
**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** **Bc. 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 postavený na rekurentních neuronových sítích. K tomu je použita architektura enkodér-dekodér umožňující překlad po celých větách. Výsledkem je knihovna *nmt*, určená k provádění experimentů s různými parametry modelu. Jejich výsledky jsou porovnány vůči systému postavenému na nástroji pro statistický překlad Moses.



# Abstract

The goal of this thesis is to describe and build a system for neural machine translation. System is built with recurrent neural networks – encoder-decoder architecture in particular. The result is a *nmt* library used to conduct experiments with different model parameters. Results of the experiments are compared with system built with the 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
1. května 2018

## Poděkování

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

# Obsah

# Kapitola 1

# Úvod

Schopnost dorozumět se s ostatními lidmi na planetě je nesmírně důležitá, české přísloví dokonce praví „Kolik řečí znáš, tolikrát jsi člověkem“. Proto učenci od pradávných let vytvářeli slovníky a vědci od vzniku výpočetní techniky zkoumají jak vytvořit kvalitní 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 intenzivním rozvojem strojového učení a umělé inteligence, který nastal v posledních letech, se k tomuto ideálu blížíme mílovými kroky. Automatické rozpoznání mluvené řeči je již ve skvělé kvalitě dostupné v běžných spotřebitelských zařízeních a automatický překlad se taky značně vylepšuje.

Obsahem této práce je popis a realizace takovéhoto automatického překladového systému. Systém by měl být schopný naučit se za pomocí velkého množství zarovnaných vět v různých jazycích překládat věty mezi těmito jazyky. K tomu je použita architektura rekurentních neuronových sítí enkodér-dekodér, která vykazuje v posledních letech dobré výsledky.

V následující kapitole 2 je popis překladového systému a jsou zde rozebrány důležité pojmy a teorie ze kterých je je systém vystavěn. Větší část této kapitoly vznikla v rámci semestrálního projektu. Kapitola 3 popisuje implementovaný systém a kapitola 4 podává výsledky experimentů s vytvořeným systémem a porovnává je s výsledky podanými nástrojem pro statistický překlad Moses.

Vytvořený balíček je publikovaný na githubu `github.com/jojkos/neural-machine-translation`.

# Kapitola 2

# Strojový překlad

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

## 2.1 Popis cílového systému

Cílem této práce práce je vytvořit systém pro strojový překlad textu pomocí umělých neuronových sítí. Pro snadnou představu, je to podobné tomu co dělá Google Translator[1] – blíže popsáno v článku [2]. Vezme se věta v původním jazyce a vytvoří se z ní co nejvěrnější překlad v jazyce cílovém a to za pomocí natrénované rekurentní neuronové sítě. V této sekci je vysvětleno, jak by takový systém mohl vypadat jaké komponenty potřebuje k tomu, aby fungoval.

**Dataset:** Aby bylo možné nacvičit neuronovou síť pro překlad, je nejprve zapotřebí mít dataset. 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 [14]).

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

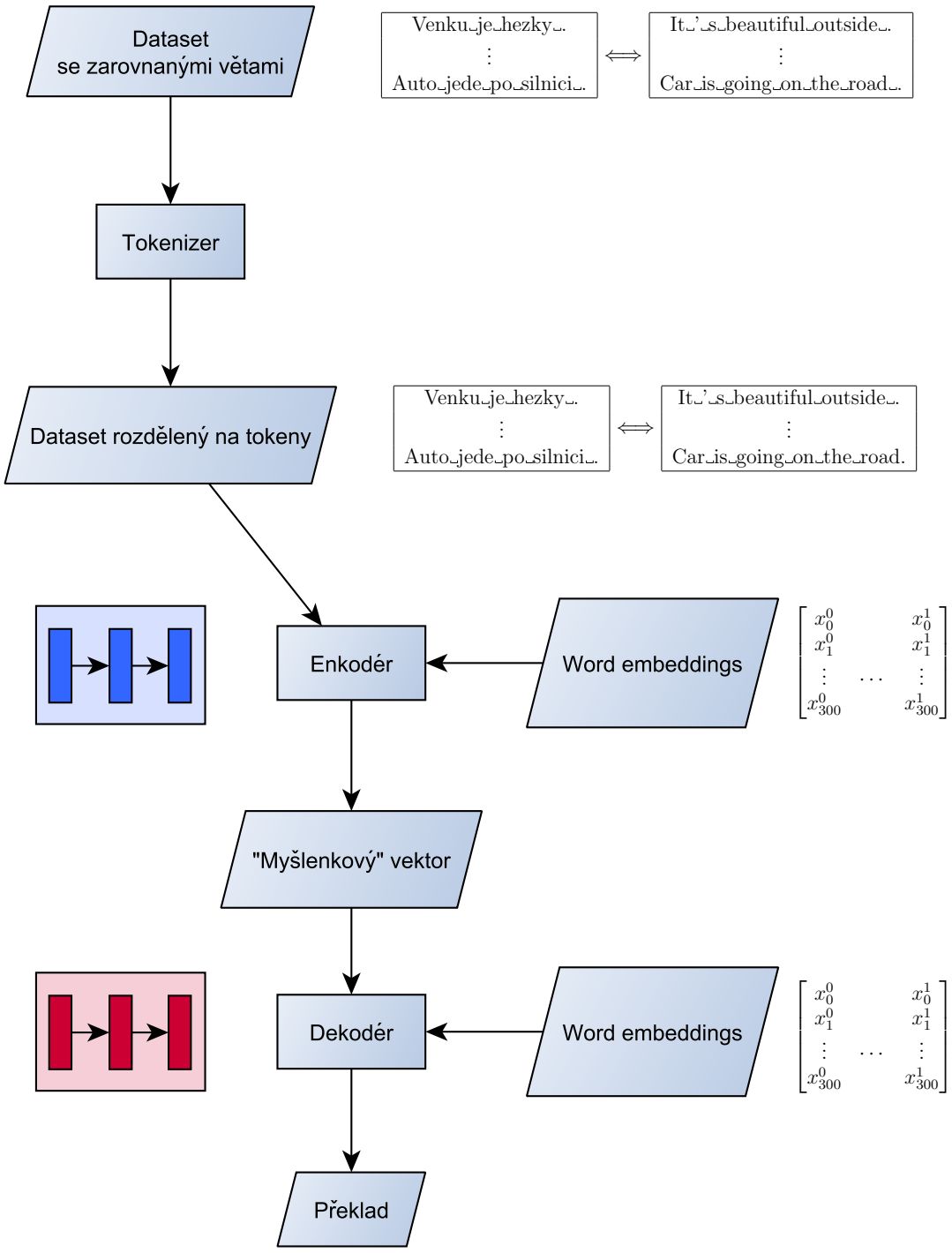
**Slovník:** Slovník se vytvoří jako seznam $n$ nejčastějších slov v datasetu ve vstupním a cílovém jazyce. Čím je slovník menší, tím jsou menší požadavky na vypočetní výkon. Na druhou stranu je pak potřeba vyřešit při trénování a překladu problém se slovy nevyskytujícími se ve slovníku (popsáno v sekci 2.2.4).

**Word Embeddings:** Obecně je možné vytvářet jazykové modely, které generují text po písmenech, částech slov nebo po slovech [19]. 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 2.2.3.

[1] translate.google.cz

**Model:** V současné době nejúspěšnější architekturou používanou pro překlad je enkodér-dekodér umožňující překlad po celých větách (sequence to sequence, dále seq2seq [28]). To je rozdíl oproti starším statistickým metodám překladu, kde se překládalo po slovech či frázích.

Nejprve enkodér vezme word embedding na vstupu a pomocí rekurentní neuronové sítě (sekce 2.3) převede větu na vstupu do velkého vektoru reprezentujícího její význam (tzv. myšlenkový vektor – intuice je taková, že když člověk překládá větu, také nejprve pochopí její význam a až poté ji 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 2.2), který je 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.

## 2.2 Jazykové modely

Zatímco u programovacích jazyků existuje 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 precizně 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 tato neurčitá pravidla převést do formálních pravidel, tak, aby jim rozuměl počítač. Řešením pro tento problém mohou být jazykové modely, které nevznikají nadefinováním formálních pravidel, ale natrénováním modelu z příkladů. Sekce vychází z práce [17] a článku [21].

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 2.1.

$$p(w) = \prod_{i=1}^{m} p(w_i|w_{<i}) \tag{2.1}$$

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

### 2.2.1 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 pravděpodobnost věty P(E) (pro nějaký daný kontext nad kterým se model natrénoval).

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 neboli 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 2.2).

$$P(x_i \mid x_{i-(n-1)}, \ldots, x_{i-1}) \tag{2.2}$$

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

### 2.2.2 Log-lineární modely

Stejně jako v případě $n$-gram modelů (sekce 2.2.1), 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 2.2.3).

### 2.2.3 Neuronové sítě a word embeddings

Stejně jako předchozí modely i NLM (neural language model) je trénován tak aby předpovídal rozdělení pravděpodobností přes slova v cílovém slovníku na základě aktuálního kontextu (rovnice 2.1).

Předchozí modely při použití většího datasetu a tím pádem většího slovníku čelí „prokletí“ dimenzionality. Jednotlivá slova jsou běžně reprezentována jako **one-hot vektor** (obrázek 2.2). 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 neboli sparse vector).

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

Obrázek 2.2: 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 neboli dense vektory), ukázka v obrázku 2.3. Ke každému slovu ze slovníku se přiřadí takovýto vektor. Výhodou je, že může nést, na rozdíl 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 mají sobě blízké vektory, může model brát například jako synonyma.

$$V = [\text{farmář, jí, steak, kráva}] \quad wordEmbedding_{steak} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{300} \end{bmatrix}$$

Obrázek 2.3: Word embedding vektor pro slovo „steak“ ze slovníku $V$. Slovo je vyjádřeno vektorem o velikosti 300, který v sobě může uchovávát některé jeho sémantické vlastnosti.

Nejznámější ukázkou vlastností word embeddings je ukázka 2.4 z článku [20].

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

Obrázek 2.4: Ukázka vlastností word embeddings. $\approx$ 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 [18], glove [24] a fasttext [4].

Embbedings jsou vhodné pro **transfer learning**. To je způsob využití znalostí natrénovaných již třeba dříve 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 tak může být výrazně lepší.

### 2.2.4 Zpracování neznámých slov

Jazykový model typicky pracuje s pevně danou velikostí slovníku (počtem slov, co se mohou vyskytovat), což je problém, protože překlad je obecně problém s otevřenou slovní zásobou. Existuje-li dataset $\varepsilon_{train}$ obsahující texty na kterých se model bude učit a dataset $\varepsilon_{test}$, který bude sloužit k ověření výkonnosti 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 pro zlepšení výkonu, omezit celkový počet slov, se kterými se bude model pracovat. Tím pádem se však zvýší výskyt neznámých slov. Práce [21] 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, kdyby 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 rozdělení pravděpodobnosti i přes neznámá slova. Lze natrénovat jazykový model,který 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 $\varepsilon_{train}$ 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 uváděným řešením je místo slov jako nejmenších tokenů, se kterými se pracuje, použití takzvaných **subword units** neboli jednotek menších než slovo [26].

**Subword units:** Kvůli velkému množství slov a jejich tvarů se obvykle používá velikost slovníků kolem 30 000–50 000 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 z několika částí, předložkami a spojenými slovy (například „kladkostroj“). Ukázalo se, že v případě, že se slova rozdělí na takovéto podčásti, tak se:

1. Zmenší velikost slovníku - kdyby se místo slov pro trénování sítě používala jednotlivá písmena, velikost slovníku by byla 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, které 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 [26] 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, který by zapříčinil náročnější trénink modelu. Výrazně delší sekvence by znamenala větší vzdálenost, přes kterou by se musel model naučit přenést informace.

