Neuronové sítě 2 - Neuronové sítě a text

18NES2 - 12.hodina, ZS 2024/25

Zuzana Petříčková

18. prosince 2024

Neuronové sítě 2 - Neuronové sítě a text

Co jsme dělali minule: Neuronové sítě a sekvenční data

- Sekvenční data a jejich reprezentace (v Kerasu a tensorflow)
- Sekvenční data a jejich zpracování pomocí MLP, CNN
- Rekurentní neuronové sítě (RNN)
  - Základní model: jednoduchá (Vanilla /Elmann) RNN
  - LSTM
  - GRU

Zbývá dokončit:

- Pokročilejší aspekty RNN: zobecňování (např. rekurentní dropout), vícevrstvé RNN, obousměrné RNN,...

Hluboké učení a text

Zpracování přirozeného jazyka (NLP, natural language processing)

- Velmi široká oblast AI, která se v současnosti rychle rozvíjí
- Zaměřuje se na porozumění, analýzu a generování přirozeného jazyka

Příklady úloh:

- **many-to-one**:
  - klasifikace textu, analýza sentimentu
  - filtrování obsahu (např. detekce spamu, škodlivého obsahu), detekce klíčových slov
  - modelování textu: predikce dalšího slova, korekce pravopisu
- **many-to-many**:
  - strojový překlad (např. Google Translate)
  - shrnutí textu (automatické sumarizace)
  - generování textu (GPT, Bert), chatboti (ChatGPT,...)

Architektury hlubokého učení pro text

MLP:

- neberou v úvahu sekvenčnost a vzájemné závislosti slov → lze řešit pomocí předzpracování vstupních dat (využití n-gramů)
- vhodné, pokud máme málo vzorků dat: klasifikace

RNN (Seq2Seq, LSTM, GRU):

- Cca. od roku 2014, vrchol popularity v letech 2015–2017 : strojový překlad, analýza sentimentu, predikce dalšího slova.
- Zohledňují sekvenčnost dat, ale mají problémy s GPU akcelerací a paralelizací kvůli své sekvenční povaze.

CNN (TextCNN,...):

- V NLP neprávem často opomíjený, přitom efektivní model,
- Extrahují lokální vzory a zachycují souvislosti mezi sousedními slovy
- Vhodné jen pro některé úlohy: klasifikace, filtrování obsahu, detekce klíčových slov.

...

Architektury hlubokého učení pro text

Transformery (BERT, GPT,...):

- Od roku 2017 dominují NLP.
- Udržují kontext celé sekvence pomocí **self-attention** mechanismu.
- Výhody:
  - Výkonnost, paralelizace
  - Schopnost zachytit dlouhé závislosti.
- Nevýhody:
  - Vysoká náročnost na pamět i GPU zdroje
  - Vysoká energetická náročnost (učení i použití)
  - Vyžadují větší množství trénovacích dat

Architektury hlubokého učení pro text

Dva možné přístupy k textovým datům:

- text je množina slov: **bag of words**:
  - model implicitně nebere v úvahu vzájemnou polohu slov v textu
  - jednoduchý model (např. MLP)
  - již jsme si ukazovali
- text je posloupnost slov
  - bereme v úvahu vzájemnou polohu slov v textu, kontext, případně sémantické vztahy
  - CNN, RNN (sekvenční modely), Transformery

