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

SEMESTRÁLNÍ PROJEKT
TERM PROJECT

AUTOR PRÁCE **JONÁŠ HOLCNER**
AUTHOR

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

BRNO 2018

Abstrakt

Do tohoto odstavce bude zapsán výtah (abstrakt) práce v českém (slovenském) jazyce.

Abstract

Do tohoto odstavce bude zapsán výtah (abstrakt) práce v anglickém jazyce.

Klíčová slova

Sem budou zapsána jednotlivá klíčová slova v českém (slovenském) jazyce, oddělená čárkami.

Keywords

machine translation, neural machine translation, neural networks, recurrent neural networks, LSTM, encoder, decoder, encoder-decoder model, sequence to sequence, seq2seq, keras, tensorflow, moses, bleu, attention, bi-directional encoder

Citace

HOLCNER, Jonáš. *Strojový překlad pomocí umělých neuronových sítí*. Brno, 2018. Semestrální projekt. 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 tento semestrální vypracoval samostatně pod vedením pana Igora Szökeho. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jonáš Holcner
28. prosince 2017

Poděkování

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

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 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 encoder-decoder architekturou.

V kapitole 2 je naformálně nastíněn návrh a cíl této práce. V následující kapitole 3 jsou pak rozebrány důležité pojmy a teorie ze kterých je tato práce vystavěna. 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 [21]. 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 neuronovnou 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 sitě 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.

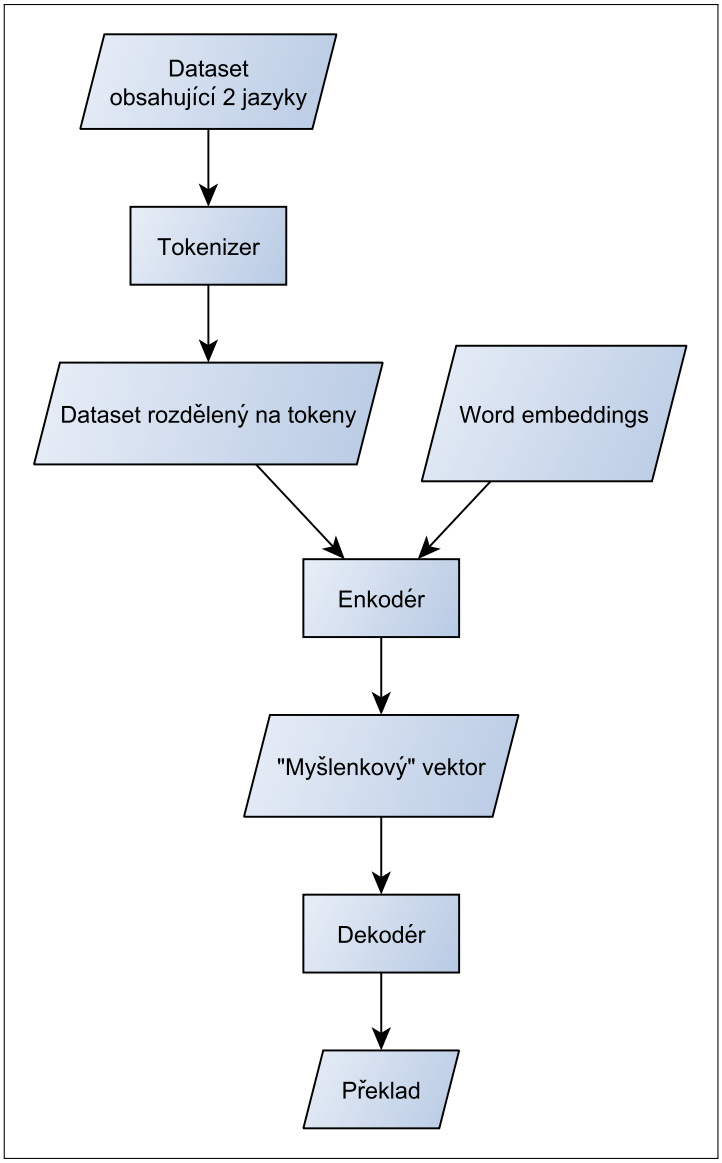
Slovník: Slovník se vytvoří jako seznam n nejčastějších slov v datasetu ve vstupním a cílovém jazyce. Čím je slovník menší, tím se zmenší výpočetní požadavky, ale na druhou stranu je potřeba vyřešit při trénování a překladu problém se slovy nevyskytujícími se ve slovníku (popsáno v sekci 3.1.4).