## 2.3 Rekurentní neuronové sítě

V této kapitole je popsán základní koncept rekurentních neuronových sítí (RNN[2]), jejich srovnání s běžnými neuronovými sítěmi a dále pak popis variant rekurentních sítí – LSTM (sekce 2.3.3) a GRU (sekce 2.3.4). Sekce vychází z práce [17], práce [21] a článku [22].

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

[2] z anglického recurrent neural network

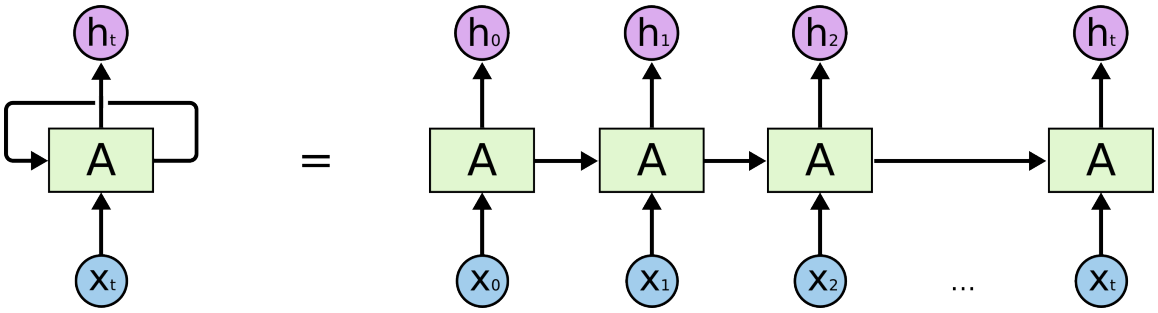
je nejvíce zajímavé pro tuto práci, pro tvorbu jazykových modelů, generátorů textu a tím pádem i pro překlad.

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 2.3).

$$y = f(x, W) \tag{2.3}$$

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 2.5). 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 2.4). Přičemž počáteční stav $h_0$ je obvykle nastaven na nulu.

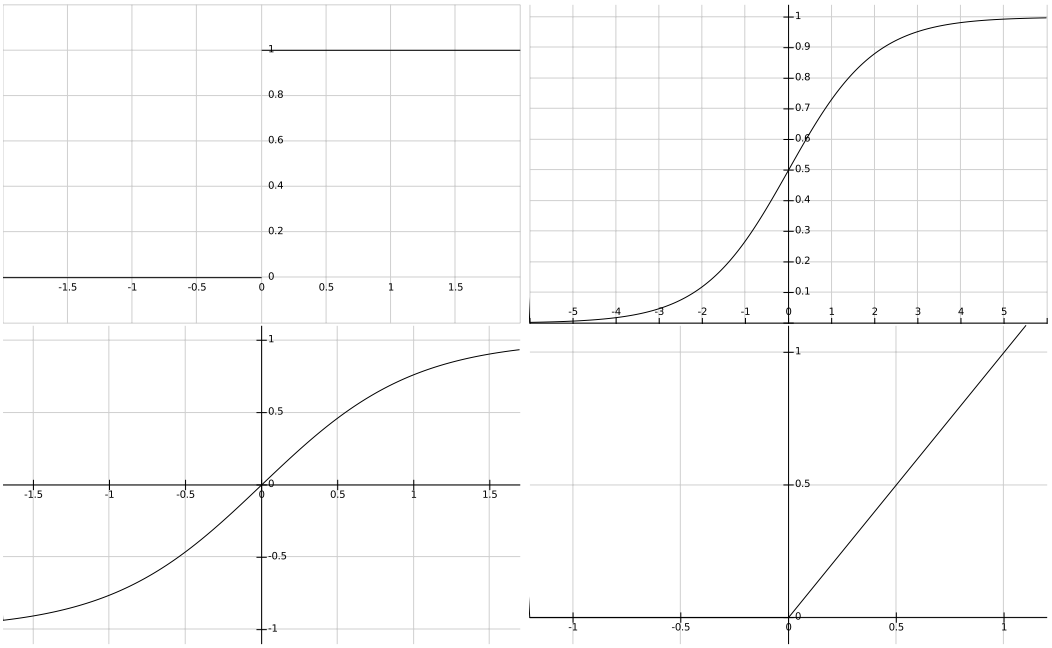



Obrázek 2.5: Znázornění RNN – $x_t$ je vstup, $A$ zastupuje vnitřní chování RNN a $h_t$ je skrytý stav. Rozdílem oproti běžné dopředné neuronové síti je zpětná smyčka a skrytý stav. Pravá část obrázku ukazuje pro lepší představu místo zpětné smyčky rozbalenou strukturu přes jednotlivé časy $t$. Intuitivně se pak dá odhadnout, že RNN umí dobře pracovat s podobnými strukturami jako jsou sekvence a seznamy. Obrázek převzat z [22].

Rovnice RNN pro výpočet vnitřního stavu:

$$h_t = \begin{cases} f(W_{xh}x_t + W_{hh}h_{t-1} + b_h) & \text{pokud t} \geq 1, \\ 0 & \text{jinak.} \end{cases} \tag{2.4}$$

$W_{xh}$ znázorňuje váhy pro aktuální vstup, $W_{hh}$ jsou váhy pro skrytý stav z minulého kroku a $b_h$ je bias. Funkce $f$ z rovnice 2.4 je nelineární funkcí a nejčastěji se používá jedna z funkcí *step*, *sigmoida*, *tanh* nebo *relu* (obrázek 2.6).



Obrázek 2.6: Nelineární funkce *step*, *sigmoida*, *tanh* a *relu*.

Rovnice pro RNN jazykový model jsou následující:

$$m_t = M_{e_{t-1}} \tag{2.5}$$
$$h_t = RNN(m_t, h_{t-1}) \tag{2.6}$$
$$s_t = W_{hs} h_t + b_s \tag{2.7}$$
$$p_t = softmax(s_t) \tag{2.8}$$

kde 2.5 je aktuální kontext, 2.6 je zjednodušený přepis rovnice RNN (2.4) a rovnice 2.8 je funkce softmax, která je podrobněji popsána v 2.9. Softmax vezme všechny hodnoty skóre pro jednotlivá slova a transformuje je do pravděpodobnostního rozdělení $p_t$. Díky tomu pak lze již snadno určit, které slovo bude vygenerováno s největší pravděpodobností.

$$p_t(y) = \frac{e^{p_t(y)}}{\sum_{k=1}^{K} e^{p_{t_k}}} \tag{2.9}$$

Protože vektor $m$ z rovnice 2.5 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 2.2.2 byl jako problém uveden příklad „Farmář jí steak“ a „Kráva jí steak“, 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 jí farmář"a správně se aktivovat a pak nabízet slova jako „steak“ 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 2.3.2.

### 2.3.1 Trénování

Cílem trénování sítě je nalézt takové parametry $\theta$ (kombinace vah $W$ a biasu $b$), aby se minimalizovala hodnota takzvané *loss funkce*. Loss funkce vyjadřuje jak moc špatně výstupy sítě odpovídají datům, na kterých se síť trénuje. Průchod sítí a následné vypočítání loss funkce se nazývá *dopřednou propagací*.

K optimalizaci parametrů pro nalezení minima loss funkce se používá *zpětná propagace*. Vypočte se přírůstek pro každý parametr tak, aby síť s novými váhami o něco lépe pracovala a loss funkce se snížila. Existuje více různých metod optimalizace pro tento výpočet podrobně popsaných v práci [25].

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 v jedné dávce.

### 2.3.2 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 [3]).

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 through 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 se 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 2.3.3).

### 2.3.3 LSTM

Long short term memory, dále LSTM, (původní článek [12] a varianta LSTM s forget gate z článku [9], která je zde použitá), neboli dlouhá krátkodobá paměť, je varianta RNN navržená jako řešení problémů mizejícího/explodujícího gradientu a vzdálených závislostí.

Stejně jako základní RNN (sekce 2.3) se dá LSTM představit jako opakující se modul v řetězové struktuře (viz obrázek 2.5). Rozdíl je ve vnitřku modulu $A$. Zatímco RNN používá pouze jednu nelineární funkci (rovnice 2.4), struktura LSTM je složitější (obrázek 2.7 a následující rovnice).

$$u_t = tanh(W_{xu}x_t + W_{hu}h_{t-1} + b_u) \tag{2.10}$$
$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \tag{2.11}$$
$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \tag{2.12}$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \tag{2.13}$$
$$c_t = i_t \odot u_t + f_t \odot c_{t-1} \tag{2.14}$$
$$h_t = o_t \odot tanh(c_t) \tag{2.15}$$

RNN má pouze skrytý stav $h$. LSTM má navíc ještě paměťovou buňku $c$ (rovnice 2.14). Protože gradient této buňky je roven jedné, netrpí tak LSTM problémy ze sekce 2.3.2 a mohou tak v ní být zachyceny i vzdálené závislosti.

Rovnice 2.10 je update funkcí a je ekvivalentem rovnice 2.4 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 zmíněná zapomínací brána neboli forget gate (rovnice 2.11). 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.

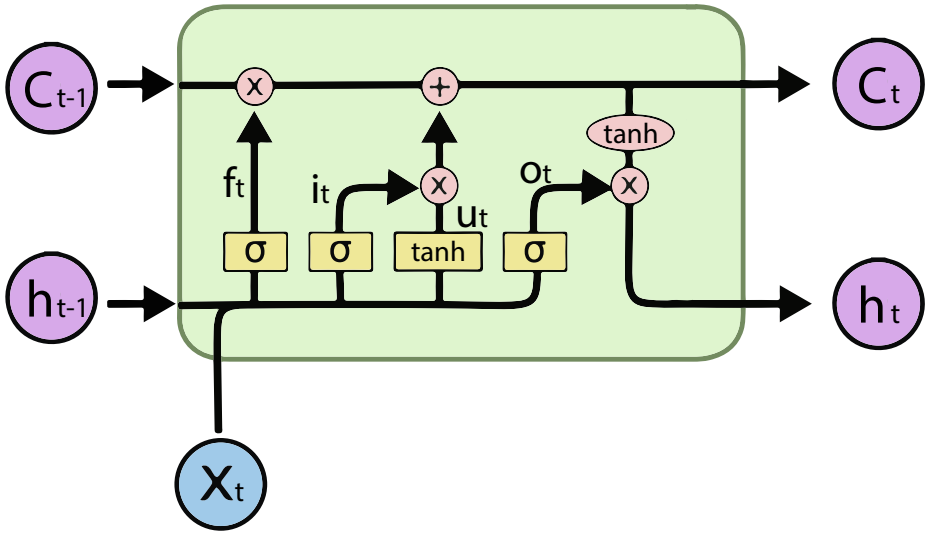
Dalším krokem je vyhodnocení toho, co se má přidat do paměti. Nejdříve vstupní brána neboli input gate (rovnice 2.12) rozhodne, které hodnoty se změní nebo přidají. V návaznosti na minulý příklad by síť mohla chtít uložit aktuální rod nalezeného podstatného jména. Update funkce (rovnice 2.10) vyhodnotí, jaké hodnoty se mají přidat.

Následuje aktualizace paměti $c_t$ (rovnice 2.14). 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 2.13) a v rovnici 2.15 se získá výsledek.