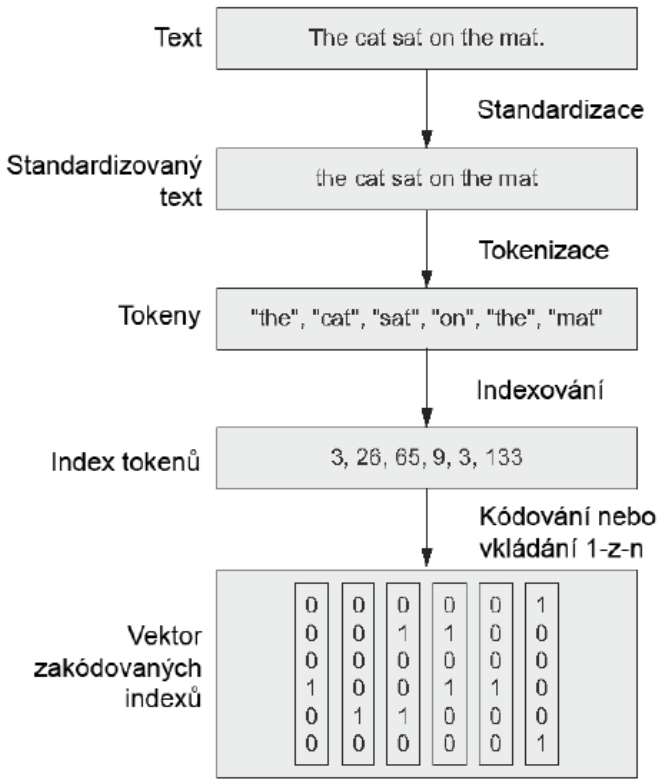
$\rightarrow$ různé reprezentace dat

Jaký model zvolit?

- Závisí na úloze: jak dlouhé jsou sekvence, kolik máme k dispozici trénovacích dat, jaké máme k dispozici výpočetní prostředky
- Pro ,,malou" úlohu volíme ,,jednodušší" model

Příprava textových dat

1. **Standardizace:** např. převod na malá písmena, odstranění interpunkce, převod slov do základního tvaru (stemming),...
2. **Tokenizace:** text rozdělíme na jednotky: znaky, části slov,slova nebo skupiny slov (**n-gramy**)
3. **Vektorizace**: text převedeme na číselný vektor
   - **Indexace tokenů**
   - Samotná vektorizace
   - Kódování vektorů



Kódování 1-z-n (Word index)

Zdroj: F. Chollet: Deep learning v jazyku Python, obr 11.1

Příprava textových dat

Tokenizace: text rozdělíme na tokeny

1. **slova nebo podslova**: hodí se pro sekvenční modely (RNN, Transformer), CNN
2. **n-gramy**: pro modely, co berou vstup jako množinu slov (MLP)
   - 2-gramy: "the cat sat on the mat" → "the", "the cat", "cat", "cat sat", "sat", "sat on", "on", "on the", "the", "the mat", "mat"
   - 3-gramy: "the", "the cat", "the cat sat", "cat", "cat on", "cat on the", ...,"on the mat", "the", "the mat", "mat"
3. **znaky**: speciální případy: japonština, čínština, rozpoznávání, korekce překlepů, ,,slova mimo slovník" OOV (out-of-vocabulary)

Příprava textových dat

Indexace:

- vytvoříme **slovník** všech tokenů (v trénovací množině, nebo v nějakém větším textovém korpusu),
- každému tokenu přiřadíme jednoznačné číslo (index)
- velikost slovníku omezíme na *n* nejčastějších slov (efektivita, lepší učení)
- konvence:
  - index ,,0": "nejsem token"
  - index ,,1": ,,jsem token, ale nejsem indexovaný"
  - vyšší indexy: nejčastější token má index 2, pořadí tokenů je podle jejich četnosi (sestupně)
  - ,,0" se používá jako vycpávka na konci - **padding** (pokud požadujeme, aby všechny sekvence měly stejnou délku)

Vektorizace:

- Text převedeme na posloupnost (vektor) indexů jednotlivých tokenů: "the cat sat on the mat" $\rightarrow$ [3, 28, 65, 9, 3, 1, 0]

Příprava textových dat

Vektorizace v Kerasu:

- Speciální vrstva: **TextVectorization**.
- Klíčové parametry:
  - **max_tokens**: Maximální počet tokenů (např. 200 000 nejčastějších slov).
  - **output_mode**: Způsob vektorizace: posloupnost ("int") nebo bag of words ("binary", "count", "tf-idf").
  - **output_sequence_length**: Maximální délka sekvence (např. 100 nebo 200 tokenů).
  - **ngrams**: Kolika-gramy chceme použít.
- Aplikace na data pomocí metody **adapt()**.