Word Embeddings: Obecně je možné vytvářet jazykové modely, které generují text po písmenech, částech slov nebo po slovech [15]. V této práci se bude pracovat s celými slovy (tokeny). 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 [20]) s použitím encoder-decoder architektury. Na rozdíl od starších statistických metod překladu, kde se překládalo po frázích, moderní překlad pomocí neuronových sítí

[1] translate.google.cz

probíhá po celých sekvencích (větách). Nejprve enkodér vezme word embedding na 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 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 [17].

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 nacvičen.

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-linearání 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 zakódovné 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 „žená“ k této dočasné hodnotě se pak získá ženská varianta krále – královna.

[[taky dobrej popis nlm a embeddings (distributed representation) http://www.scholarpedia.org/article/Neural_net_language_models]]

[[popis toho že řeší problémy co má n-gram (jen pár dozadu, nevím jak přesně?) a log-linear (díky nečemu (embeddings?) to vyloučí špatný kombinace jako kráva žere krávu nebo co (viz tutorial)]]

Zmíním co existuje za druhy, lehce jejich rozdíly a vznik (co jsou zač) a pak se víc rozepíšu o fasttextu, protože to je ten co jsem použil (rovnice a kdesi cosi)

bengio neural net language models[3]

Existuje několik moderních variant výpočtů word embeddings:

- word2vec [14]
- glove [19]
- fasttext [5]

Transfer learning: Transfer learning je způsob využití znalostí získaných někde jinde pro jiný problém. Word embeddings je možné buďto získat v průběhu učení modelu nebo použít už připravené, předučené, které jsou k dispozici online od firem jako je Google nebo Facebook. Díky tomu může model získat více znalostí o jednotlivých slovech a celkový výsledek může být výrazně lepší.

3.1.4 Zpracování neznámých slov

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. Také může být vhodné omezit celkový počet slov se kterými se bude model trénovat pro zlepšení výkonu. Práce [17] uvádí tři běžné způsoby jak se vypořádat s takovými **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 vyskytli a model se tedy mohl naučit všechny znaky.

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 $P_{unk}(e_t) = 1/|V_{all}|$.

Přídáním 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. **[[odkázat se sem ze sekce kde bude napsaný že tohle je varianta co používám]]**

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 3.2.4) a GRU (sekce 3.2.5). Sekce vychází z práce [13], práce [17] a článku [18].

RNN (článek [8]) 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 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. Dopředná neuronová síť pracuje vždy s aktuální hodnotou x na vstupu, pro kterou pomocí vah W získá výstup y (rovnice 3.3).

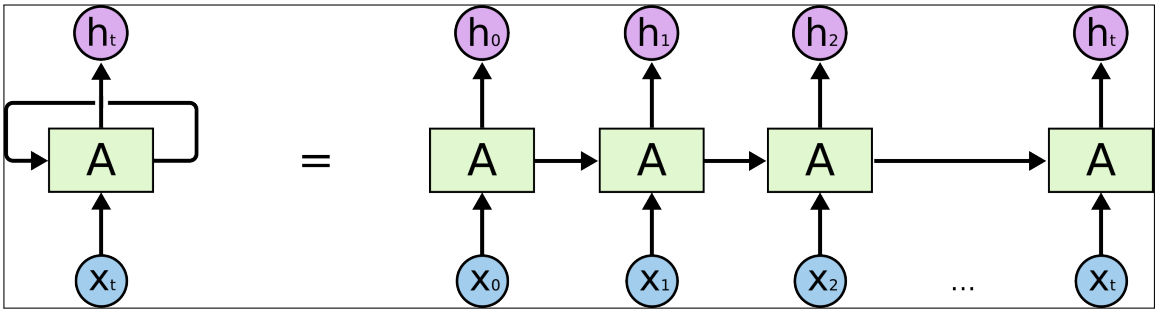
$$y = f(x, W) \tag{3.3}$$

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.