Pravděpodobnosti jazykového modelu se získají rovnicí:

$$p_t = softmax(W_{hs}h_t + b_s) \tag{2.16}$$



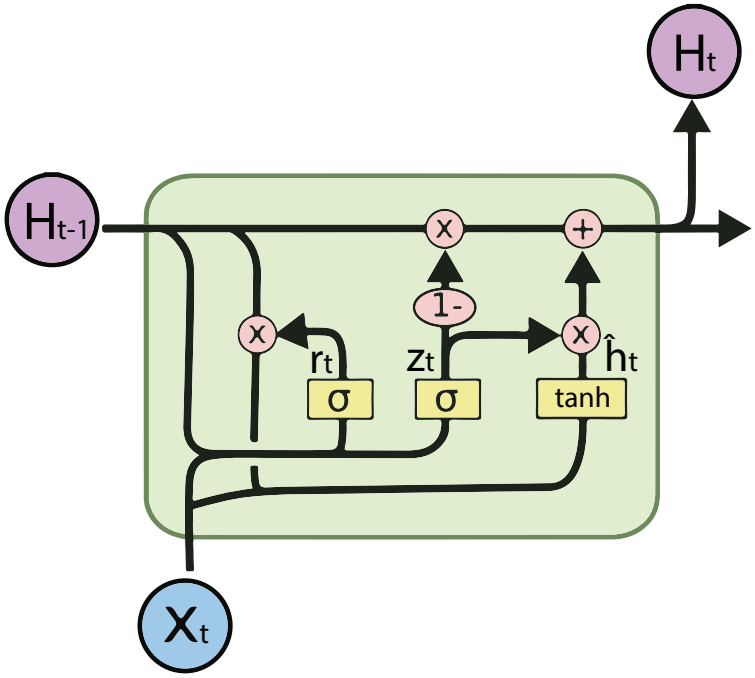

Obrázek 2.7: Jeden časový úsek LSTM kde $h$ je skrytý stav, $c$ je paměťová buňka a $x$ je vstup. Vnitřní struktura koresponduje s rovnicemi 2.10 až 2.15. Obrázek převzat z [22], upraven.

### 2.3.4 GRU

LSTM z minulé sekce je dobrým řešením pro problémy ze sekce 2.3.2. Má však dosti komplikovanou strukturu a tím pádem jsou i vyšší nároky na výpočetní výkon. To podnítilo vznik další varianty RNN — GRU, neboli gated recurrent unit (článek [6]). GRU je o něco jednodušší, a proto je to vhodnější varianta pro úsporu výkonu.

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \tag{2.17}$$
$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \tag{2.18}$$
$$\tilde{h}_t = tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h) \tag{2.19}$$
$$h_t = (1 - z_t)h_{t-1} + z_t\tilde{h}_t) \tag{2.20}$$

GRU má pouze dvě brány a skrytý stav $h$. Nový stav se počítá v rovnici 2.20 interpolací mezi minulým stavem $h_{t-1}$ a kandidátem na nový stav $\tilde{h}_t$ upravený hodnotou **update** brány (rovnice 2.18). Kandidát se získá v rovnici 2.19, která je podobná update funkci z RNN (rovnice 2.4), ale je upravena hodnotou **resetovací** brány (rovnice 2.17). Struktura je zobrazená na obrázku 2.8.

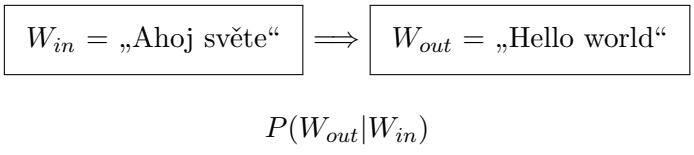



Obrázek 2.8: Jeden časový úsek GRU, kde $h$ je skrytý stav a $x$ je vstup. Vnitřní struktura koresponduje s rovnicemi 2.17 až 2.20. Obrázek převzat z [22], upraven.

## 2.4 Seq2seq model s architekturou enkodér-dekodér

V předchozích sekcích se práce zabývá rekurentními neuronovými sítěmi a jazykovými modely na nich postavenými. V této sekci bude popsáno, jak tyto sítě vzít a poskládat je vhodným způsobem pro překlad vět. Sekce vychází z práce [21].

**Seq2seq** (článek [28]) neboli sequence to sequence je způsob překladu po celých větách. Jde o modelování pravděpodobnosti $P(E|F)$, tedy pravděpodobnosti výstupu $E$ na základě vstupu $F$ (obrázek 2.9).




Obrázek 2.9: 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. Tento výsledný skrytý stav, takzvaný myšlenkový vektor, vyjadřuje význam dané věty. Dekodér inicializovaný tímto stavem generuje (dekóduje) z myšlenkového vektoru přeloženou větu. Díky tomu, že dekodér generuje na základě stavu, kterým byl inicializován, není nijak vázaný na délku původní věty a přeložená věta tak může být jinak dlouhá.

$$m_t^{(f)} = M_{f_t}^{(f)} \tag{2.21}$$

$$h_t^f = \begin{cases} RNN^{(f)}(m_t^{(f)}, h_{t-1}^{(f)}) & \text{pokud t} \geq 1, \\ 0 & \text{jinak.} \end{cases} \tag{2.22}$$

$$m_t^{(e)} = M_{e_{t-1}}^{(e)} \tag{2.23}$$

$$h_t^e = \begin{cases} RNN^{(e)}(m_t^{(e)}, h_{t-1}^{(e)}) & \text{pokud t} \geq 1, \\ h_{|F|}^f & \text{jinak.} \end{cases} \tag{2.24}$$

$$p_t^{(e)} = \text{softmax}(W_{hs} h_t^{(e)} + b_s) \tag{2.25}$$

Pro každé slovo v čase $t$ ze vstupní sequence $F$ se vyhledá jeho embedding (rovnice 2.21). Následně se v rovnici 2.22 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 2.23). 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 **startovací** symbol <s>. Rovnice pro výpočet skrytého stavu dekodéru (2.24) je prakticky stejná jak u enkodéru. 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 rozdělení se pak jako u všech jazykových modelů spočítá pomocí funkce *softmax* (rovnice 2.25).

### 2.4.1 Průběh trénování a generování

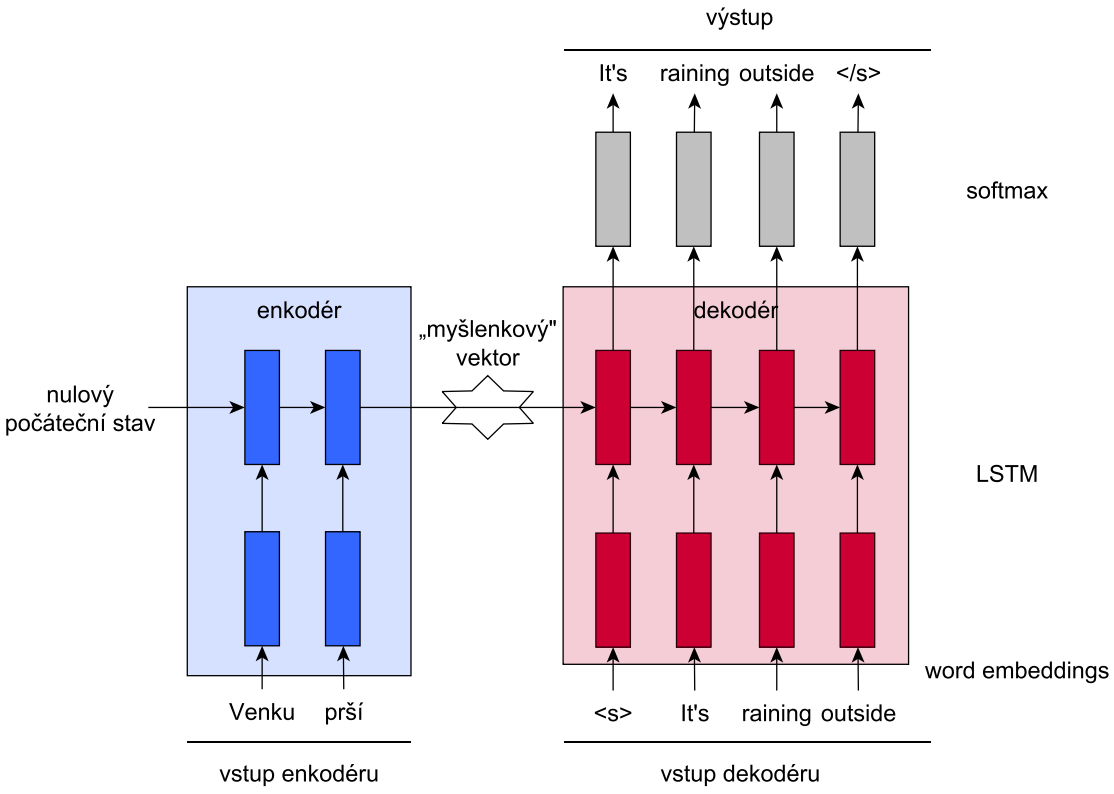
Cílem jazykového modelu je předpovídat následující slovo ve větě. Při trénování se nejprve do enkodéru pošle výchozí věta, aby se získal inicializační stav pro dekodér. Do dekodéru, inicializovaného získaným stavem, se nejprve pošle **startovací** symbol. Startovací symbol dekodéru říká, že má začít překládat. Při trénování se mu pak následně posílají korektní další slova referenčního překladu, aby se zrychlilo učení. Tato metoda se nazývá „teacher forcing“ ([10]). Proces je znázorněný na obrázku 2.10. Po naučení modelu se ve fázi generování do dekodéru posílají slova, která již sám vygeneroval (obrázek 2.11). Dekodér generuje

tak dlouho, dokud nenarazí na **koncový** symbol, kterým je v době trénování zakončená každá věta. Ve skutečnosti však výstupem dekodéru není přímo slovo, ale rozdělení pravděpodobnosti přes všechna slova cílového slovníku získaného funkcí *softmax* v rovnici 2.25. Je několik možností jak z tohoto rozdělení vybrat konkrétní slovo:

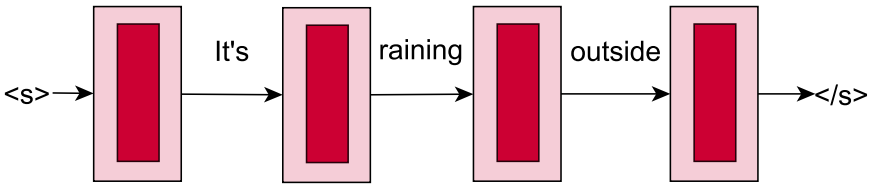
**Náhodný výběr:** Z rozdělení pravděpodobnosti $P(E|F)$ se slovo vybere náhodně.

**Chamtivý výběr:** Chamtivý (greedy) výběr spočívá ve výběru 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 neboli 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 2.10: Architektura enkodér-dekodér. Enkodér obdrží větu na vstupu a vytvoří inicializační stav pro dekodér („myšlenkový vektor“). Tímto vektorem je inicializován počáteční stav dekodéru. Ten v době trénování dostává na vstupu správně přeloženou větu (teacher forcing) a v době generování na vstup dostává slova, která sám vygeneroval. Konečný výstup se získá pomocí vrstvy softmax.




Obrázek 2.11: Ukázka práce dekodéru v době predikce překladu (inference). Na vstup jako první přichází startovací symbol <s>. Následně v každém dalším kroku na vstup dekodéru dostane svůj vlastní výstup z kroku minulého. Takto generuje tak dlouho, dokud nenarazí na koncový symbol </s>.