Příprava textových dat

Kódování vektorizovaných dat:

- vektory tvaru [3, 28, 65, 9, 3, 0] potřebujeme převést na tenzory, co budou vstupy neuronové sítě

Dva možné přístupy:

1. **text je množina slov (nebo n-gramů): bag of words**:
   - každý token reprezentujeme jako one-hot vektor dimenze jako je počet tokenů ve slovníku [0,..,0,1,0,...,0]
   - text pak reprezentujeme jako multi-hot vektor stejné dimenze [...0,1,1,0,1,0...]
   - varianta TF-IDF: váží každý token podle jeho četnosti v textu a v celém korpusu [...0,2.8,1.5,0,0.9,0...]
2. text je posloupnost slov

Příprava textových dat

Kódování vektorizovaných dat:

- vektory tvaru [3, 28, 65, 9, 3, 0] potřebujeme převést na tenzory, co budou vstupy neuronové sítě

Dva možné přístupy:

1. text je množina slov
2. **text je posloupnost slov**
   - každé slovo v textu opět můžeme reprezentovat jako one-hot vektor tvaru [0,..,0,1,0,...,0] $\rightarrow$ text je reprezentován pomocí matice 0 a 1 tvaru (délka sekvence, počet slov ve slovníku) **1-z-n**
   - častěji se používá **word embedding**

Word embedding

Nevýhody one-hot kódování

- zbytečné prostorové nároky: velká matice (délka sekvence $\times$ velikost slovníku), velmi řídká (skoro samé 0)
- nevýhoda: v tomto prostoru je vzdálenost všech dvojic slov stejná

Word embedding

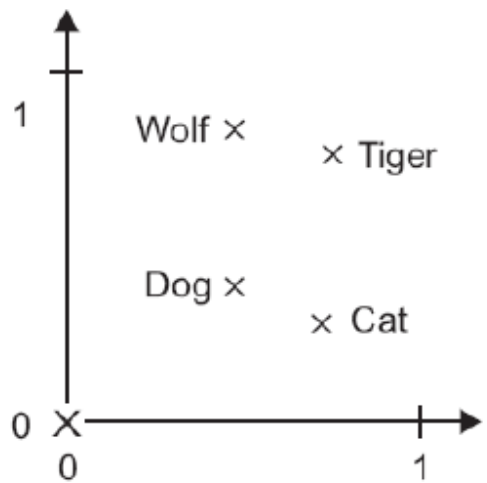
- každé slovo je reprezentováno jako **hustý** vektor malé dimenze
- slova s podobným významem jsou v tomto prostoru blízko sebe.
- bere v úvahu sémantiku: význam, kontext nebo gramatické role slov

Word embedding

Zjednodušeně: významové dimenze

Každá dimenze vektoru představuje nějakou vlastnost slova:

- sémantická podobnost, např. domácí mazlíček: pes: 1.0, kočka:1.0, had:0.5, vorvaň: 0.2, baterie:0.0
- gramatické vztahy: sloveso, přísudek, kořen slova
- analogické vztahy: "král - muž + žena = královna"
  "král + plural = králové"



Zdroj: F. Chollet: Deep learning v jazyku Python, obr 11.3

Embedding vrstva

- jednoduchá vyhledávací tabulka (look-up table)
- mapuje indexy tokenů na husté vektory - embeddingy
  - vstupem je 2D tenzor tvaru (batch_size, sequence_length)
  - výstupem je 3D tenzor tvaru (batch_size, sequence_length, embedding_dimension)
- jádrem vrstvy je **embedding matice** tvaru (počet tokenů ve slovníku $\times$ počet významových dimenzí)
  - každý řádek matice představuje embedding-vektor pro token s daným indexem
- Pro každý index tokenu najde embedding vrstva příslušný vektor ve slovníku a vrátí ho.