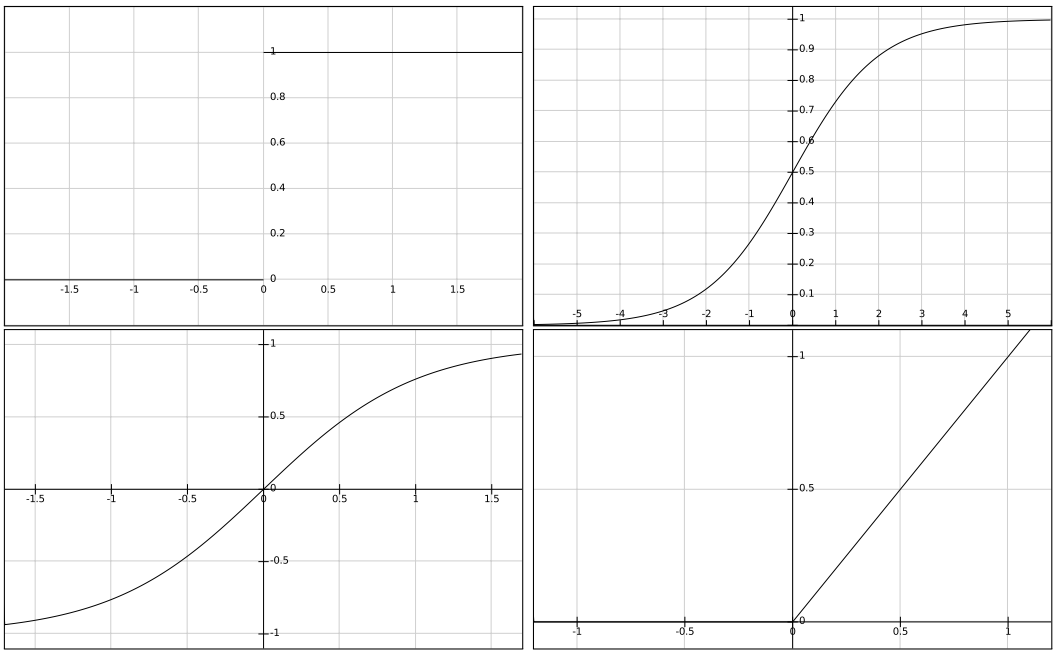
$$h_t = \begin{cases} f(W_{xh}x_t + W_{hh}h_{t-1} + b_h) & \text{pokud t} \geq 1, \\ 0 & \text{jinak.} \end{cases} \tag{3.4}$$

[1] z anglického recurrent neural network



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].

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 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 3.2.3.

[[doplnit rovnice (patrne z tutorialu) a vysvětlení jak se to aplikuje dál, stejně tak obrázky s unrolled rnn a popisem toho jak zachovává nějakou informaci (třeba že podstatné jméno je mužské) skrze jednotlivé kroky (i když ne úplně přes vzdálené a tím se dostanu k long term dependencies)]]

3.2.1 Trénování

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

3.2.2 Gradient descent methods

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

[[z tutorialu, sekce 6.5 strana 35..nejak popsat batch, minibatch, online batch, sentence padding a masking]]

3.2.3 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 [4]).

[[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* **[[kde je popsaná]]** 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 tak žádný efekt na úpravu vah nebo jimi pohne příliš a tak zaviní špatné učení se sítě.

