Jazykové modely:
Moderné nástroje na riešenie komplexných úloh

BIBIÁNA LAJČINOVÁ
MICHAL PITOŇÁK
PATRIK VALÁBEK

Umelá inteligencia je aktuálne jednou z najrýchlejšie sa rozvíjajúcich oblastí informačných technológií. Obzvlášť disciplína spracovania prirodzeného jazyka (Natural Language Processing – NLP), ktorá je interdisciplinárnym odborom kombinujúcim počítačovú vedu, lingvistiku, matematiku, filozofiu a ďalšie oblasti. Medzi úlohy, ktorým sa táto oblasť venuje, patrí napríklad porozumenie prirodzenému jazyku, jeho generovanie, analýza sentimentu, vyhľadávanie a extrakcia informácií a podobne. O niektorých konkrétnych úlohách a ich riešeniach si môžete prečítať v článku Identifikácia entít pre extrakciu adries z transkribovaných rozhovorov s využitím syntetických dát [1] na webovej stránke Národného kompetenčného centra pre HPC. Dozviete sa o konkrétnej aplikácii metódy identifikácie entít (Named Entity Recognition – NER) na textové dáta obsahujúce adresy, pričom cieľom je extrahovať názov ulice, číslo domu, názov obce a poštové smerovacie číslo pomocou optimalizácie jazykového modelu SlovakBERT. V ďalšom článku Využitie veľkých jazykových modelov na efektívnu analýzu náboženských textov sa zaoberáme tvorbou vektorových databáz z náboženských textov pomocou verejne dostupných aj proprietárnych modelov, za účelom efektívneho vyhľadávania pasáží s relevantným obsahom (information retrieval).

Na vývoj obidvoch aplikácií boli použité výpočtové zdroje HPC systému Devana, z dôvodu vysokej výpočtovej náročnosti optimalizácie jazykových modelov. Tieto modely často obsahujú miliardy parametrov, a preto je prakticky nemožné ich optimalizovať na bežných osobných počítačoch. HPC systém Devana obsahuje 8 GPU výpočtových uzlov, pričom každý uzol má 4 GPU karty modelu nVidia A100 so 40 GB pamäte. Vďaka tomu je pri veľmi veľkých modeloch možné využiť distribuované trénovanie, či už implementovaním dátovej alebo modelovej paralelizácie. Distribuované trénovanie umožňuje efektívne rozloženie výpočtovej záťaže na viacero GPU a / alebo výpočtových uzlov a teda urýchlenie trénovania modelov.

Jazykové modely

Základným stavebným kameňom spracovania prirodzeného jazyka sú jazykové modely. Tieto modely "rozumejú" textu, poprípade dokážu aj generovať nový text, vďaka ich architektúre a skutočnosti, že boli trénované na veľkom množstve textových dát. Ich architektúra je založená na hlbokých neurónových sieťach, konkrétne na špecifickej topológii nazývanej *transformers*. Táto technológia bola prvýkrát predstavená v roku 2017 v článku *Attention is all you need* [2] a spôsobila prevrat v oblasti NLP. Umožňuje spracovanie sekvenčných dát, vďaka čomu je vhodná práve na prácu s jazykom / textom (ale aj na iné druhy sekvenčných dát). Na rozdiel od tradičných neurónových sietí používaných na sekvenčné dáta, ako sú rekurentné siete, obsahujú modely založené na architektúre transformers aj mechanizmus pozornosti. Tento

Umelá inteligencia je aktuálne jednou z najrýchlejšie sa rozvíjajúcich oblastí informačných technológií, obzvlášť disciplína spracovania prirodzeného jazyka.

mechanizmus umožňuje neurónovej sieti určiť, na ktoré časti vstupu sa má sústrediť a ako jednotlivé časti vstupu spolu súvisia, pre predikovanie najpravdepodobnejšieho výstupu.

Ďalším kľúčovým komponentom spracovania prirodzeného jazyka je tokenizácia. Táto operácia pretvára text na "tokeny", čo sú numerické reprezentácie slov, častí slov alebo jednotlivých znakov, v závislosti od použitého modelu. Tokeny zachytávajú kontext a význam slov a predstavujú vstup do neurónovej siete jazykových modelov.

Medzi ďalšie pojmy, ktoré sú frekventované v kontexte jazykových modelov, patria:

- **Embedding**
  vektorová reprezentácia textu (vety, frázy), ktorá zachytáva význam a kontext v numerickej podobe.