Embedding vrstva - jak ji vytvořit?

1. učí se spolu s modelem $\rightarrow$ embeddingy na míru
2. použiji již předučenou embedding vrstvu: pokud mám málo dat (Word2Vec - Google, GloVe - Stanford)

Příklad: Predikce sentimentu

- Unigramy + MLP : ET: 0.880
- Bigramy + MLP: ET: 0.892

Zkrácené sekvence (60 slov)

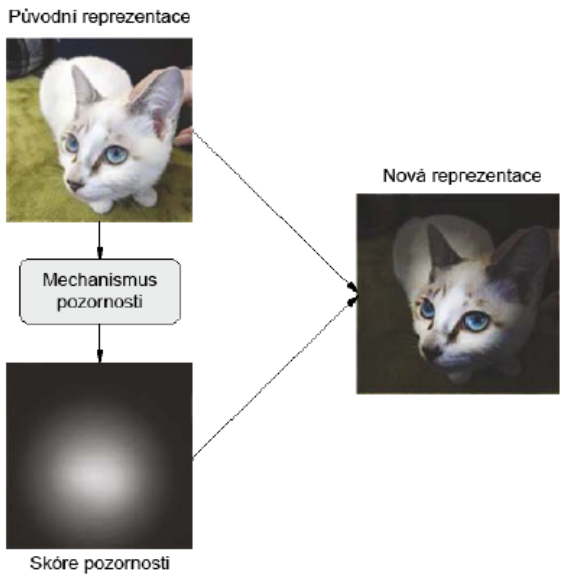
- Binání kódování + LSTM: 0.843
- Embeddingy + LSTM: 0.866
- Embeddingy + maskování + LSTM: 0.877
- Předučené embeddingy+ LSTM: 0.872

Transformery

- Ashish Vaswani et al.: Attention is all you need (2017), .
- nový revoluční prvek: mechanismus: **attention** (pozoronost), **self-attention**

Základní myšlenka: když čteme text, některým částem textu věnujeme větší pozornost než jiným

- co kdyby model dělal totéž, tj. vážil tokeny dle důležitosti?



Zdroj: F. Chollet: Deep learning v jazyku Python, obr 11.5

Transformery

Attention mechanismy umí mnohem více než jen vážit tokeny:

- díky nim si model může plně uvědomovat **kontext**

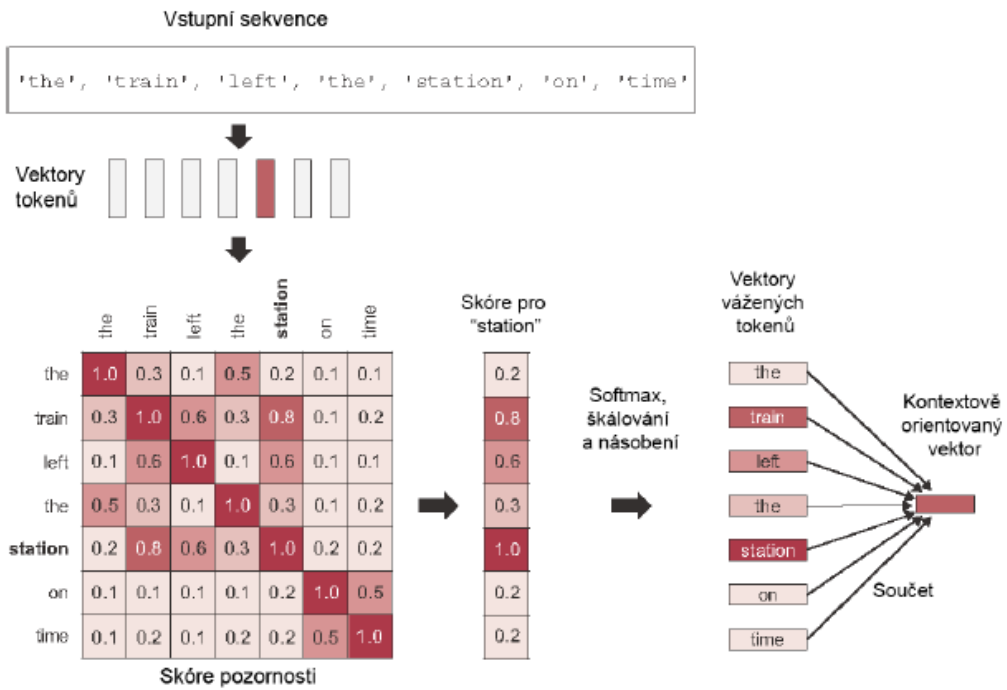
"*The train left the* **station** *on time..*"