Obrázek 3.5: **[[ukázka problémů s gradientem, něco jako figure 16 v 6.3 tutorialu]]**

Jako možné řešení těchto problémů vznikla varianta rekurentní sítě LSTM (sekce 3.2.4).

3.2.4 LSTM

Long short term memory, dále LSTM, (původní článek [10] a varianta LSTM s forget gate, která se zde používá [9]) 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 představut 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 3.6). Rovnice jsou následující:

$$u_t = tanh(W_{xu}x_t + W_{hu}h_{t-1} + b_u) \tag{3.10}$$
$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \tag{3.11}$$
$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \tag{3.12}$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \tag{3.13}$$
$$c_t = i_t \odot u_t + f_t \odot c_{t-1} \tag{3.14}$$
$$h_t = o_t \odot tanh(c_t) \tag{3.15}$$

RNN má pouze skrytý stav h. LSTM má navíc ještě paměťovou buňku c (rovnice 3.14). Protože gradient této buňky je právě jedna, netrpí tak LSTM problémy ze sekce 3.2.3 a mohou tak v ní být zachyceny i vzdálené závislosti.

Rovnice 3.10 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 3.11). 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.

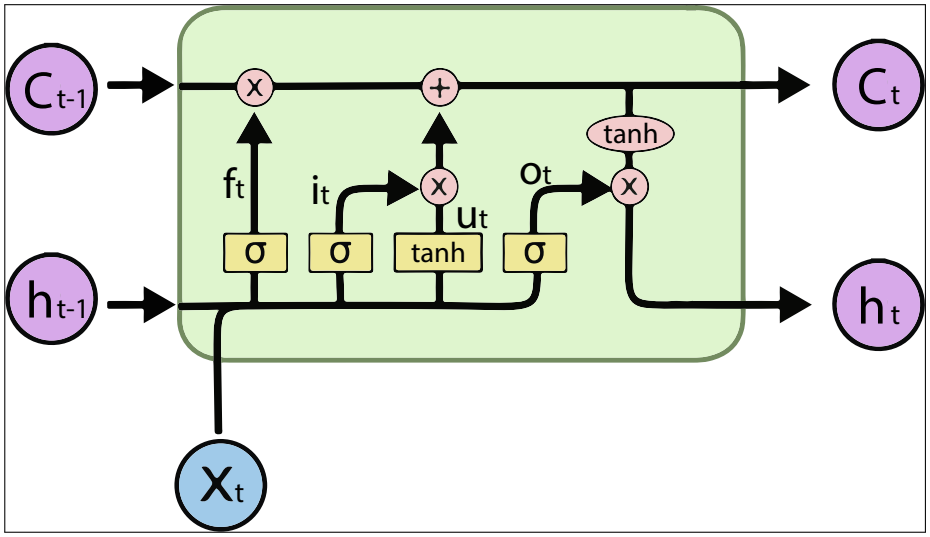
Dalším krokem je vyhodnocení toho co se má přidat do paměti. Nejdřív vstupní brána nebo-li input gate (rovnice 3.12) rozhodne, které hodnoty se změní nebo přidají. V návaznosti na minulý příklad by síť mohla chtít uložit aktuální rod nalezeného podstatného jména. Update funkce (rovnice 3.10) vyhodnotí jaké hodnoty se mají přidat.

Následuje aktualizace paměti c_t (rovnice 3.14. 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 3.13 a v rovnici 3.15 se získá výsledek.

Pravděpodobnosti jazykového modelu se získají rovnicí:

$$p_t = softmax(W_{hs}h_t + b_s) \tag{3.16}$$



Obrázek 3.6: Jeden časový úsek LSTM. h je skrytý stav, c je paměťová buňka a x je vstup. Vnitřní struktura koresponduje s rovnicemi 3.10 až 3.15. Obrázek převzat z [18], upraven.

3.2.5 GRU

LSTM (sekce 3.2.4) je dobrým řešením pro problémy ze sekce 3.2.3. 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 [7]), která je o něco jednodušší a proto je možné ji použít pro úsporu výkonu.

