí *softmax* vrací pravděpodobnosti pro všechna slova z cílového slovníku.



Obrázek 4.4: Vrstvy a jejich propojení v modelu enkodéru použitém při generování překladů při použití 3 vrstev LSTM.



Obrázek 4.5: Vrstvy a jejich propojení v modelu dekodéru použitém při generování překladů při použití 3 vrstev LSTM.

Hlavní třídou je *Translator*. Tato třída implementuje za pomocí knihovny Keras enkodér-dekodér architekturu (sekce 3.3). Enkodér i dekodér jsou vytvořeny z jedné vrstvy LSTM. Všechny vrstvy modelu jsou zachyceny na obrázku 4.3. Sekvence vstupující do enkodéru jsou převráceny podle 3.3.2.

Metodou *fit* se spustí trénování modelu **[[nekde vysvetlit pojmy jako epocha, batch size a vsechny tyhle dalsi veci..kde a jak?]]**

Stručný popis obsahu repozitáře `nmt`

třída Translator: Je hlavní třídou. Přijímá veškerá nastavení týkající se modelu, trénovací a testovací dataset, vytváří model a provádí trénink, překlad a jeho vyhodnocení.

Přehled hlavních metod:

- metoda `fit` zahajuje trénování
- metoda `translate_test_data` přeloží testovací dataset s pomocí natrénovaného modelu
- metoda `get_bleu_for_test_data_translation` vyhodnotí skóre BLEU pro vzniklý překlad

třída Dataset: Drží v sobě dataset v jeho tokenizované formě, tedy pole sekvencí jednotlivých vět. Řeší jeho načtení ze souboru a zpracováno do podoby vhodné pro trénování.

třída Vocabulary: Na základě všech sekvencí v datasetu daného jazyka vezme a drží v sobě x nejčastějších slov, se kterými pak model pracuje.

třída Candidate: Je pomocná třída použitá při výpočtu paprskového prohledávání 3.3.1. Drží v sobě hodnotu jednoho z aktuálně rozgerovaných kandidátních překladů.

třída SpecialSymbols: Obsahuje výčet speciálních symbolů použitých při trénování.

- *_PAD* pro zarovnávací nulu
- *_GO* pro startovací token
- *_EOS* pro koncový token
- *_UNK* pro neznámý token

třída Utils: Obsahuje pomocné metody pro úpravu dat, výpočty a pro volání přiložených skriptů jako je *SubwordNMT* a výpočet BLEU skóre.

knhovna SubwordNMT: Součástí repozitáře je knihovna *SubwordNMT* použitá pro aplikování BPE (4.1.1). Knihovna je přidaná jako *git submodule*.

testy: Repozitář *nmt* obsahuje sadu testů pokrývajících jeho funkcionalitu, převážně jednotlivých metod (unit testy) a celkového fungování trénování a překladu. Testy jsou implementovány ve frameworku *pytest*.

4.3.2 Rozdělení dat podle velikosti

Všechny sekvence při trénování v rámci jedné dávky musí mít stejnou délku. Věty však mají délku různou. To bývá řešeno tak, že se každá věta zarovná pomocí nul na délku nejdelší věty (obrázek 4.6). Na první pohled může být jasné, že pokud se všechny věty v

datasetu zarovnají na velikost nejdelší věty, způsobí to zbytečně větší vytížení paměti a delší trénování modelu.

„Venku dneska svítí slunce .“ „Ahoj světe .“	$\Longrightarrow$	[1, 2, 3, 4, 5] [6, 7, 5, 0, 0]

Obrázek 4.6: Příklad zarovnání sekvencí na stejnou délku. V levé tabulce jsou ukázány věty po tokenizaci, první věta má pět tokenů a druhá tři. V druhé tabulce jsou pak věty převedené do matic, kde číslo vyjadřuje pozici daného slova ve slovníku respektive v embeddings a 0 je zarovnání.

Tento problém je omezen pomocí rozdělení vět v datasetech do skupin podle jejich velikostí. Věty o podobné velikosti jsou shluknuty do jedné skupiny a tím se zamezí zbytečnému nárůstu požadavků na paměť. Dávky se pak při generování (4.3.3) berou vždy z jedné skupiny, tak aby všechny sekevence v jedné dávce byly stejně dlouhé.