- rozhlasová stanice?
- mezinárodní vesmírná stanice?
- stanice technické kontroly (STK)?
- ...

"**See** *you soon*" vs. "*I* **see** *what you mean*"

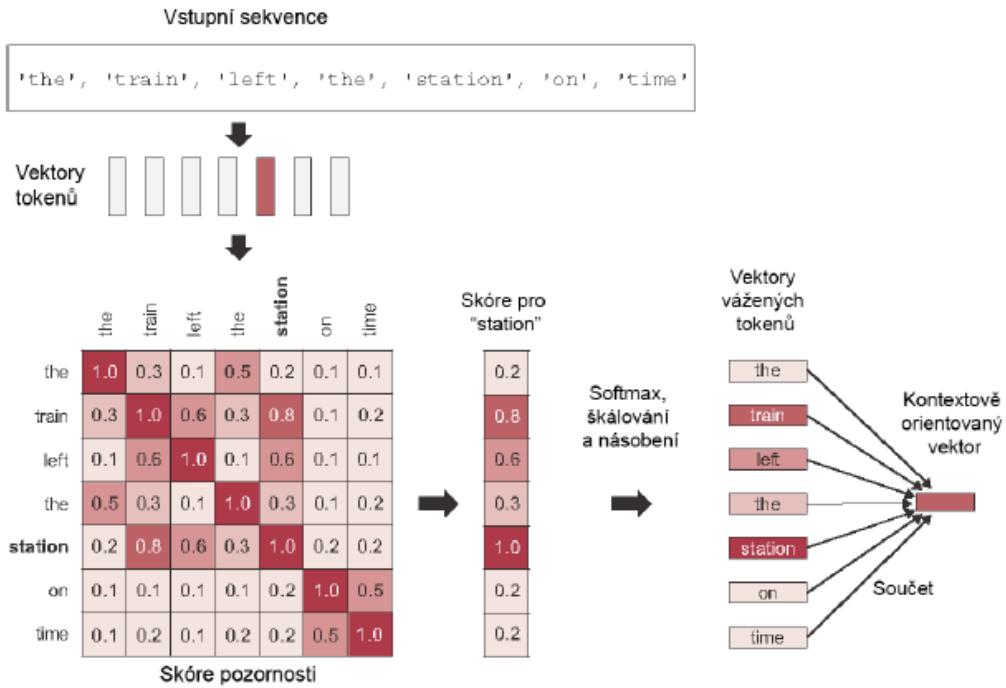
Attention (Pozornost)

1. pro každou dvojici tokenů ve větě spočítáme **attention score** (míru pozornosti): skalární součin jejich vektorů
2. pro každé slovo spočteme jeho **novou reprezentaci**: jako pozorností vážený součet všech tokenů ve větě



Zdroj: F. Chollet: Deep learning v jazyku Python, obr 11.5

Attention (Pozornost)



Zdroj: F. Chollet: Deep learning v jazyku Python, obr 11.5

- výstup = sum(vstup * attention_score(vstup, vstup))

Obecněji:

- výstup = sum(C * attention_score(A, B))

Attention (Pozornost)