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \tag{3.17}$$
$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \tag{3.18}$$
$$\tilde{h}_t = tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h) \tag{3.19}$$
$$h_t = (1 - z_t)h_{t-1} + z_t\tilde{h}_t) \tag{3.20}$$

GRU má pouze dvě brány a skrytý stav h. Nový stav se počítá v rovnici 3.20 interpolací mezi minulým stavem h_{t-1} a kandidátem na nový stav $\tilde{h}_t$ upravený hodnotou **update** brány (rovnice 3.18). Kandidát se získá v rovnici 3.19, která je podobná update funkci z RNN (rovnice 3.4), ale je upravena hodnotou **resetovací** brány (rovnice 3.17).

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 [20]) 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 **encoder-decoder**. 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. Generování skutečně neprobíhá po jednotlivých slovech, ale celých větách, jak udává název.

$$m_t^{(f)} = M_{f_t}^{(f)} \tag{3.21}$$

$$h_t^f = \begin{cases} RNN^{(f)}(m_t^{(f)}, h_{t-1}^{(f)}) & \text{pokud t} \geq 1, \\ 0 & \text{jinak.} \end{cases} \tag{3.22}$$

$$m_t^{(e)} = M_{e_{t-1}}^{(e)} \tag{3.23}$$

$$h_t^e = \begin{cases} RNN^{(e)}(m_t^{(e)}, h_{t-1}^{(e)}) & \text{pokud t} \geq 1, \\ h_{|F|}^f & \text{jinak.} \end{cases} \tag{3.24}$$

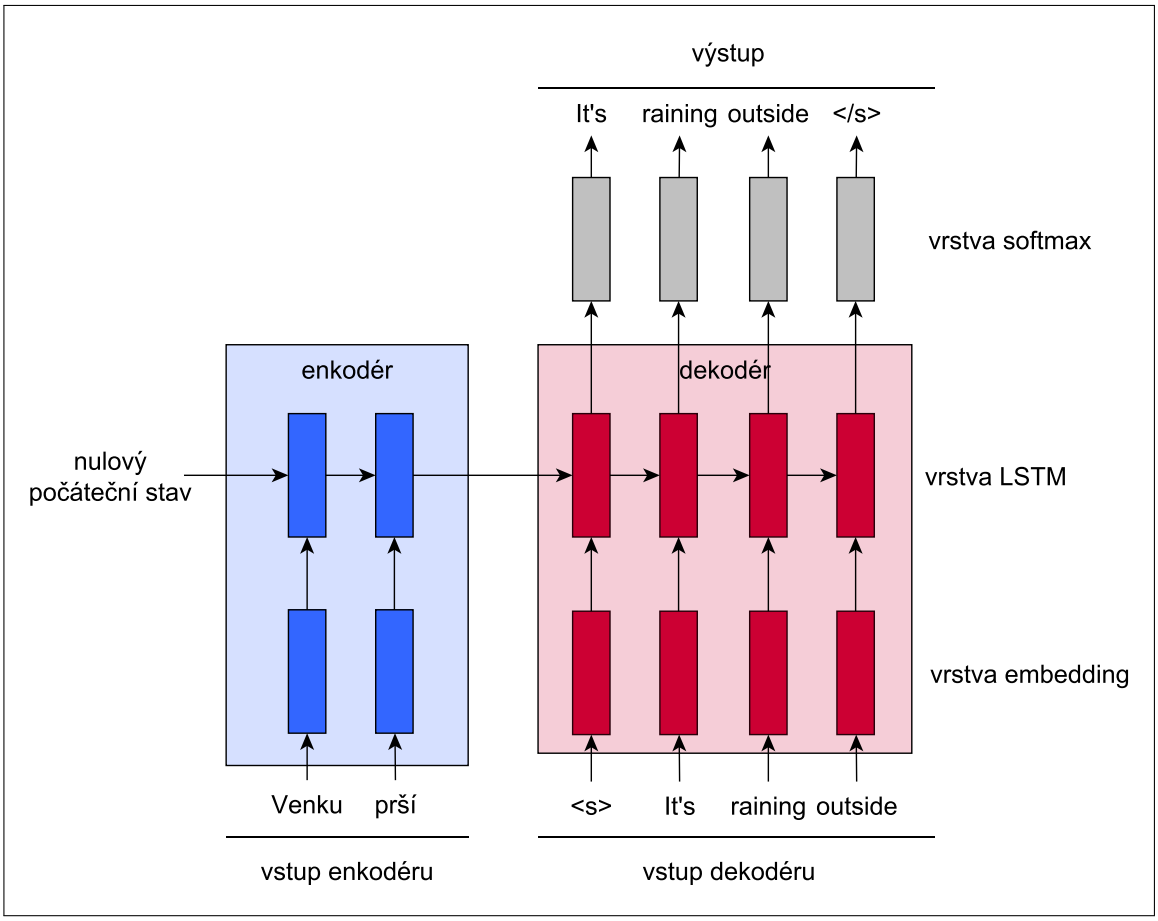
$$p_t^{(e)} = \text{softmax}(W_{hs} h_t^{(e)} + b_s) \tag{3.25}$$