### 2.4.2 Metody optimalizace

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

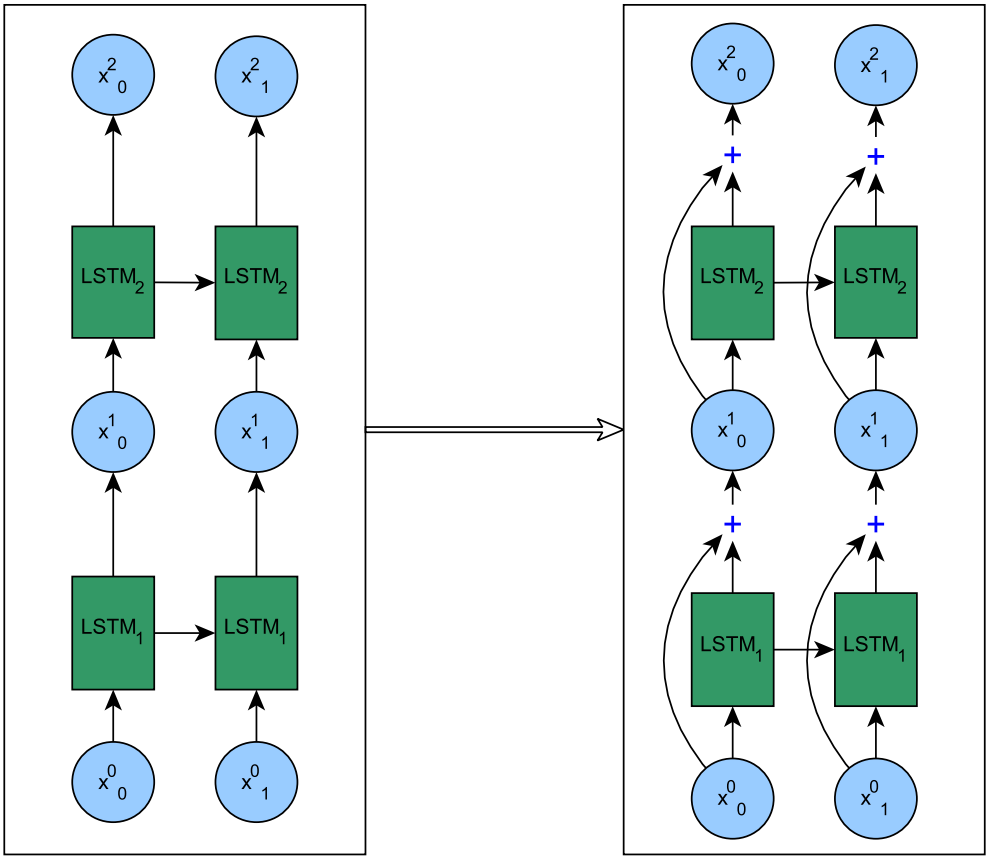
**Převrácení vstupu:** Článek [28] udává, že výrazným způsobem pomůže, když se slova ve vstupní sekvenci převrátí a do enkodéru se věta předává pozpátku. Pravděpodobně je to díky tomu, že závislosti, které by běžně byly vzdálené – typicky poslední slovo ve vstupní větě a jeho přeložená varianta v přeložené 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 [2] udává, že více vrstev může do určité hloubky pomoci. V práci jich vědci použili 8 jak pro enkodér tak dekodér. Při použití většího množství již má model problém se úspěšně učit.

**Reziduální propojení:** V práci [2] doporučují při použití více vrstev LSTM použít takzvané reziduální propojení nebo zapojení mezi vrstvami. Podle jejich experimentů při použití většího počtu vrstev a běžném zapojení začne být učení pomalé a složité. 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 do každé následující vrstvy výstup z vrstvy minulé sečtený dohromady se vstupem minulé vrstvy. Tedy do $LSTM_1$ vchází $x_1 + x_0$ (obrázek 2.12).

**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 ([27]), 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.

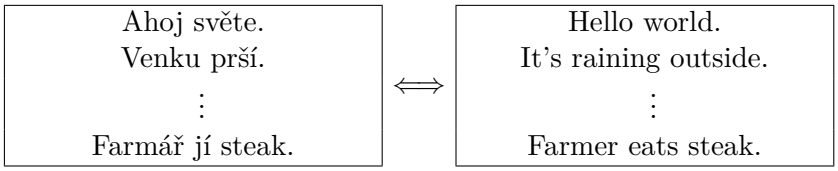



Obrázek 2.12: Rozdíl mezi normálním a reziduálním propojení pro zlepšení učení více vrstvých LSTM. Levý obrázek znázorňuje běžné zapojení, zatímco na pravém je ukázka reziduálního propojení. Do každé (kromě první) vrstvy $LSTM_i$ vstupuje součet výstupu vrstvy minulé $x_i$ sečtený se vstupem minulé vrstvy $x_{i-1}$.

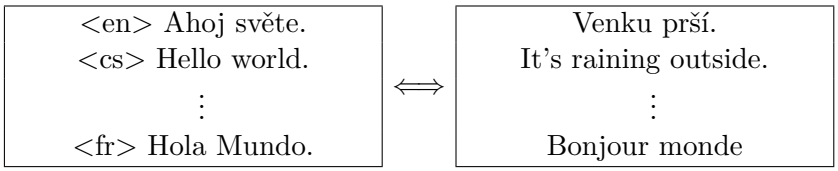
### 2.4.3 Překlad mezi více jazyky s jedním modelem

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 [13] ukazuje, že se stejným enkodér-dekodér modelem lze bez jeho úprav překládat mezi několika jazyky. Jediným rozdílem je přidání speciálního tokenu před zdrojové věty. Tento token udává do jakého jazyka se má daná věta překládat (znázorněno na obrázku 2.13). V práci se používá slovník wordpieces (2.2.4) 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 2.13: Obrázek ukazuje rozdíl mezi páry vět použitými při běžném překladu mezi dvěmi jazyky (2.13a) a při použití stejného modelu pro překlad mezi více jazyky (2.13b). 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.

## 2.5 Automatické hodnocení vlastností strojových překladových systémů

Tato sekce popisuje různé metody automatického hodnocení překladů vytvořených strojovým překladovým systémem. Všechny fungují na základě porovnání překladů vyrobených testovaným systémem s referenčním překladem daného textu.

Je také možné hodnotit překlad ručně, za pomocí lidí, ale pro tuto práci je takovéto hodnocení příliš složité a není tedy vhodné.

### 2.5.1 BLEU

*BLEU* [23] neboli bilingual evaluation understudy, je jedním z nejpopulárnějších způsobů hodnocení kvality systému mezi výzkumníky. Snaží se hodnotit takovým způsobem, aby

čím je větší skóre, tím se překlad víc blíží k něčemu co by přeložil profesionální lidský překladatel.

BLEU pracuje s přesností n-gramů. Skóre 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%.

### 2.5.2 NIST

*NIST* [7] je metoda založená na metodě BLEU, vyvinutá Národním institutem standardů a technologie. Zatímco BLEU jednoduše pracuje s přeností n-gramů a nedává jim různou váhu, NIST váhu upravuje podle výskytu daného n-gramu. Čím vzácnější n-gram je, tím větší váhu v hodnocení dostane.

### 2.5.3 METEOR

*METEOR* [16] je metoda založená na harmonickém průměru přesností unigramů. Na rozdíl od BLEU metody, která se aplikuje na úrovni celého datasetu, METEOR podává dobrou korelaci s lidským hodnocením i na úrovni vět.

### 2.5.4 LEPOR

*LEPOR* [11] je nejnovější z hodnotících metod a je kombinací metod předchozích. Ostatní metriky mají tendenci dobře fungovat jen pro některé jazykové páry, LEPOR poskytuje škálu nastavitelných parametrů, kterými se má jazykový bias omezit.

# Kapitola 3

# Implementace

Tato kapitola popisuje všechny autorem vytvořené a použité části. Sekce 3.1 je o výběru a předzpracování datasetů. Následující sekce 3.2 se zabývá vytvořením referenčního systému, vůči kterému se porovnávají výsledky v kapitole 4. Poslední sekce této kapitoly (3.3) popisuje vytvořený překladový systém.

## 3.1 Datasety

Jako dataset (nebo korpus) se v této práci považují dva soubory. Na každém řádku souboru je jedna věta a ta svým významem odpovídá větě na stejném řádku v druhém souboru. Dataset nese nějaký název (název souboru stejný pro oba jazykové soubory) a jako koncovku používá dvoupísmennou zkratku jazyka. Pro lepší představu je přiložen obrázek 3.1.

| exampleDataset.cs | | exampleDataset.en |
|---|---|---|
| Ahoj světe. | | Hello world. |
| Venku prší. | ⟺ | It's raining outside. |
| ⋮ | | ⋮ |
| Farmář jí steak. | | Farmer eats steak. |

Obrázek 3.1: 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.

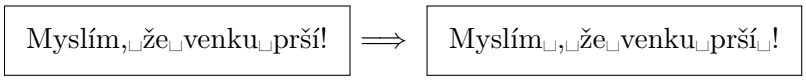
### 3.1.1 Předzpracování

Před použitím na trénování a vyhodnocování překládacího systému je potřeba datasety vhodným způsobem předpřipravit. Tato sekce popisuje aplikované metody. Všechny použité skripty jsou dostupné na githubu programu Moses[1]. Cílem je snížit velikost výsledných slovníků a zbavit se nevhodných vět.

**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

[1] https://github.com/moses-smt/mosesdecoder

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. Ukázka tokenizace je na obrázku 3.2.




Obrázek 3.2: 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 písmena 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 různých 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 2.2.4, může být pro trénování výhodné nepoužívat jako nejmenší jednotku slova, ale jejich části. Pro aplikování BPE je použita knihovna *subword-nmt*[2]. Podle doporučení se BPE vytváří dohromady nad datasety zdrojového i cílového jazyka. Zvolený počet *merge* operací je 15000.

## 3.2 Referenční systém v Moses

Moses [15] je nástroj na vytváření statistických strojových překladových systémů. Vzniklý model bude sloužit jako referenční, vůči kterému se porovnají výsledky implementovaného překladového systému (sekce 3.3). Kromě toho se také používají některé skripty z tohoto nástroje pro přípravu datasetů (sekce 3.1.1) a získání skóre BLEU. Konkrétní postup jeho přípravy je dostupný na stránkách Moses [3], byla použita výchozí nastavení.

## 3.3 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 (tenzory). Tensorflow se stala velice populární v oblasti vývoje neuronových sítí.

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

- Theano – knihovna pro efektivní práci s mnohorozměrnými poli. Využívá pole z hojně používané pythonovské knihovny Numpy. Nedávno se knihovna dostala na verzi 1.0 a zároveň s tím se ukončil její vývoj.
- CNTK – Cognitive Toolkit 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 knihovna pro Python 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.

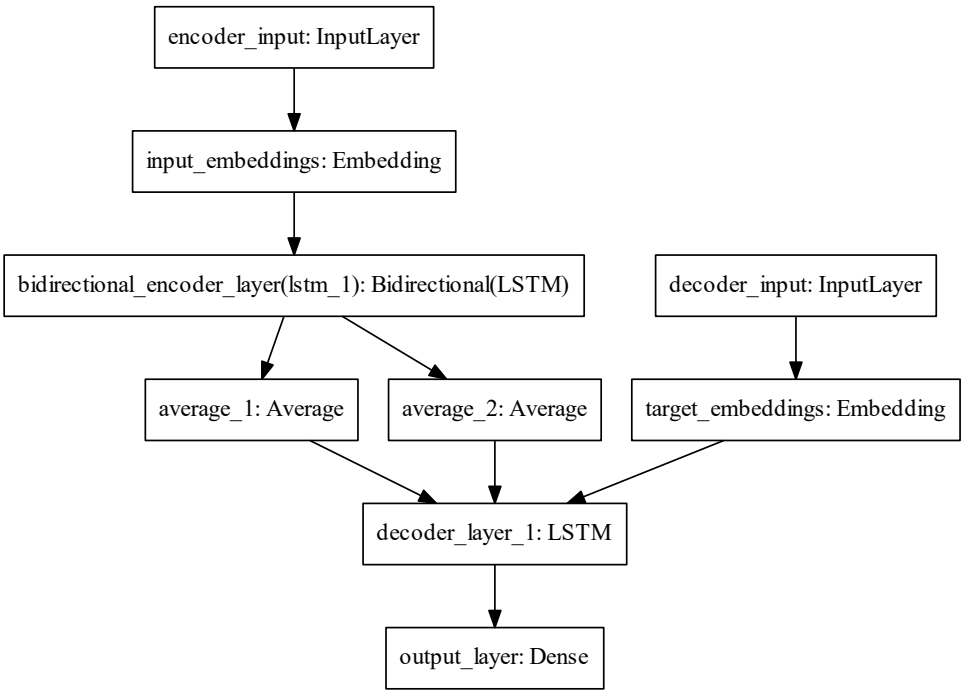
### 3.3.1 Balíček nmt

Překladový systém je naimplementován formou balíčku pro Python (knihovny), který je zveřejněn na githubu[5]. Pro jeho implementaci byla zvolena knihovna Keras [5] 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].

