

# SFC projekt: Demonstrace činnosti GRU sítě

Bc. Vojtěch Vlach (xvlach22)

2023/24

## 1 Úvod

V tomto projektu jsem se rozhodl demonstrovat činnost GRU sítě pomocí jazykové úlohy. A to rozdělování slov na slabiky, vytvořil jsem si vlastní dataset a využil jsem implementaci GRU v knihovně PyTorch, pro trénování jsou využit domácí počítač bez GPU a částečně Google Colab s GPU runtime.

## 2 Popis úlohy

Pro tento projekt jsem si zvolil úlohu rozdělení slova na slabiky. Omezil jsem se pouze na česká slova, kde mohu aspoň trochu posoudit správnost intuitivně. Přesněji se tedy jedná o problém, kde na vstupu je české slovo a na výstupu by mělo být slovo rozdělené na slabiky. Nenalezl jsem mnoho literatury, která by se tímto problémem zabývala, předpokládám, že je to úloha moc jednoduchá na to, aby se na tom dal zveřejňovat seriózní článek. Výjimku tvoří např. Hyphenation using Deep Neural Network<sup>1</sup>.

### 2.1 Získání datasetu

Pro výrobu datasetu jsem využil fonetický korpus Češtiny<sup>2</sup>, který se zabývá kromě fonetické výslovnosti češtiny také četností výskytu fonémů, slov a slabik. Konkrétně jsem využil následující sadu: Lexemes from Slovník spisovné češtiny, 4th edition, 2005, která obsahuje právě fonetický přepis a jeho rozdělení na slabiky, které jsem využil. Slova v datasetu byly ve formátu, které jsou naznačeny ve sloupcích **Původní slovo**, **Fonetický přepis** a **Fonetické slabiky** viz tabulka 1.

Z těchto sloupců jsem jednoduchým algoritmem získal přepis na slabiky. Program procházel sloupce **Původní slovo** a **Fonetické slabiky** současně, nahrazoval fonetické výrazy původními písmeny a zanechal u nich informaci o slabikách. Více viz Python Notebook `python-scripts/Dataset_creating.ipynb`.

Pro vytvoření samotných vstupů sítě a ground-truths jsem zvolil postup, kde síť detekuje konec slabiky a poslední písmeno nahradí příslušným tokenem (@). Doplnění oddělovače do výsledného slova (např. -) zajistí jednoduchý algoritmus. Tento způsob jsem zvolil proto, aby se nezpožďovaly výstupní písmena od těch vstupních. Tímto způsobem se síť může učit závislost pouze na několika málo posledních znacích. V opačném případě by zde vznikala závislost na znacích s větším rozestupem, což teoreticky zesložituje úlohu. Výsledkem tohoto zpracování byly sloupce **Vstup při trénování** a **Ground truth**, viz tabulka 1

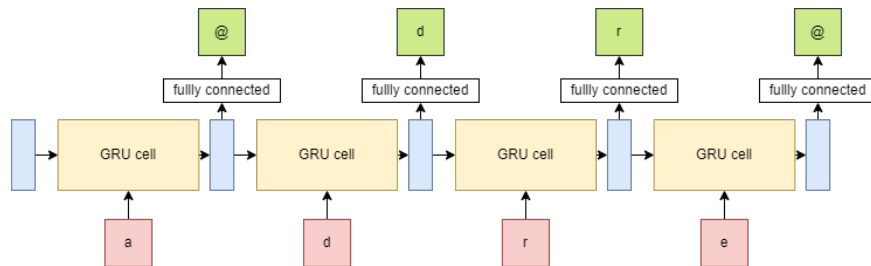
---

<sup>1</sup>Gergely et al. Hyphenation using Deep Neural Network

<sup>2</sup>The Phonological Corpus of Czech

Tabulka 1: Tabulka 1: Proces získání datové sady.

Původní slovo	Fonetický přepis	Fonetické slabiky
adresa	adresa	a.dre.sa
antimilitaristický	antimilitariStiTSkí	an.ti.mi.li.ta.riS.tiTS.kí
antikvariát	antikvarijáT	an.ti.kva.ri.jáT
popřípadě	popřípadě	po.pří.pa.ďe
Přepis na slabiky	Vstup při trénování	Ground truth
a-dre-sa	adresa	@dr@sa
an-ti-mi-li-ta-ris-tic-ký	antimilitaristicky	a@t@m@l@t@ri@ti@ky
an-ti-kva-ri-át	antikvariát	a@t@kv@r@@t
po-pří-pa-ďe	popripade	p@pr@p@de



Obrázek 1: Obrázek 1: Znázornění začátku činnosti sítě pro slovo **adresa**.

Červená: vstup, žlutá: GRU buňka, bílá: plně propojená vrstva, modrá: vektor skrytého stavu, zelená: výstup sítě.