4.3.3 Generování dávek

Protože se data při trénování upravují datasety do podoby velkých matic, není možné, kvůli paměťovým omezením, vyrobit jednu matici pro všechny data zaráz. Proto se dávky (batche) v průběhu trénování vytváří postupně, pomocí generátorové funkce. Ta před každou epochou (jedna epocha znamená, že modelem prošly všechny vstupní data), zamíchá vstupní data – správné míchání vstupních dat je důležité pro dobrou konvergenci modelu – a postupně generuje matice s jednotlivými dávkami. V případě, že je zapnuté dělení do skupin podle velikosti, probíhá míchání na úrovní skupin a dávek tak, aby se model nepřetrénoval v jednu chvíli na například krátké věty a v jiné části epochy zase na dlouhé. Je použit `random.seed(0)` pro opakovatelnost experimentů.

[[hodnoceni skore v beam search]]

Kapitola 5

Experimenty a vyhodnocení

V této sekci jsou prezentovány experimenty provedené s překladovým systémem. Systém byl natrénován pro překlad z češtiny (cs) do angličtiny (en). Jako trénovací dataset byl použit „news-commentary-v12.cs-en“[1]. Po vyčištění provedeném podle 4.1.1 obsahuje 132114 párů vět. Jako testovací dataset byl použit „newstest2017-csen“[2], který po vyčištění obsahuje 2841 párů vět. Oba tyto sety pochází z každoročně publikovaného překládacího úkolu konference WMT. Jak baseline systém vytvořený v nástroji Moses (sekce 4.2), tak systém prezentovaný v této práci (sekce 4.3) byly natrénovány a otestovány na těchto dvou datasetech.

Trénink probíhal 23 epoch. Jak enkodér, tak dekodér je jednovrsté LSTM s 256 jednotkami. Zvolený optimalizátor je RMSprop. Maximální počet různých tokenů ve výstupním jazyce je 30000 a ve výstupním 15000. Byly použity předučené word embeddings (varianta fastText), jak pro vstupní, tak výstupní jazyk, poskytnuté firmou Facebook [3].

Hodnotící metrikou je standardní skore BLEU. Pro jeho výpočet byl použit skript *multi-blue.pl* dodávaný s nástrojem Moses. Výsledky jsou zaznamenány v tabulce 5.1.

[[popis BLEU]] [[BPE x bez BPE]] [[hodnoceni pokud bylo pouzito BPE je udelano po jeho odstraneni - prevedeni na cela slova]]

[[ukládání modelu po každé epoše, jen nejlepší?]] [[early stopping]]

[[loss funkce je categorical_crossentropy, co vyzkouset ruzny? mozna popsat ruzny loss funkce v teorii?]] [[https://stackoverflow.com/questions/34518656/how-to-interpret-loss-and-accuracy-for-a-machine-learning-model ze sleduju val_loss ZAJIMA ME LOSS, acc se pocita zahadne a loss ukazuje proste jak moc si dobre zrovna ten model vede]]

	baseline systém	prezentovaný systém
BLEU skóre	14.0	2.03

Obrázek 5.1: Porovnání výsledků baseline systému vytvořeném v nástroji Moses a překladového systému, který je výsledkem této práce.

[1] http://data.statmt.org/wmt17/translation-task/training-parallel-nc-v12.tgz
[2] http://data.statmt.org/wmt17/translation-task/test.tgz
[3] https://github.com/facebookresearch/fastText/blob/master/pretrained-vectors.md

[[napsat nejaky ocekavani, jakoze BPE bude lepsi, vetsi hloubka (podle google do urcite miry) bude lepsi, vetsi dataset bude lepsi atd (necim to podlozit vzdycky) a pak to podle toho overit.]]

5.0.1 Použité datasety

ocitovat WMT, popsat jak jsem je vytvořil a jakej na co použiju, nad kterým se natrénuje moses

[[popsat jaky vsechny pouzivam datasety, kde jsem je ziskal (citace) jak jsem je presne predzpracoval a proc. KOLIK EPOCH TO BEZELO, idealne ze to bezelo tak dlouho, dokud se to nezhorsovalo pro testovaci dataset (dev dataset) a uklada se vzdycky jen nejlepsi model po kazde epose]]

[[napsat, že jsem je prováděl na školním clusteru s pomocí main.py skriptu, který využívá mojí nmt knihovnu, ideálně někde zmínit, že je možný to pouštět po epochách a rozkládat na víc puštění nebo obnovit po chybě]]

[[navrhnout testy, rovnou si je sem pokaždé vypsat a pak až je pustit, celý je dát do příloh nějak]] [[vždycky když se změni struktura modelu, přidat obrázek vyprintovanej z kerasu]] [[napsat ze bleu se pocita az po oddelani subwords]] [[moses je testovanej ne dev setem ale test setem]]