Pro každé slovo v čase t ze vstupní sequence F se vyhledá jeho embedding (rovnice 3.21). Následně se v rovnici 3.22 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 3.23). 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 **startovní** token <s>. Rovnice pro výpočet skrytého stavu dekodéru (3.24) 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 3.25).

3.3.1 Generování výstupu

Cílem jazykového modelu je předpovídat následující slovo ve větě. Aby model věděl kdy má začít a ukončit predikci, používají se speciální **počáteční a koncový symbol** (<s>, </s>). Vstupní věta (sekvence) x by například mohla být $x = \{$<s>, Venku, sedí, kočka$\}$. Věta y generovaná modelem, by pak mohla být například tatáž věta, ale posunutá o jeden časový úsek dopředu – $y = \{$Venku, sedí, kočka, </s>$\}$. V ilustraci [[...]] je názorná ukázka.



Obrázek 3.8: **[[Obrázek jako je v nmt thesis strana 19, figure 2.3, VYSVĚTLIT ROZDÍL MEZI TRÉNOVÁNÍM (techer forced learning nebo jak se to jmenuje) A INFERENCI]]**

Jak již bylo řečeno, dekodér je inicializovaný koncovým stavem enkodéru $h^f_{|F|}$. Jako první vstup obdrží dekodér startovní symbol <s> a předpoví první slovo na výstup. Vygenerované slovo se v dalším kroku t použije jako aktuální vstup dekodéru. Takto probíhá generování dokud dekodér nedá na výstup další speciální symbol **<eos>** (end of sentence), který značí

konec věty. Těmito symboly je při trénování sítě potřeba obalit každou větu, tak jak je vidět i na obrázku **??**. Ve skutečnosti však výstupem dekodéru není přímo slovo, ale rozložení pravděpodobnosti přes všechna slova cílového slovníku získaného funkcí *softmax* v rovnici 3.25. 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 – 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 ukončený koncovým symbolem <eos>, vybere se z nich ta s největší pravděpodobností, jako výsledek.

[[možnost obrázku vysvětlujícím jednotlivé způsoby výběru – trojobrázek jako pro sigmoidy..?]]

[[lépe popsat beam search, ale az v lete asi, viz tutorial]]

3.3.2 Metody optimalizace

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

Převrácení vstupu: Článek [20] udává, že výrazným způsobem pomůže, když se slova ve vstupní sekvenci převrátí a do enkodéru se věta předává pozpátku (obrázek ..). **[[stejnej obrázek jak u seq2seq, ale s obráceným vstupem nebo takovej jakej je v tutorialu figure 24]]** 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 encodér: 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 [21] 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.