- **Fine-tuning**
  proces, pri ktorom sa predtrénovaný model optimalizuje na konkrétnu úlohu, ale už pomocou menšieho množstva dát a (väčšinou) zmeny menšieho počtu parametrov. Tento proces zlepšuje výkon modelu pri riešení špecifických úloh. Predtrénovaný model je už natrénovaný na enormnom množstve dát, k čomu je taktiež potrebný aj obrovský výpočtový výkon. Počas fine-tuning-u sa už iba doladia jeho parametre na špecifických dátach (v menšom objeme), ktoré počas predtrénovania neboli dostupné.

Medzi aktuálne "state-of-the-art" modely patrí napríklad BERT (Bidirectional Encoder Representations from Transformers), vyvinutý spoločnosťou Google. Ide o verejne dostupný (open-source) model obsahujúci len encoder časť, s pomerne jednoduchou architektúrou, ale vďaka svojej flexibilite je stále využívaný v rôznych úlohách. Ďalším populárnym modelom je GPT (Generative Pre-trained Transformer) od OpenAI. Je to multilingválny model, špecializovaný na generatívne úlohy a dokáže produkovať plynutý a kreatívny text. Avšak, tento model nie je verejne dostupný a je možné ho používať iba cez API (Application Programming Interface), ako aj ďalšie modely od internetových gigantov ako Facebook, Microsoft atď. Tieto komerčné modely síce majú špičkovú kvalitu, avšak nemusia byť vhodné pre každého, kvôli finančným nákladom, alebo na všetky úlohy, napríklad z dôvodu ochrany citlivých a / alebo proprietárnych dát.

Ako trénovať a používať jazykový model?

Pre užívateľov HPC systému Devana je na našich stránkach pripravený súbor návodov a ukážkových programov (v jazyku Python) na vytvorenie vlastných aplikácií využívajúcich veľké jazykové modely. Tieto návody sú zamerané na multilingválne a slovenské jazykové modely, ktoré užívatelia môžu optimalizovať (fine-tunovať) na základe vlastných dát ale aj používať ich voľne dostupné verzie bez fine-tuningu. Ku každému programu sú dostupné informácie o tom, ako ich spustiť na HPC systéme Devana pomocou dávkového systému Slurm, a ďalšie detaily. Medzi konkrétne aplikácie, ktoré sú v návodoch zahrnuté, patrí:

- **Analýza sentimentu pomocou modelu BERT na Twitter dátach**

  Viď. nasledujúca sekcia

- **Predikcia s použitím predtrénovaného LLM modelu na úlohu question-answering**

  Táto úloha sa zaoberá použitím verejne dostupných modelov Mistral 7B Instruct [3] a Aya [4] na úlohu question-answering. Na testovanie presnosti modelov používame verejne dostupný dataset MedQA [5], ktorý obsahuje otázky a odpovede z oblasti medicíny. Cieľom je ukázať, že veľké jazykové modely sa dajú použiť aj bez fine-tuningu.

- **Fine-tuning modelu Mistral 7B Instruct na úlohu question-answering**

  Táto úloha sa zaoberá fine-tuningom modelu Mistral 7B Instruct [3] na úlohu question-answering. Na trénovanie modelu používame verejne dostupný dataset MedQA [5], ktorý obsahuje otázky a odpovede z oblasti medicíny. Cieľom je vylepšiť schopnosť modelu správne odpovedať na otázky týkajúce sa medicínskeho obsahu, čím sa zvyšuje jeho presnosť a relevantnosť v danom kontexte.

- **Fine-tuning modelu DistilBERT na účely identifikácie entít (NER) s hľadaním optimálnych hodnôt hyperparametrov pomocou knižnice Optuna**

  V tejto úlohe optimalizujeme model DistilBERT [6] na úlohu identifikácie entít v texte. Používame verejne dostupný dataset CONLL2003 [7], ktorý obsahuje anotácie pre rôzne typy entít, kon-

krétne osoby (PER), miesta (LOC), organizácie (ORG) a rôzne iné (MISC). Optimálne hodnoty hyperparametrov trénovania hľadáme pomocou knižnice Optuna, ktorá nám umožňuje systematicky hľadať najlepšie hodnoty rýchlosti učenia, veľkosti dávky, počtu epoch atp.

- **Vytvorenie vektorovej databázy pomocou open-source embedding modelov**

  V tejto úlohe sa zameriavame na vytvorenie vektorovej databázy pomocou open-source embedding modelov. Používame model BGE M3 [8] na generovanie vektorových reprezentácií textov, ktoré nám umožňujú efektívne vyhľadávanie v texte. Texty, z ktorých generujeme embeddingy, pochádzajú z verejne dostupného datasetu 20newsgroups [9], ktorý obsahuje správy a články z novín. Tieto embeddingy slúžia na vytvorenie vektorovej databázy, ktorá uľahčuje rýchle vyhľadávanie a analýzu dokumentov na základe ich obsahovej podobnosti.