5.0.2 1. experiment

popis datasetů, parametrů grafy výsledek

5.0.3 2. experiment

[[zkusit dropout a pak ten rekurentni dropout primo v LSTM]]

kdyz by se mi chtelo, tak to pro kazdej projet s nekolika ruznyma beam size a pak uz treba pouzivat jen ten nejlepsi

1. nejakej zakladni bez BPE (final1)
2. stejnej bez BPE, ale s vyrazne vetsim slovnikem
3. stejnej, ale s BPE a slovnikem jak ten prvni (oduvodnit pocet bpe operaci, ze 15k je snesitelna velikost slovniku)
4. BPE bude nejlepsi? kdyz jo, tak s vic jednotkama
5. s tim poslednim (protoze je nejlepsi) vyzkouset dropout 0.1, 0.2, 0.3
6. 2 vrstvy pro encoder (s nejlepsim dropautem, takze 0.1)
7. 2 vrstvy pro decoder
8. 2 vrstvy pro oboje
9. 4 vrstvy pro oboje (finalni model?)

[[google rika ze to dobre funguje bez residual connections do 4 layeru]]

Kapitola 6

Závěr

Práce popisuje komponenty potřebné pro vytvoření překladového systému s pomocí neuronových sítí. Za pomocí těchto komponent byl vytvořen systém realizovaný pythoním balíčkem *nmt*. Byly zvoleny trénovací a testovací data s kterými byl tento systém otestován. Hodnotícím kritériem je skore BLEU, které pro testovací dataset newtest2017 vyšlo 2.03. V porovnání se skóre 14.0, které vyšlo pro systém natrénovaný nástrojem Moses to není velmi úspěšný výsledek. Systém bude potřeba rozšířit a vylepšit, tak aby fungoval výrazně lépe.

Je několik vhodných rozšíření systému, které by mohly pomoci dosáhnout lepších výsledků. Použití více, případně i větších, vrstev LSTM. Na místo invertování vstupů enkodéru, je možné první vrstvu enkodéru udělat obousměrnou (viz 3.3.2). Dále by bylo vhodné přidat paprskové vyhledávání na místo hladového pro určování slov při generování. Dosavadní implementace systému vykazuje velký problém s neznámými (*UNK*) slovy. Tento problém by se mohl vyřešit pomocí tzv. *attention* mechanismu a přenášení neznámých slov v nezměněné podobě do výsledného překladu.

[[attention]] [[knihovna neni nutne specificka pro nmt, ale zavisla na datech, upravil bych api na vic genericky a zkusil natrenovat chatbota]] [[ze jsem se hodne zameroval na experimenty a data, ale chtel bych vylepsit dekompozici knihovny a jeji api a distribuovat na PyPI]]

Literatura

[1] Abadi, M.; Barham, P.; Chen, J.; aj.: TensorFlow: A system for large-scale machine learning. *CoRR*, ročník abs/1605.08695, 2016, 1605.08695.
URL http://arxiv.org/abs/1605.08695

[2] Bengio, Y.; Simard, P.; Frasconi, P.: Learning Long-term Dependencies with Gradient Descent is Difficult. *Trans. Neur. Netw.*, ročník 5, č. 2, Březen 1994: s. 157–166, ISSN 1045-9227, doi:10.1109/72.279181.
URL http://dx.doi.org/10.1109/72.279181

[3] Bojanowski, P.; Grave, E.; Joulin, A.; aj.: Enriching Word Vectors with Subword Information. *CoRR*, ročník abs/1607.04606, 2016, 1607.04606.
URL http://arxiv.org/abs/1607.04606

[4] Chollet, F.; aj.: Keras. https://github.com/keras-team/keras, 2015.

[5] Chung, J.; Gülçehre, Ç.; Cho, K.; aj.: Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *CoRR*, ročník abs/1412.3555, 2014, 1412.3555.
URL http://arxiv.org/abs/1412.3555

[6] Elman, J. L.: Finding Structure in Time. *Cognitive Science*, ročník 14, č. 2, 1990: s. 179–211, ISSN 1551-6709, doi:10.1207/s15516709cog1402_1.
URL http://dx.doi.org/10.1207/s15516709cog1402_1

[7] Gers, F. A.; Schmidhuber, J. A.; Cummins, F. A.: Learning to Forget: Continual Prediction with LSTM. *Neural Comput.*, ročník 12, č. 10, Říjen 2000: s. 2451–2471, ISSN 0899-7667, doi:10.1162/089976600300015015.
URL http://dx.doi.org/10.1162/089976600300015015