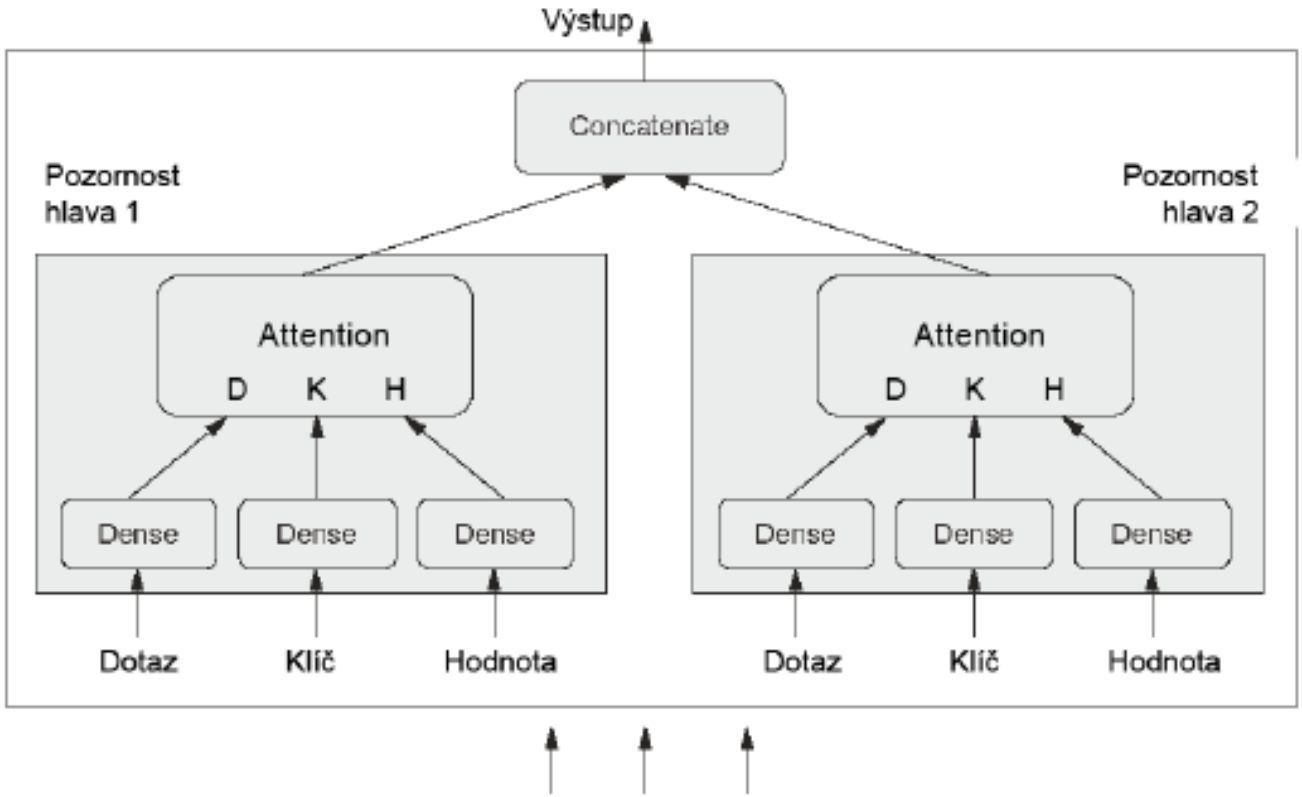
Terminologie: ze světa vyhledávačů

- výstup = sum(hodnota * attention_score(dotaz, klíč))
  - hodnota (value), dotaz (query), klíč (key)

Varianty

- **Klasifikace:** hodnota = dotaz = klíč = vstupní text
- **Strojový překlad:** hodnota = klíč = vstupní text, dotaz = přeložený text
- **Shrnutí textu:** hodnota = klíč = vstupní text, dotaz = otázka nebo požadavek na shrnutí
- **Generování textu:** hodnota = klíč = předchozí generované tokeny, dotaz = aktuální pozice
- **Recommender Systems:** hodnota = klíč = historická interakce uživatele, dotaz = aktuální doporučovaná položka