### 3 Implementace

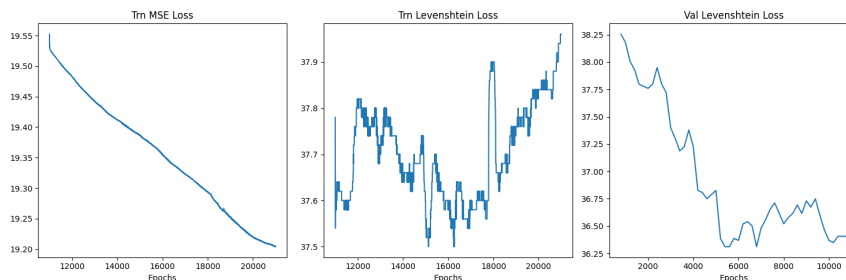
Všechny implementované skripty popsané v této kapitole jsou k nalezení ve složce **src**.

Pro svoji implementaci jsem se inspiroval článkem Gated Recurrent Unit (GRU) With PyTorch<sup>3</sup>, který ukazuje možnost využití pro predikci časové řady. Jeho řešení jsem si upravil, aby vyhovovalo mému úloze.

**GRUNet.** Základní část mé implementace je třída **GRUNet** (viz soubor **src/gnu\_pytorch.py**, která definuje jednoduchou rekurentní síť s plně propojenou vrstvou na výstupu. Struktura sítě je definována jako GRU vrstva (+ ReLU aktivace) a plně propojená vrstva pro agregování vektoru skrytého stavu na formát výstupu. Vizualizování na Obrázku 1.

**Funkce train.** Dále jsem naimplementoval funkci **train**, která spouští trénování pomocí zadaných parametřů. Umožňuje procházet dataset rozdělený na dávky a pro každý spočítá chybu a provede optimalizační krok. Jako chybovou funkci využívá **Mean Square Error** a k optimalizačním krokům využívá optimalizátor **Adam**. Dále umožňuje nahrát dříve uložený model a pokračovat v trénování. V průběhu shromažďuje statistiky o chybách a jednou za **save\_step** epoch uloží aktuální stav modelu a grafy znázorňující průběh chyby na trénovací i validační sadě.

<sup>3</sup>Loye Gabriel, 2019: Gated Recurrent Unit (GRU) With PyTorch



Obrázek 2: Obrázek 2: Průběh trénování. Zleva: Průběh chyby Mean-Square-Error, průběh Levenshteinovy vzdálenosti na trénovací sadě, průběh Levenshteinovy vzdálenosti na validační sadě.

**Další drobnosti.** Další části důležité pro běh trénování jsou tyto soubory `charset.py` (obsahující definici kódování slov ze `stringu` na `float tensor`), `helpers.py` (obsahující podpůrné funkce pro trénování: počítání levenshteinovy vzdálenosti, ukládání a nahrávání modelu a vytváření grafů), `dataset.py` (obsahující stejnojmennou třídu, která nahrává soubor a zpracovává ho na vstupy sítě a ground-truths a poskytuje možnost iterování přes batche),

## 4 Spuštění + ovládání

Pro spuštění by měl stačit `python3.8+`. První krok je nainstalování všech potřebných knihoven pomocí skriptu `install.sh`. Dále stačí spouštět skripty pomocí příkazové řádky.

Pro spuštění trénování je následující příkaz:

```
python3 src/gru_pytorch.py
```

Tento příkaz spustí trénování podle nastavených parametrů přímo v kódu, např:

```
train(trn_dataset, val_dataset, learn_rate=0.001, device=device,
batch_size=64, epochs=4000, save_step=500, view_step=100,
load_model_path='models')
```

Pro manuální testování modelu je připraven soubor `src/inference.py`, který nabízí jednoduchý CLI interface k modelu. Je spustitelný pomocí příkazu:

```
python3 src/inference.py
```

pozn. pro úpravy a spouštění python notebooku `Dataset_creating.py` je navíc potřeba prostředí pro Jupyter Notebook.

## 5 Zhodnocení

V požadovaném čase se mi nepodařilo natrénovat model, který by měl uspokojivé výsledky, ale podle grafu níže graf na Obrázku 2 lze vidět, že pomocí trénování se model postupně zlepšoval a kdyby měl více času, nebo lepší hardware, je možné, že by dosáhl i lepších výsledků.

V tomto stavu se tedy lze bavit pouze o demonstraci postupného zmenšování chyby, ale nikoliv praktického použití. Příklady viz Tabulka 2

Tabulka 2: Tabulka 2: Příklad výstupů nedotřénovaného modelu.

<b>Vstup</b>	<b>Ground-truth</b>	<b>Výstup</b>
vyhotovovat	v@h@t@v@vat _____	uxquvvuuvdt _____
cecko	ce@ko _____	epmnu _____
vynalez	v@n@lez _____	uxrtnpw _____