Prvú z uvedených aplikácií analyzujeme detailnejšie.

Analýza sentimentu pomocou modelu BERT na dátach zo sociálnej siete Twitter

V súčasnej dobe je analýza sentimentu jednou z častých úloh pre algoritmy spracovania prirodzeného jazyka. Táto úloha sa zameriava na identifikáciu a hodnotenie sentimentu (t.j. "nálady", väčšinou sa rozlišuje pozitívny, negatívny a neutrálny) v textoch, čo je užitočné napr. pre analýzu názorov verejnosti, zákazníkov, nálad a trendov v rôznych spoločenských a biznisových doménach. Modely ako BERT sa ukázali ako účinné nástroje práve na tento účel. Dáta používané na ich trénovanie sú častokrát získavané zo sociálnych sietí ako Twitter a Facebook, pretože zrkadlia širokú škálu názorov a emocionálnych prejavov používateľov. V tabuľke 1 sú uvedené príklady "tweetov" s rôznymi sentimentami:

Tweet	Sentiment
Skvelý deň! Slnečno a všetci sú usmiati.	pozitívny
Nenávidím, keď musím čakať v dlhých radoch.	negatívny
Dnes je utorok.	neutrálny

Keďže sa jedná o úlohu, kde každému pozorovaniu priraďujeme jeden sentiment, hovoríme o klasifikácii (do troch tried). Preto budeme používať model BERT s výstupnou vrstvou s troma neurónmi.

Ďalej si ukážeme, ako vytvoriť skript na fine-tunovanie modelu BERT na verejne dostupných dátach z Twitter-u.

Príprava dát, výber modelu a trénovanie

Pred samotným trénovaním modelu je kľúčové správne pripraviť dáta. V našom prípade používame dáta, ktoré sú vo formáte *csv*, tvorené tweetmi v anglickom jazyku, pričom každý z nich je označený jedným z troch možných sentimentov – pozitívny, neutrálny a negatívny.

V tomto prípade predspracovanie dát spočíva len v odstraňovaní riadkov s chýbajúcimi hodnotami.

```
df = pd.read_csv('Twitter_Data.csv')
df = df.dropna(subset=['category', 'clean_text'])  # Drop rows with NaN values
```

Pretože ide o dáta v anglickom jazyku, vybrali sme model *BERT base model* (*uncased*) [10], čiže model BERT v zmenšenej verzii (obsahuje menej parametrov oproti verzii *large*), ktorý nerozlišuje medzi veľkými a malými písmenami. K tomuto modelu patrí aj príslušný tokenizer.

Predtrénovaný model načítame z Hugging Face hub-u[1]. Keď sa používa metóda *from_pretrained*, načíta sa predtrénovaný model (existujúci model s natrénovanými váhami). To znamená, že nasleduje iba fine-tuning, nie trénovanie od základu.

```
MODEL = "bert-base-uncased"
tokenizer = BertTokenizer.from_pretrained(MODEL, do_lower_case=True)
# Load pretrained model with 3 labels: positive, negative, neutral
model = BertForSequenceClassification.from_pretrained(MODEL, num_labels=3)
```

Po rozdelení dát na trénovaciu, validačnú a testovaciu množinu všetky tri časti tokenizujeme.

[1] Hugging Face je platforma pre NLP a strojové učenie. Poskytuje široký výber pretrénovaných modelov a nástrojov, ktoré sú k dispozícii na ich webovej stránke https://huggingface.co.

```
# Split data into train, validation, test sets
train_data, test_data = train_test_split(df.to_
dict(orient='records'), test_size=0.2, random_state=42)
train_data, val_data = train_test_split(train_data, test_size=0.1,
random_state=42)

# Create DatasetDict for train, validation, test datasets
dataset_dict = DatasetDict({
    'train': Dataset.from_dict({'text': [item['clean_text'] for
item in train_data], 'label': [item['category_1'] for item in
train_data]}),
    'validation': Dataset.from_dict({'text': [item['clean_text']
for item in val_data], 'label': [item['category_1'] for item in
val_data]}),
    'test': Dataset.from_dict({'text': [item['clean_text'] for
item in test_data], 'label': [item['category_1'] for item in test_
data]})
})

tokenized_dataset = dataset_dict.map(tokenize_function,
batched=True)
```