Kapitola 4

Návrh/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

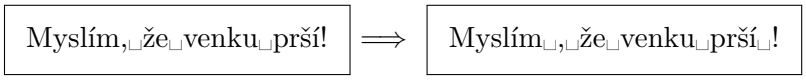
[[pripadne ocitovat opus.nlpl.eu]]

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 prázdně či špatně zarovnané řádky. Dále se zkrátí věty na maximální délku 40 **[[bude to napsany tady fakt? nekde jinde jeste?]]** tokenů. Příliš dlouhé sekvence by znamenaly značnou zátěž na paměť a rychlost trénování překladového systému. Pro vyčištění se používá skrpt z nástroje Moses *clean-corpus-n.perl*.

[[Jak rozlišit návrh a realizaci?]]

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). Baseline systém je nacvičen a otestován na stejných datech jako výsledný překladový systém. Konkrétní postup jeho přípravy je dostupný na stránkách Moses [3], byly použity výchozí nastavení.

4.3 Překladový systém

Pro implementaci překladového systému byl zvolen jazyk Python[4] v jeho verzi 3.6. Na výběr bylo z několika vhodných knihoven/frameworků pro práci se strojovým učením:

- 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.

[3] statmt.org/moses/?n=Moses.Baseline
[4] python.org

- 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 knihovne – Tensorflow, Theano nebo CNTK.

Zvolena byla knihovna Keras [6] pro svůj jednoduchý a více intuitivní přístup a také pro množství návodů, které pro tuto knihovnu vznikají. Jako backend je použita knihovna Tensorflow [1].

4.3.1 Balíček nmt

Překladový systém je naimplementován formou pythoního balíčku (knihovny) dostupného na githubu [5]. S pomocí vyro

Bucketing: A padding. Rozdělení sekvencí na skupiny podle délky, abych nepaddingoval zbytečně a tím neplýtval výkon. **[[porovnat čas jaký to běží a rozdíl v přesnosti překladu vůči bez bucketingu]]**

Generování encoded vět aby se vešly do paměti:

4.4 popis fungování systému a jednotlivých tříd

[[zkusit nějak použit vygenerovanou dokumentaci? nebo aspon do příloh]]

[5] https://github.com/jojkos/master-thesis/tree/master/code/nmtPackage

Kapitola 5

Experimenty a vyhodnocení

5.1 skóre BLEU

[[vysazet hezky vzorce]]

Naprogramoval jsem. Posbíral jsem data. Pustil jsem to. Výsledky jsou takové. Je to tak a tak rychlé.

Obrázek 5.1: One image. [[Napsat pořádný titulek]]

Kapitola 6

Závěr

- Autor se ohlíží za tím, co udělal: „V práci je. Hlavní úspěchy jsou. Důležitými výsledky jsou. Podařilo se.“
- Autor uvede nápady, které nestihl realizovat v podobě možností pokračování: „Ještě by šlo zkusit. Kdybych byl na začátku věděl, co vím teď, dělal bych.“
- Autor (ve vlastním zájmu) rekapituluje, jak bylo naplněno zadání práce.

Plány do budoucna

- použití bidirectional první vrstvy encoderu, pro lepší zachování contextu [21] na místo použití obrácených vstupů
- použití wordpieces [21] místo celých slov pro lepší handling rare words (mozna)
- přidat attention [2]
- přidat beam search [17], sehnat původní článek co přinesl beam search
- vyzkouset ruzny poctvy vrstev LSTM a velikosti vrstev
- pridat early stopping

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] Bahdanau, D.; Cho, K.; Bengio, Y.: Neural Machine Translation by Jointly Learning to Align and Translate. *CoRR*, ročník abs/1409.0473, 2014, 1409.0473.
URL http://arxiv.org/abs/1409.0473

[3] Bengio, Y.: Neural net language models. *Scholarpedia*, ročník 3, č. 1, 2008: str. 3881, doi:10.4249/scholarpedia.3881, revision #91566.

[4] 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

[5] 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

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

[7] 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

[8] 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

[9] 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

[10] 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

[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] 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

[20] 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

[21] 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