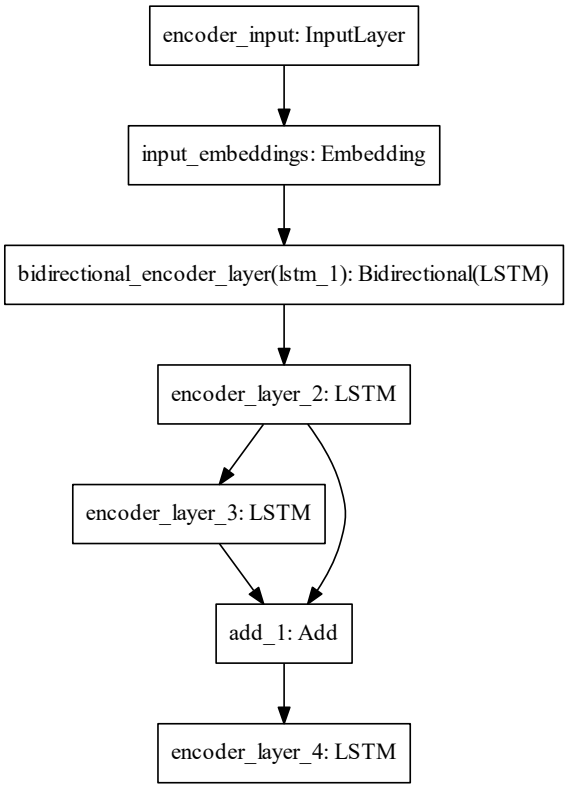
Slovníky výchozího a cílového jazyku se omezují na zvolenou maximální velikost a neznámá slova jsou nahrazeny symbolem <unk> (viz sekce 2.2.4). Knihovna umožňuje použití předtrénovaných word embeddings (2.2.3) ve formátu *fastText*. Jsou implementovány a použity optimalizace popsané v sekci 2.4.2 – používá se obousměrný enkodér, je možné vytvořit model s různou hloubkou sítí a při trénování se používá dropout a teacher forcing (2.4.1). Vrstvy enkodéru a dekodéru jsou tvořeny jednotkami LSTM (2.3.3) s použitím reziduálních propojení. Pro co nejpřesnější generování předpovídaných vět je použito paprskové prohledávání (2.4.1).

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 3.3) (do modelu vstupují věty ve výchozím jazyce, věty v cílovém jazyce pro použití teacher forcing), model enkodéru použitý při překladu pro získání inicializačních hodnot pro dekodér (obrázek 3.4) a model dekodéru použitý při překladu pro generování vět (obrázek 3.5).

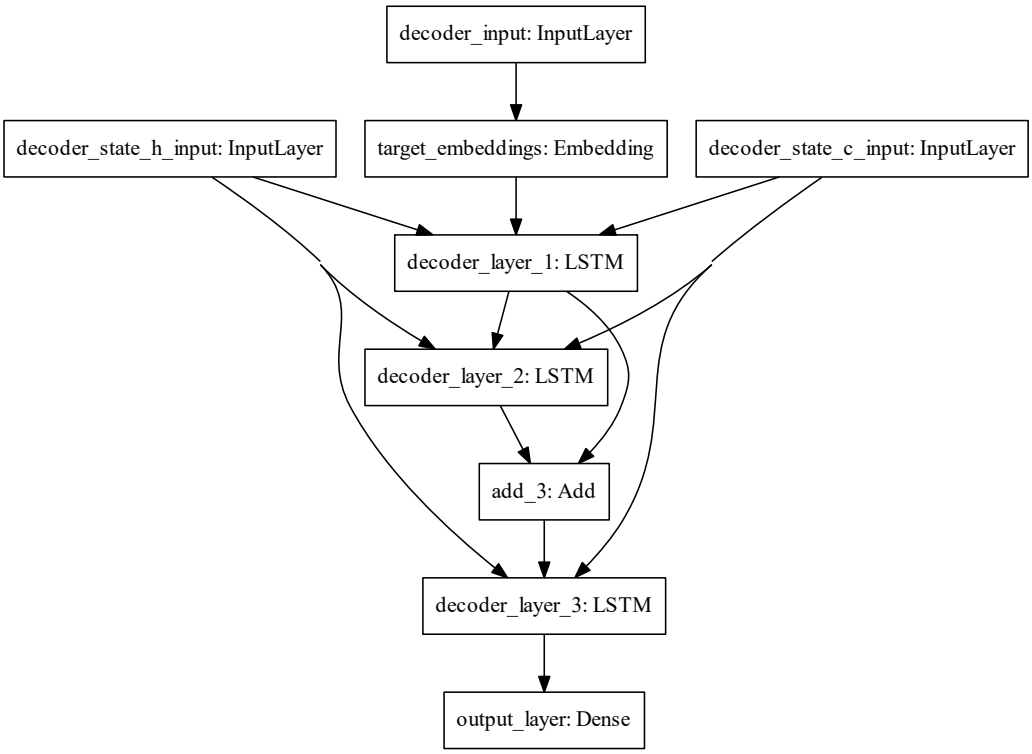
[5] https://github.com/jojkos/neural-machine-translation




Obrázek 3.3: Vrstvy a jejich propojení v enkodér-dekodér modelu použitém při trénovaní za použití jedné vrstvy LSTM pro enkodér a jedné vrstvy LSTM pro dekodér. Do *encoder_input* přicházejí sekvence ve výchozím jazyce. V *input_embeddings* se převedou do podoby embeddings vektoru. Následuje obousměrná vrstva enkodéru. Do *decoder_input* přichází sekvence začínající startovacím tokenem <s> následovaná korektním překladem v cílovém jazyce, protože se používá teacher forcing. Do *decoder_input* přicházejí sekvence v cílovém jazyce. V *target_embeddings* se převedou do podoby embeddings vektoru. Následuje vrstva dekodéru, který je inicializovaný výsledným stavem enkodéru. Zprůměrovaným, protože enkodér je obousměrný, takže má dvojnásobně veliký vnitřní stav. Poslední vrstva *output_layer* s aktivační funkcí *softmax* vrací pravděpodobnosti pro všechna slova z cílového slovníku.




Obrázek 3.4: Vrstvy a jejich propojení v modelu enkodéru použitém při generování překladů při použití tří vrstev LSTM. První vrstva je obousměrná, další vrstvy jsou již jen dopředné. Protože velikost embeddings se nemusí shodovat s počtem použitých jednotek v LSTM a první vrstva je obousměrná, reziduální propojení se používá až od třetí vrstvy a dál.




Obrázek 3.5: Vrstvy a jejich propojení v modelu dekodéru použitém při generování překladů při použití 3 vrstev LSTM. Každá vrstva dekodéru je inicializována koncovým stavem enkodéru. Protože velikost embeddings se nemusí shodovat s počtem použitých jednotek v LSTM, reziduální propojení se používá až od druhé vrstvy a dál.

### Stručný popis obsahu `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

Všechny parametry třídy *Translator* jsou popsány v příloze B.

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

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

**třída Candidate:** Je pomocná třída použitá při výpočtu paprskového prohledávání 2.4.1. 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 (3.1.1). Knihovna je přiložena 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*.

### 3.3.2 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 nebo zarovnávacího tokenu na délku nejdelší věty (obrázek 3.6). 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 .“ | $\Longrightarrow$ | [1, 2, 3, 4, 5] |
| „Ahoj světe .“ | | [6, 7, 5, 0, 0] |

Obrázek 3.6: 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 se používá k 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 (popsáno v sekci 3.3.3) berou vždy z jedné skupiny tak, aby všechny sekevence v jedné dávce byly stejně dlouhé.

### 3.3.3 Generování dávek

Protože se data při trénování upravují datasety do podoby velkých matic, není možné, kvůli paměťovým omezením, vyrobit jednu matici pro všechna data zaráz. Proto se dávky (batche) v průběhu trénování vytváří postupně, pomocí generátorové funkce. Ta před každou epochou (jedna epocha znamená, že modelem prošla všechna vstupní data), zamíchá vstupní data. Správné míchání vstupních dat je důležité pro dobrou konvergenci modelu. Dále postupně generuje matice s jednotlivými dávkami. V případě, že je použité dělení do

skupin podle velikosti, probíhá míchání na úrovni 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.

# Kapitola 4

# Experimenty a výsledky

V této sekci jsou prezentovány experimenty prováděné na vytvořeném systému a na referenčním systému. Všechny experimenty byly pouštěny na výpočetním clusteru VUT FIT[1] na GPU.

V experimentech 4.3.1 byl systém natrénován pro překlad z češtiny (cs) do angličtiny (en). Tyto experimenty byly provedeny za účelem najití nejlepších hyperparametrů pro model a pro jeho otestování.

V experimentu 4.3.2 byl systém natrénován pro překlad z češtiny (cs) do angličtiny (en) za pomoci nejlepších parametrů zjištěných z předchozích experimentů.

V experimentu 4.3.3 byl systém natrénován pro překlad mezi více jazyky za pomoci nejlepších parametrů zjištěných z předchozích experimentů.

Pro všechny experimenty byly použity předučené word embeddings, jak pro vstupní tak výstupní jazyk, poskytnuté firmou Facebook[2].

Jako optimalizační metoda byl použit RMSprop ([25]) s koeficientem učení 0.001. Každý experiment běžel tak dlouho, dokud se 5 epoch nezlepšil výsledek loss funkce počítané pro validační dataset. Z experimentu se ukládají váhy modelu s nejlepším výsledkem loss funkce.

Použitá velikost jedné trénovací dávky je 128. Je použito generování dávek (3.3.3) a rozdělení dat podle velikosti (3.3.2).

Hodnotící metrikou je standardní skóre BLEU (2.5.1), protože je to nejčastěji používaná metrika a je použita i ve všech citovaných pracích. Pro jeho výpočet byl použit skript *multi-blue.pl* dodávaný s nástrojem Moses.

## 4.1 Použité datasety

Všechny použité datasety pocházejí z každoročně publikovaného překládacího úkolu konference o strojovém překladu **WMT** (dříve **w**orkshop on statistical **m**achine **t**ranslation). Konkrétně jsou použity datasety z WMT17[3] obsahující textová data z různých domén, jako jsou titulky, knihy, noviny a internet.

Stažené datasety byly rozdělený na několik menších datasetů, které spolu nesdílí páry vět:

[1] http://www.fit.vutbr.cz/CVT/cluster/

[2] https://github.com/facebookresearch/fastText/blob/master/pretrained-vectors.md

[3] http://data.statmt.org/wmt17/translation-task/preprocessed/

**Trénovací malý cs-en dataset:** Obsahuje milion párů vět. Je určený pro trénování v experimentech, pomocí kterých se hledají optimální hyperparametry modelu a pro trénování referenčního systému.

**Trénovací velký cs-en dataset:** Obsahuje skoro třicet čtyři milionů párů vět. Je určený pro finální otestování modelu s nejlepšími výsledky nad malým trénovacím datasetem.

**Validační cs-en dataset:** Obsahuje dva tisíce párů vět. Je určený pro testování modelu trénovaném s malým trénovacím datasetem za účelem hledání optimálních hyperparametrů modelu.

**Testovací cs-en dataset:** Obsahuje dva tisíce párů vět. Je určený pro finální otestování modelu natrénovaného s velkým trénovacím datasetem. Používá se místo validačního modelu aby se vyzkoušelo, že výsledný model je skutečně schopný generalizace a nebyl jen úzce přizpůsoben pro dobré výsledky s validačním datasetem.