[8] Goyal, A.; Lamb, A.; Zhang, Y.; aj.: Professor Forcing: A New Algorithm for Training Recurrent Networks. 2016, s. 4601–4609.
URL http://papers.nips.cc/paper/6099-professor-forcing-a-new-algorithm-for-training-recurrent-networks.pdf

[9] Hochreiter, S.; Schmidhuber, J.: Long Short-Term Memory. *Neural Comput.*, ročník 9, č. 8, Listopad 1997: s. 1735–1780, ISSN 0899-7667, doi:10.1162/neco.1997.9.8.1735.
URL http://dx.doi.org/10.1162/neco.1997.9.8.1735

[10] Johnson, M.; Schuster, M.; Le, Q. V.; aj.: Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation. *CoRR*, ročník abs/1611.04558, 2016, 1611.04558.
URL http://arxiv.org/abs/1611.04558

[11] Józefowicz, R.; Vinyals, O.; Schuster, M.; aj.: Exploring the Limits of Language Modeling. *CoRR*, ročník abs/1602.02410, 2016, 1602.02410.
URL http://arxiv.org/abs/1602.02410

[12] Koehn, P.; Hoang, H.; Birch, A.; aj.: Moses: Open Source Toolkit for Statistical Machine Translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, ACL '07, Stroudsburg, PA, USA: Association for Computational Linguistics, 2007, s. 177–180.
URL http://dl.acm.org/citation.cfm?id=1557769.1557821

[13] Luong, M.-T.: *NEURAL MACHINE TRANSLATION*. Dizertační práce, STANFORD UNIVERSITY, 2016.
URL https://github.com/lmthang/thesis

[14] Mikolov, T.; Chen, K.; Corrado, G.; aj.: Efficient Estimation of Word Representations in Vector Space. *CoRR*, ročník abs/1301.3781, 2013, 1301.3781.
URL http://arxiv.org/abs/1301.3781

[15] Mikolov, T.; Sutskever, I.; Deoras, A.; aj.: Subword Language Modeling with Neural Networks. In *Subword Language Modeling with Neural Networks*, 2011.
URL http://www.fit.vutbr.cz/~imikolov/rnnlm/char.pdf

[16] Mikolov, T.; Yih, W.-t.; Zweig, G.: Linguistic Regularities in Continuous Space Word Representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Association for Computational Linguistics, 2013, s. 746–751.
URL http://www.aclweb.org/anthology/N13-1090

[17] Neubig, G.: Neural Machine Translation and Sequence-to-sequence Models: A Tutorial. *CoRR*, ročník abs/1703.01619, 2017, 1703.01619.
URL http://arxiv.org/abs/1703.01619

[18] Olah, C.: Understanding LSTM Networks. 2015, [Online; navštíveno 3.12.2017].
URL http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[19] Papineni, K.; Roukos, S.; Ward, T.; aj.: BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, s. 311–318, doi:10.3115/1073083.1073135.
URL https://doi.org/10.3115/1073083.1073135

[20] Pennington, J.; Socher, R.; Manning, C. D.: GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, s. 1532–1543.
URL http://www.aclweb.org/anthology/D14-1162

[21] Ruder, S.: An overview of gradient descent optimization algorithms. *CoRR*, ročník abs/1609.04747, 2016, 1609.04747.
URL http://arxiv.org/abs/1609.04747

[22] Sennrich, R.; Haddow, B.; Birch, A.: Neural Machine Translation of Rare Words with Subword Units. *CoRR*, ročník abs/1508.07909, 2015, 1508.07909.
URL http://arxiv.org/abs/1508.07909

[23] Srivastava, N.; Hinton, G.; Krizhevsky, A.; aj.: Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.*, ročník 15, č. 1, Leden 2014: s. 1929–1958, ISSN 1532-4435.
URL http://dl.acm.org/citation.cfm?id=2627435.2670313

[24] Sutskever, I.; Vinyals, O.; Le, Q. V.: Sequence to Sequence Learning with Neural Networks. *CoRR*, ročník abs/1409.3215, 2014, 1409.3215.
URL http://arxiv.org/abs/1409.3215

[25] Wu, Y.; Schuster, M.; Chen, Z.; aj.: Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR*, ročník abs/1609.08144, 2016, 1609.08144.
URL http://arxiv.org/abs/1609.08144

[[presny sekvence prikazu pro pripravu datasetu]]
[[mozna i presny prikazy pro jednotlivy testy?]]
sge runner? moses runner?