Multi-head attention

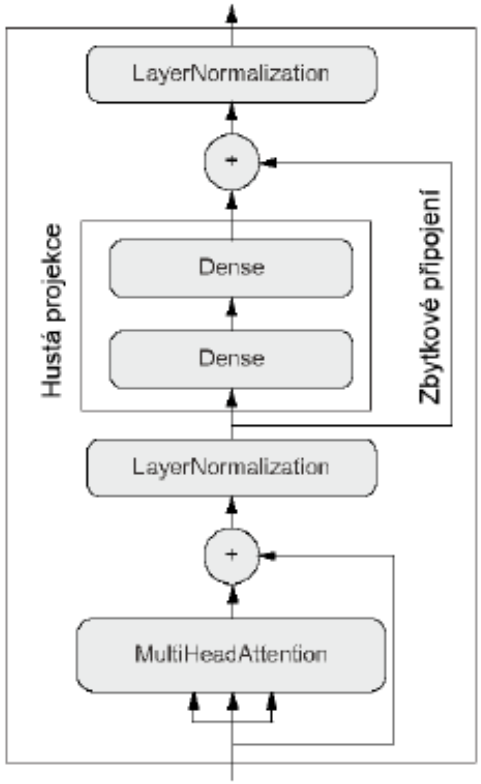


Zdroj: F. Chollet: Deep learning v jazyku Python, obr 11.8

Keras:

- speciální vrstva: self.attention = layers.MultiHeadAttention(num_heads=4, key_dim=256)

Transformer encoder

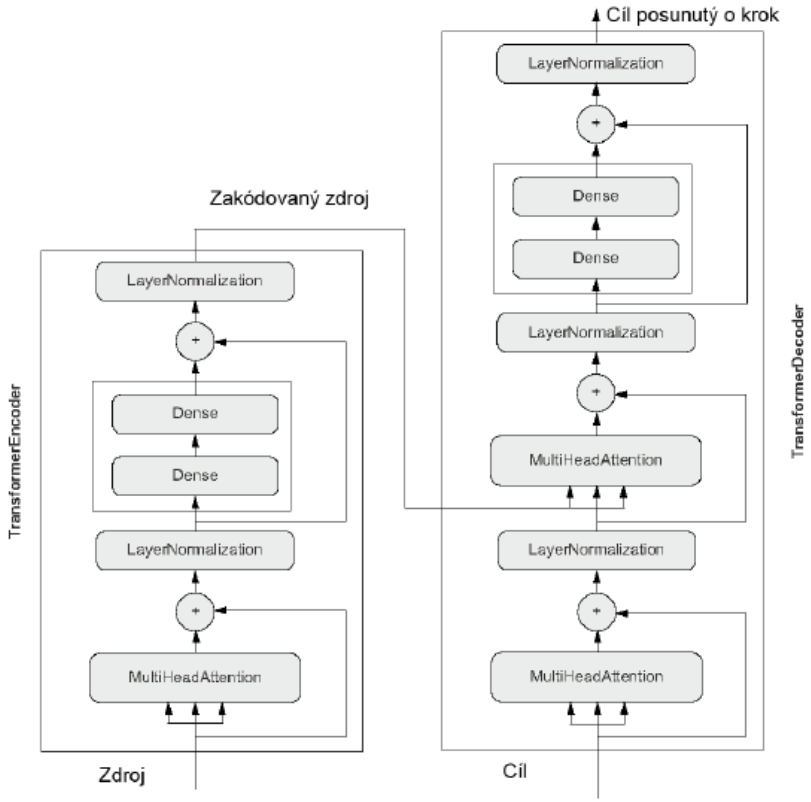


Zdroj: F. Chollet: Deep learning v jazyku Python, obr 11.9

Pro klasifikaci nám stačí enkodér

- multi-head attention + normalizace + MLP + residual connections

Transformer



Zdroj: F. Chollet: Deep learning v jazyku Python, obr 11.14

Pro automatické překlady nebo generování textu potřebujeme enkodér i dekodér (dohromady tvoří transformer)