**Trénovací en-de dataset:** Obsahuje milion párů vět. Je určený pro trénování modelu pro překlad nad více jazyky.

**Testovací en-de dataset:** Obsahuje dva tisíce párů vět. Je určený pro otestování překladu nad více jazyky.

**Testovací cs-de dataset:** Obsahuje dva tisíce párů vět. Je určený pro otestování zero-shot překladu, tedy překladu, pro který model nebyl explicitně natrénován.

V případě, že byl dataset v experimentu předzpracován pomocí subword-nmt pro rozdělení slov na menší jednotky, byl vždy přeložený text před vyhodnocením BLEU skóre převeden zpátky na celá slova.

## 4.2 Referenční systém v Moses

Systém (popsaný v 3.2) byl natrénován za účelem porovnání jeho výsledku s výsledky prezentovaného systému. Pro trénování byl použit **trénovací malý dataset** a pro vyhodnocení **testovací dataset**. Výsledek je v tabulce 4.1.

| systém | BLEU skóre |
|---|---|
| referenční v Moses | 23.08 |

Tabulka 4.1: Výsledky modelu natrénovaného pomocí statistického nástroje Moses.

## 4.3 Experimenty

### 4.3.1 Hledání optimálních hyperparametrů

V této sekci je popsán průběh a výsledky experimentů prováděných za účelem nalezení co nejlepších hyperparametrů modelu. Všechny experimenty jsou provedeny na překladu z češtiny do angličtiny za pomocí malého trénovacího cs-en datasetu a překlad je testován pomocí validačního cs-en datasetu.

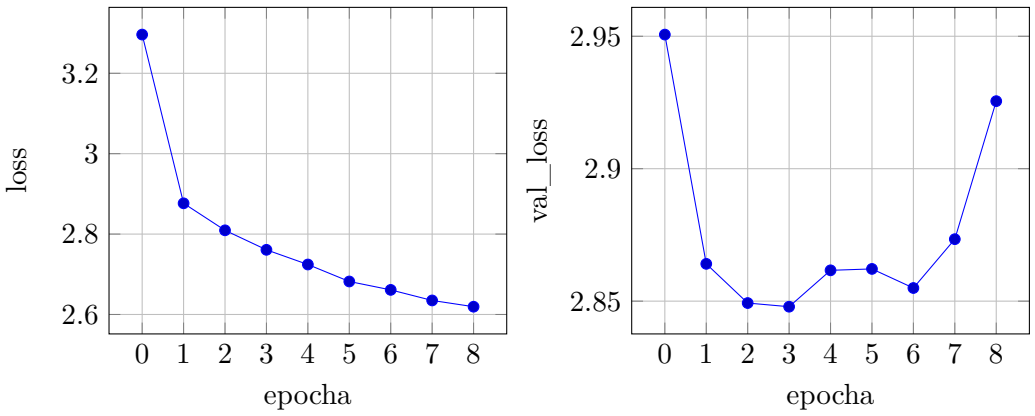
### 1. experiment – baseline

Model a jeho parametry z tohoto experimentu bude považován za baseline vůči kterému budou porovnány výsledky modelů z dalších experimentů, aby se dalo určit, k jak velkému došlo vylepšení.

Parametry:

- používají se celá slova
- jedna vrstva LSTM v enkodéru o velikosti 512
- jedna vrstva LSTM v dekodéru o velikosti 512
- velikost slovníku je omezena na 15000
- paprskové prohledávání o velikosti 1

Trénování bylo ukončeno po 9. epoše. Nejlepší dosažený výsledek **loss funkce** pro validační dataset je **2.855**. **Skóre BLEU** překladu vygenerovaného pro validační dataset je **7.26**. Pro možnost porovnání s výsledným nejlepším modelem byl proveden překlad i testovacího datasetu a to se **skóre 7.36**.



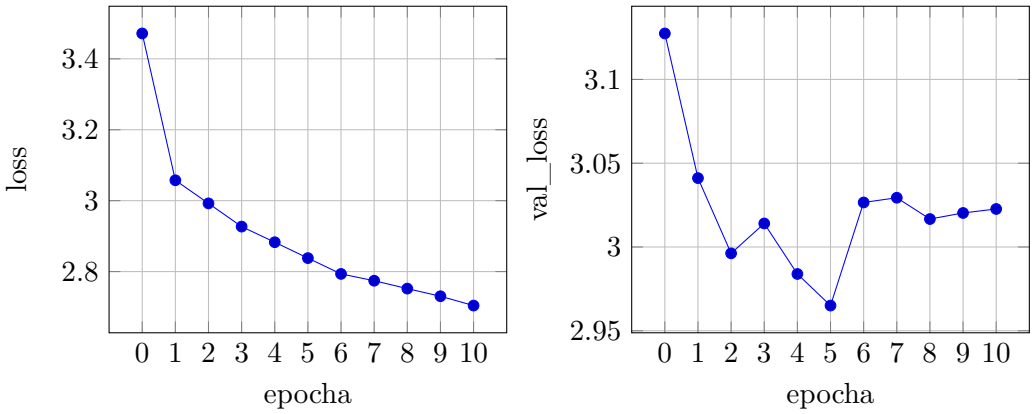

Obrázek 4.1: Průběh vývoje hodnot loss funkce po jednotlivých epochách. Levý graf je pro trénovací dataset a pravý pro validační dataset.

### 2. experiment

Parametry:

- používají se celá slova
- jedna vrstva LSTM v enkodéru o velikosti 512
- jedna vrstva LSTM v dekodéru o velikosti 512
- velikost slovníku je omezena na 30000
- paprskové prohledávání o velikosti 1

Trénování bylo ukončeno po 11. epoše. Nejlepší dosažený výsledek **loss funkce** pro validační dataset je **2.965**. **Skóre BLEU** překladu vygenerovaného pro validační dataset je **7.79**. To znamená, že zvětšení maximální velikosti slovníku za cenu značného navýšení velikosti modelu a času trénování přineslo o něco lepší výsledek.



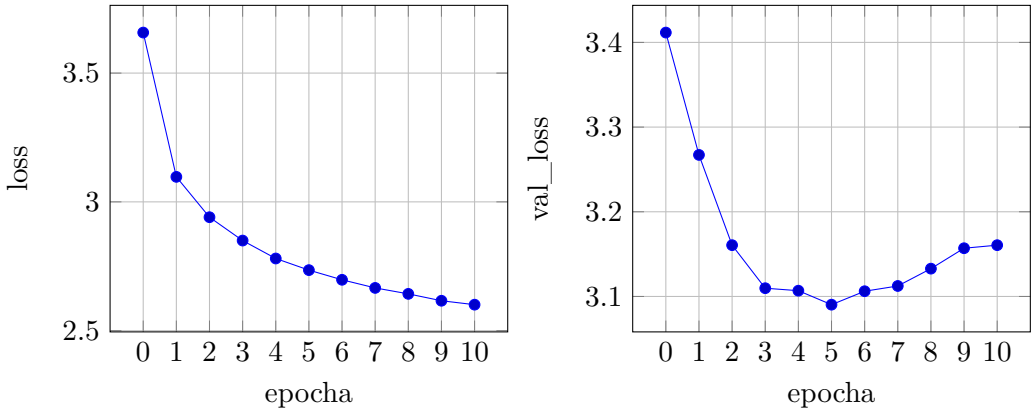

Obrázek 4.2: Průběh vývoje hodnot loss funkce po jednotlivých epochách. Levý graf je pro trénovací dataset a pravý pro validační dataset.

#### 3. experiment

Parametry:

- slova jsou rozdělena na menší jednotky
- jedna vrstva LSTM v enkodéru o velikosti 512
- jedna vrstva LSTM v dekodéru o velikosti 512
- velikost slovníku je omezena na 15000
- paprskové prohledávání o velikosti 1

Trénování bylo ukončeno po 11. epoše. Nejlepší dosažený výsledek **loss funkce** pro validační dataset je **3.090**. **Skóre BLEU** překladu vygenerovaného pro validační dataset je **8.37**. Při zachování parametrů z experimentu 4.3.1 a použití BPE je výsledné skóre lepší a to bez zvětšení velikosti modelu a navýšení doby trénování. Všechny další experimenty budou proto prováděny s použitím BPE a subword units.



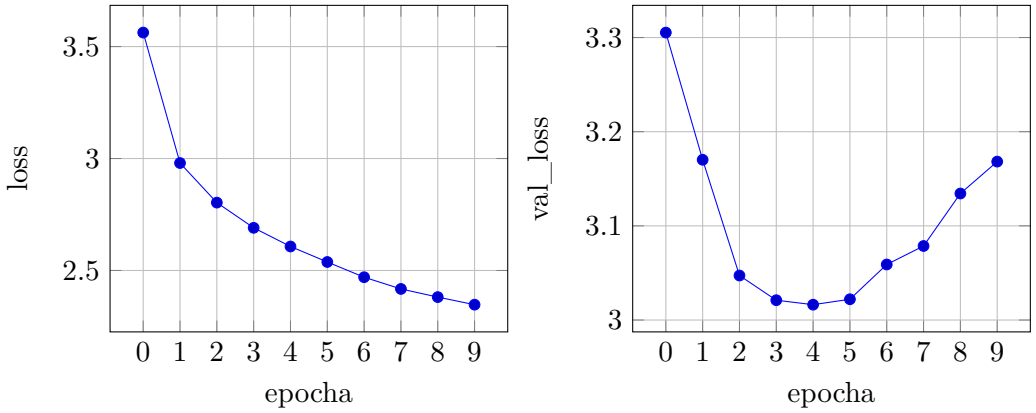

Obrázek 4.3: Průběh vývoje hodnot loss funkce po jednotlivých epochách. Levý graf je pro trénovací dataset a pravý pro validační dataset.

#### 4. experiment

Parametry:

- slova jsou rozdělena na menší jednotky
- jedna vrstva LSTM v enkodéru o velikosti 1000
- jedna vrstva LSTM v dekodéru o velikosti 1000
- velikost slovníku je omezena na 15000
- paprskové prohledávání o velikosti 1, 5, 8, 10, 12, 15

Trénování bylo ukončeno po 10. epoše. Nejlepší dosažený výsledek **loss funkce** pro validační dataset je **3.016**. Kromě zvýšení velikosti jednotek LSTM byl také testován vliv použité velikosti paprskového prohledávání při generování překladů. Výsledky jsou v tabulce 4.2 a ukázka zlepšení překladů je v tabulce 4.3. Jak navýšení počtu jednotek, tak velikost paprskového prohledávání přineslo lepší výsledek a bude tak používáno v dalších experimentech.




Obrázek 4.4: Průběh vývoje hodnot loss funkce po jednotlivých epochách. Levý graf je pro trénovací dataset a pravý pro validační dataset.

| velikost paprskové prohledávání | BLEU skóre |
|---|---|
| 1 | 8.75 |
| 5 | 9.34 |
| 8 | 9.4 |
| 10 | 9.41 |
| 12 | 9.43 |
| 15 | 9.47 |

Tabulka 4.2: Porovnání výsledků pro různou velikost paprskového prohledávání.

| | |
|---|---|
| původní věta v češtině | žádní pánové už nejsou. |
| překlad při velikosti 1 | no more gentlemen. |
| překlad při velikosti 5 | there are no more men. |
| překlad při velikosti 15 | there are no gentlemen no more. |

Tabulka 4.3: Ukázka zlepšení překladu při použití různých velikostí paprskového prohledávání.

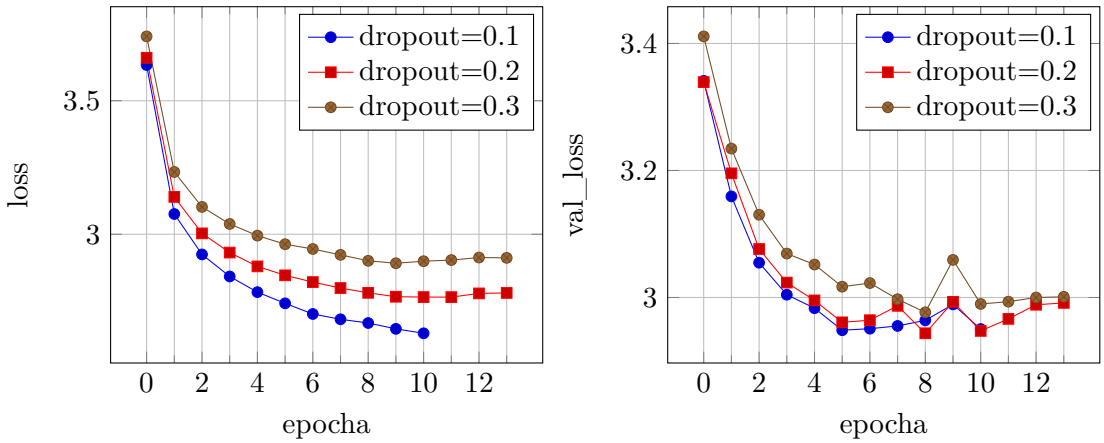
**5. experiment**

Parametry:

- slova jsou rozdělena na menší jednotky
- jedna vrstva LSTM v enkodéru o velikosti 1000
- jedna vrstva LSTM v dekodéru o velikosti 1000
- velikost slovníku je omezena na 15000

- paprskové prohledávání o velikosti
- dropout o velikosti 0.1, 0.2, 0.3

Pro dropout 0.1 bylo trénování ukončeno po 10. epoše. Pro hodnoty 0.2 a 0.3 po 13. epoše, protože model se může déle trénovat bez přetrénování. Nejlepší dosažený výsledek **loss funkce** pro validační dataset je **2.944**. Výsledky jsou v tabulce 4.4. Nejlepšího skóre bylo dosaženo pro dropout 0.1, takže tato hodnota bude použita v dalších experimentech.




Obrázek 4.5: Průběh vývoje hodnot loss funkce po jednotlivých epochách. Levý graf je pro trénovací dataset a pravý pro validační dataset.

| velikost dropout | BLEU skóre |
|---|---|
| 0.1 | 10.02 |
| 0.2 | 9.39 |
| 0.3 | 9.62 |

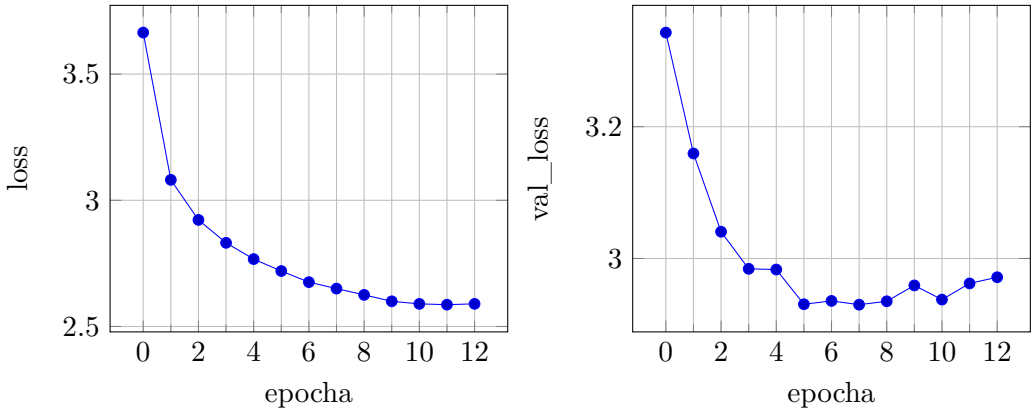
Tabulka 4.4: Porovnání výsledků při použití různé hodnoty dropout.

**6. experiment**

Parametry:

- slova jsou rozdělena na menší jednotky
- dvě vrstvy LSTM v enkodéru o velikosti 1000
- jedna vrstva LSTM v dekodéru o velikosti 1000
- velikost slovníku je omezena na 15000
- paprskové prohledávání o velikosti 15

Trénování bylo ukončeno po 13. epoše. Nejlepší dosažený výsledek **loss funkce** pro validační dataset je **2.930**. **Skóre BLEU** překladu vygenerovaného pro validační dataset je **9.99**.



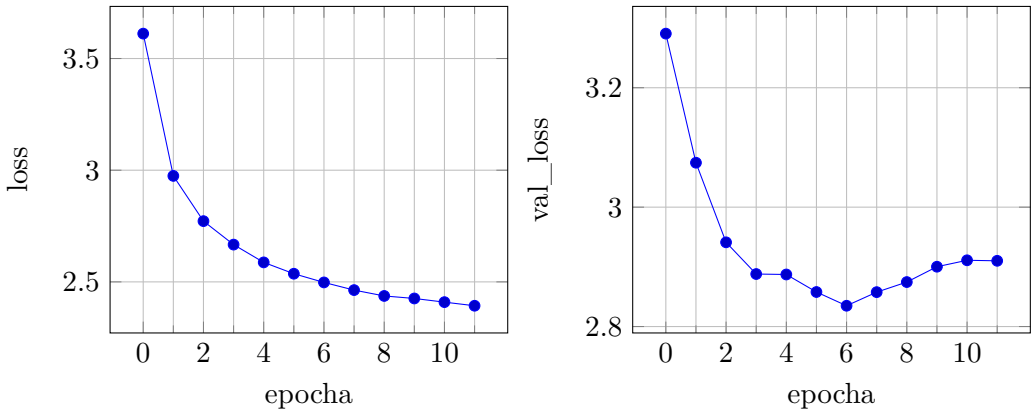

Obrázek 4.6: Průběh vývoje hodnot loss funkce po jednotlivých epochách. Levý graf je pro trénovací dataset a pravý pro validační dataset.

### 7. experiment

Parametry:

- slova jsou rozdělena na menší jednotky
- jedna vrstva LSTM v enkodéru o velikosti 1000
- dvě vrstvy LSTM v dekodéru o velikosti 1000
- velikost slovníku je omezena na 15000
- paprskové prohledávání o velikosti 15

Trénování bylo ukončeno po 12. epoše. Nejlepší dosažený výsledek **loss funkce** pro validační dataset je **2.835**. **Skóre BLEU** překladu vygenerovaného pro validační dataset je **10.14**.



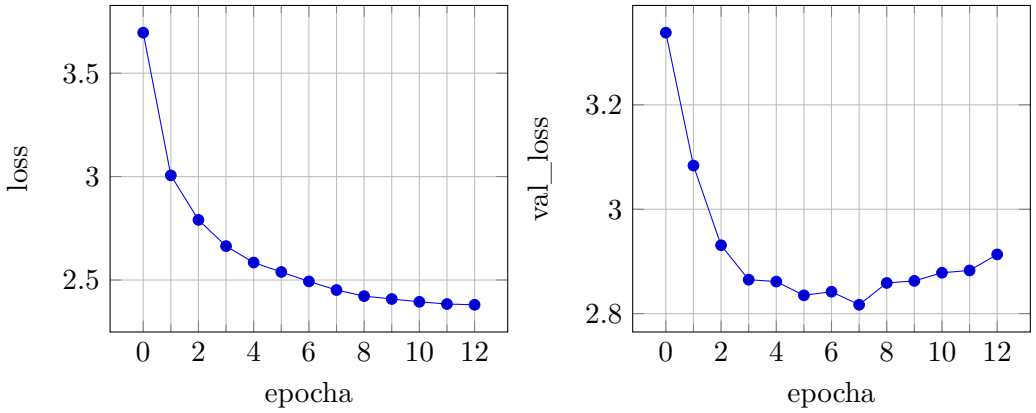

Obrázek 4.7: Průběh vývoje hodnot loss funkce po jednotlivých epochách. Levý graf je pro trénovací dataset a pravý pro validační dataset.

**8. experiment**

Parametry:

- slova jsou rozdělena na menší jednotky
- dvě vrstvy LSTM v enkodéru o velikosti 1000
- dvě vrstvy LSTM v dekodéru o velikosti 1000
- velikost slovníku je omezena na 15000
- paprskové prohledávání o velikosti 15

Trénování bylo ukončeno po 13. epoše. Nejlepší dosažený výsledek **loss funkce** pro validační dataset je **2.817**. **Skóre BLEU** překladu vygenerovaného pro validační dataset je **10.42**.



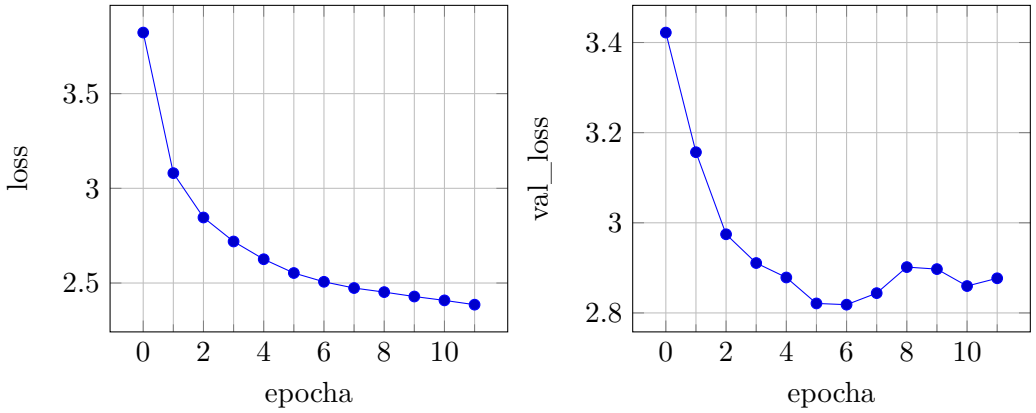

Obrázek 4.8: Průběh vývoje hodnot loss funkce po jednotlivých epochách. Levý graf je pro trénovací dataset a pravý pro validační dataset.

**9. experiment**

Parametry:

- slova jsou rozdělena na menší jednotky
- čtyři vrstvy LSTM v enkodéru o velikosti 1000
- čtyři vrstvy LSTM v dekodéru o velikosti 1000
- velikost slovníku je omezena na 15000
- paprskové prohledávání o velikosti 15

Trénování bylo ukončeno po 12. epoše. Nejlepší dosažený výsledek **loss funkce** pro validační dataset je **2.818**. **Skóre BLEU** překladu vygenerovaného pro validační dataset je **9.78**, takže nejlepším modelem zůstává model z experimentu 4.3.1.




Obrázek 4.9: Průběh vývoje hodnot loss funkce po jednotlivých epochách. Levý graf je pro trénovací dataset a pravý pro validační dataset.

### 4.3.2 Otestování nejlepšího modelu

Modelem s nejlepšími dosaženými výsledky je model s parametry z experimentu 4.3.1. Výsledky jeho překladu pro testovací dataset jsou v tabulce 4.5.

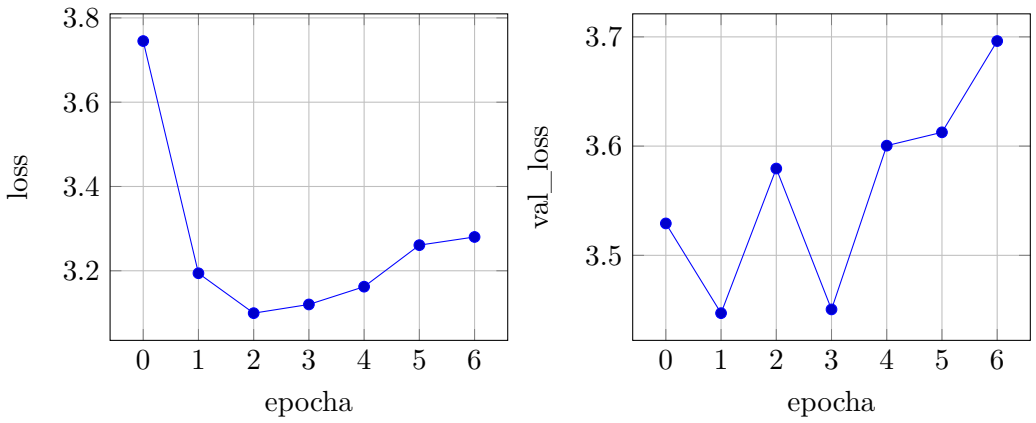
| model | BLEU skóre |
|---|---|
| nejlepší vytvořený | 9.87 |

Tabulka 4.5: Výsledné skóre BLEU nad testovacím datasetem. Překlad byl proveden s modelem, který dosáhl nejlepších výsledků pro validační dataset.

### 4.3.3 Překlad mezi více jazyky

V této sekci je popsán experiment, kdy se jeden model trénuje na překlad vícero jazykových párů. Trénovací dataset vznikl spojením a promícháním trénovacího cs-en datasetu a trénovacího en-de datasetu. Před věty ve výchozím jazyce se přidal token jazyku do kterého se překládají, tak jak je uvedeno v sekci 2.4.3. Model natrénovaný na těchto datech by tedy měl umět překládat z češtiny do angličtiny, z angličtiny do němčiny a možná trochu z češtiny do němčiny.

Použité parametry jsou stejné jako v experimentu 4.3.1, tedy parametry se kterými model dosáhl nejlepších výsledků. Trénování bylo ukončeno po 7. epoše. Nejlepší dosažený výsledek **loss funkce** pro validační dataset je **3.447**. **Skóre BLEU** pro všechny testované jazykové páry jsou v tabulce 4.6.




Obrázek 4.10: Průběh vývoje hodnot loss funkce po jednotlivých epochách. Levý graf je pro trénovací dataset a pravý pro validační dataset.

| jazykový pár | skóre BLEU |
|---|---|
| $Cs \rightarrow En$ | 7.10 |
| $En \rightarrow De$ | 7.39 |
| $Cs \rightarrow De$ | 0.14 |

Tabulka 4.6: Výsledné skóre BLEU pro testovací datasety různých jazykových párů. Překlad byl proveden jedním modelem natrénovaným na párech $Cs \rightarrow En$ a $En \rightarrow De$. Výsledky ukazují, že model je schopný překládat pro oba natrénované páry, ale za cenu horšího výsledku pro $Cs \rightarrow En$, než když byl natrénován jen pro tento jeden pár. Zero-shot překlad páru $Cs \rightarrow De$ nevykazuje úspěch.

### 4.3.4 Výsledky

Všechny dosažené výsledky získané s Cs⇒En testovacím datasetem jsou shrnuty v tabulce 4.7. Ukázky překladů jsou v tabulce 4.8. Doposud se nepodařilo objasnit proč vytvořený model nedosahuje lepších výsledků, a bylo by vhodné to dále v budoucnu zkoumat.

| model | skóre BLEU |
|---|---|
| Moses | 23.08 |
| baseline | 7.36 |
| nejúspěšnější | 9.87 |
| více jazykový | 7.10 |

Tabulka 4.7: Výsledky různých modelů pro překlad cs-en testovacího datasetu. Žádný z vytvořených modelů nedosáhl výsledků referenčního modelu vytvořeného v nástroji Moses. Nejúspěšnější model z experimentu 4.3.1 dosáhl zlepšení vůči baseline modelu, což se projevuje i na ukázkách překladu 4.8. Model natrénovaný pro překlad mezi více jazyky má skóre horší jak baseline model pro překlad $Cs \rightarrow En$, ale za to dosahuje podobného výsledku i pro překlad $Cs \rightarrow De$.

| originální text |
|---|
| 1. budoucnost se jevila černě. |
| 2. někdo mu musí dát lekci. |
| 3. nejdůležitější výsledky studie jsou uvedeny v tabulce 4 a na obrázku 3. |
| 4. uvidíme, jak se ta schůze bude odvíjet. |
| 5. Evropská unie je druhým největším obchodním partnerem ASEAN. |
| 6. 13. prosinec |

| originální překlad |
|---|
| 1. the future was grey. |
| 2. somebody's got to teach him a lesson. |
| 3. the key results of the trial are listed in Table 4 and Figure 3. |
| 4. see how the meeting goes. |
| 5. the European Union is ASEAN's second largest trading partner. |
| 6. December 13 |

| překlad referenčního systému Moses |
|---|
| 1. the jevila black. |
| 2. someone must give him a lesson. |
| 3. the study results are provided in Table 4 and the picture 3. |
| 4. we'll see how the meeting is odvíjet. |
| 5. the European Union's second largest partner ASEAN. |
| 6. 13 December |

| překlad baseline modelu |
|---|
| 1. the future future became _UNK. |
| 2. somebody must teach him a lesson. |
| 3. the following involved interested in the results of the information and the _UNK are available in the. |
| 4. see how the _UNK will be happening. |
| 5. the European Union is the European Union _UNK. |
| 6. 13 _UNK |

| překlad nejlepšího modelu |
|---|
| 1. the future was red. |
| 2. someone needs to give him a lesson. |
| 3. the most important results of the two and two are available in the level of the 3. |
| 4. let's see how the meeting goes. |
| 5. the European Agreement is the most attractive partner of the European Union. |
| 6. 13. |

| překlad více jazykového modelu |
|---|
| 1. the whole body was ty. |
| 2. someone's got to take care of him. |
| 3. the same |
| 4. we'll see how it's going to be done. |
| 5. the the EU's the right of the ASEE. |
| 6. 13 . ( 1 ) |

Tabulka 4.8: Ukázky překladu vět z testovacího datasetu. V první tabulce jsou originální věty v českém jazyce, v druhé tabulce jsou originální anglické překlady. Ve třetí tabulce je překlad, který vytvořil referenční systém vytvořený v nástroji Moses, ve čtvrté překlady baseline modelu, v páté tabulce jsou překlady vytvořené nejlepším dosaženým modelem a v poslední tabulce jsou překlady modelu trénovaného nad více jazyky. Je vidět že překlad z Moses si pomáhá přenášením neznámých slov do přeložené věty, což v případě například jmen může pomoci, ale v jiných případech jako je první věta může být ke škodě. I přes nevelkou změnu v BLEU skóre je vidět značné zlepšení mezi větami přeloženými baseline modelem a modelem s nejlepším skóre.

# Kapitola 5

# Závěr

Cílem této práce bylo prozkoumat a vytvořit systém pro strojový překlad s pomocí neuronových sítí. Systém byl vytvořen s pomocí architektury enkodér-dekodér, která používá jednotek LSTM pro překlad celých vět mezi dvěmi jazyky. Výsledkem je balíček pro Python *nmt*, který je publikován na serveru github `github.com/jojkos/neural-machine-translation` pro veřejné použití.

Pro trénink a otestování modelu vytvořeného vzniklým systémem byla zvolena data z konference WMT. Byla provedena řada experimentů pro získání hyperparametrů, se kterými model dosahuje nejlepších výsledků. Výsledky jsou hodnoceny pomocí standardní metriky BLEU. Model s nejlepším výkonem dosáhl při překladu $Cs \rightarrow En$ skóre **9.87**. Pro porovnání byl na stejných trénovacích datech natrénován systém vytvořený nástrojem Moses, který pro stejná testovací data dosáhl skóre **23.08**, tedy lepšího výsledku.

Dále byl proveden experiment, ve kterém byl model natrénován pro překlad mezi více jazykovými páry, konkrétně $Cs \rightarrow En$ a $En \rightarrow De$. Tento model dosáhl výsledků **7.10** pro první pár a **7.39** pro druhý. Za cenu mírného zhoršení překladu pro $Cs \rightarrow En$ je jeden model schopný překládat mezi více jazyky. Zero-shot překlad pro pár $Cs \rightarrow De$ byl neúspěšný.

Aktuálně dosažené výsledky sice ukazují, že systém funguje, ale nejsou dostačující a nedosahují výsledků aktuálního state-of-the-art ani systému Moses, se kterými jsou porovnávány. Bylo by tedy potřeba zjistit, čím je to způsobené a systém vylepšit.

Dále by v budoucnu bylo vhodným rozšířením přidat modul attention, který umožňuje modelu lépe se vypořádat s dlouhými větami a závislosti mezi jednotlivými slovy.

Práce byla prezentována v rámci konference Excel@FIT 2018.

# 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] et al., Y. W.: 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

[3] 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

[4] 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

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

[6] 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

[7] Doddington, G.: Automatic Evaluation of Machine Translation Quality Using N-gram Co-occurrence Statistics. In *Proceedings of the Second International Conference on Human Language Technology Research*, HLT '02, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002, s. 138–145.
URL http://dl.acm.org/citation.cfm?id=1289189.1289273

[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] 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

[11] Han, L.: LEPOR: An Augmented Machine Translation Evaluation Metric. *CoRR*, ročník abs/1703.08748, 2017, 1703.08748.
URL http://arxiv.org/abs/1703.08748

[12] 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

[13] 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

[14] 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

[15] 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

[16] Lavie, A.; Agarwal, A.: Meteor: An Automatic Metric for MT Evaluation with High Levels of Correlation with Human Judgments. In *Proceedings of the Second Workshop on Statistical Machine Translation*, StatMT '07, Stroudsburg, PA, USA: Association for Computational Linguistics, 2007, s. 228–231.
URL http://dl.acm.org/citation.cfm?id=1626355.1626389

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

[18] 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

[19] 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

[20] 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

[21] 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

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

[23] 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

[24] 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

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

[26] 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

[27] 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

[28] 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

# Příloha A

# Obsah přiloženého média

- text práce ve formátu PDF a zdrojové soubory pro prostředí LaTeX
- video ukázka výsledků práce
- ilustrativní plakát reprezentující výsledky práce
- zdrojové soubory balíčku *nmt*

  **složka tests** obsahuje testy spustitelné pomocí *pytest*

  **složka docs/_build** obsahuje dokumentaci balíčku vygenerovanou pomocí *sphinx*
- skript *main.py*, který byl použit při experimentech pro volání metod balíčku *nmt*
- složku s experimenty, která obsahuje jejich průběh, výsledný model ve formátu *h5* a překlady

# Příloha B

# Parametry třídy Translator

**clear** Jestli před začátkem trénování mají být smazané staré logy a uložený model

**dropout** Velikost dropout

**log_folder** Cesta ke složce, do které se mají ukládat záznamy

**max_source_embedding_num** Maximální počet řádků načtených ze zdrojových embeddings

**max_target_embedding_num** Maximální počet řádků načtených z cílových embeddings

**max_source_vocab_size** Maximální velikost zdrojového slovníku

**max_target_vocab_size** Maximální velikost cílového slovníku

**model_file** Pojmenování souboru s modelem

**model_folder** Pojmenování složky do které se uloží model

**num_decoder_layers** Počet vrstev v dekodéru

**num_encoder_layers** Počet vrstev v enkodéru

**num_test_samples** Kolik se má použít řádků z testovacího datasetu

**num_training_samples** Kolik se má použít řádků z trénovacího datasetu

**num_units** Počet jednotek LSTM pro vrstvy enkodéru a dekodéru

**source_embedding_dim** Rozměr zdrojových embeddings

**source_embedding_path** Cesta k souboru se zdrojovými embeddings

**source_lang** Zdrojový jazyk (koncovka datasetu)

**target_embedding_dim** Rozměr cílových embeddings

**source_target_path** Cesta k souboru s cílovými embeddings

**target_lang** Cílový jazyk (koncovka datasetu)

**test_dataset** Cesta k testovacímu datasetu

**tokenize** Jestli se má provést před tréninkem tokenizace datasetů

**training_dataset** Cesta k trénovacímu datasetu