Následne definujeme trénovacie argumenty, vrátane počtu epoch, veľkosti dávky (batch size), stratégie vyhodnocovania, atď., a môžeme pristúpiť k fine-tunovaniu modelu pomocou metódy *Trainer*.

```
training_args = TrainingArguments(
    per_device_train_batch_size=64,
    per_device_eval_batch_size=64,
    output_dir="bert_twitter",
    num_train_epochs=5,
    weight_decay=0.1,
    evaluation_strategy="epoch",
    eval_steps=300,
    save_strategy="epoch",
    save_steps=300,
    load_best_model_at_end=True,
    eval_accumulation_steps=300,
    logging_steps=300
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset["train"],
    eval_dataset=tokenized_dataset["validation"],
    tokenizer=tokenizer,
    data_collator=data_collator, # responsible for processing data
into expected format - batches
    compute_metrics=compute_metrics # function calculating metrics
to evaluate model's accuracy

trainer.train()
```

Fine-tunovaný model je možné uložiť a používať ho na predikciu na nových dátach.

```
model.save_pretrained("bert_sentiment_model")
```

Predikciu na nových dátach môžeme urobiť pomocou metódy *pipeline* z knižnice *transformers*. Vo výstupe dostaneme okrem predikovanej triedy (sentimentu) aj pravdepodobnosť, ktorá vyjadruje mieru istoty modelu pri priraďovaní textu do danej triedy.

```
model = BertForSequenceClassification.from_pretrained("bert_sentiment_model")
sentiment_pipeline = pipeline("sentiment-analysis", model=model,
tokenizer=tokenizer, device=device) # inference initialization

text = "today is a beautiful day" # input text
result = sentiment_pipeline(text)
print(result)
```

Spustenie trénovania pomocou manažéra úloh Slurm a grafického rozhrania prostredníctvom OpenOnDemand

Trénovanie modelu je ideálne realizovať pomocou manažéra úloh Slurm, ktorý poskytuje prístup na výpočtové uzly HPC systému Devana (na rozdiel od priameho spustenia na jednom z prihlasovacích uzlov). Alternatívne je možné využiť grafické rozhranie prostredníctvom služby OpenOnDemand, vhodné napr. pre interaktívnu prácu s kódom v Jupyter Notebooku. Pri využití tejto služby je možné alokovať požadovaný výpočtový výkon na určitý čas a následne spustiť trénovanie modelu a / alebo analyzovať výsledky priamo v prostredí Jupyter Notebooku.

Nasledujúci shell skript slúži na spustenie fine-tunovania BERT modelu na analýzu sentimentu na HPC systéme pomocou plánovača úloh Slurm:

```
#!/bin/bash
#SBATCH --account=<your_project_number> # Kod projektu
#SBATCH -o result.txt                   # Subor pre vystup
#SBATCH -e error.txt                    # Subor pre zachytavanie chyb
#SBATCH --gres=gpu:1                    # Alokacia 1 GPU
#SBATCH --nodes=1                       # Poziadavka na 1 uzol
#SBATCH --partition=gpu                 # Vyber GPU particie

module load singularity                 # Nacitanie modulu singularity
```

```
singularity exec --nv /storage-data/singularity_containers/pt-
2.3_llm.sif python3 sentiment_analysis_bert_train.py --batch_
size 64
```

Posledný riadok spúšťa Python skript *sentiment_analysis_bert_train.py* pomocou singularity kontajnera. Parameter *--nv* umožňuje použitie GPU akcelerátorov v rámci kontajnera. Kontajner *pt_2.3_llm.sif* obsahuje všetky potrebné knižnice vrátane PyTorch, Transformers, Pandas, Numpy a ďalších.

Distribuované trénovanie

Pri trénovaní veľkých jazykových modelov s miliardami parametrov, je paralelizácia kľúčová pre efektívne využitie dostupných výpočtových zdrojov a zníženie celkovej doby výpočtu. Existujú dva hlavné prístupy k paralelizácii: dátová paralelizácia a modelová paralelizácia.

Dátová paralelizácia spočíva v rozdelení trénovacích dát na menšie časti, ktoré sa spracúvajú súčasne, na viacerých GPU akcelerátoroch. Modelová paralelizácia spočíva v rozdelení samotného modelu na menšie časti, ktoré sa spracúvajú súčasne na viacerých GPU akcelerátoroch alebo výpočtových uzloch. Tento prístup je užitočný, ba až nevyhnutný, keď je LLM model príliš veľký na to, aby sa zmestil do pamäte jedného GPU akcelerátora.

Modely z knižnice Hugging Face umožňujú automatickú modelovú paralelizáciu. Pri paralelizácii modelu je potrebné špecifikovať zariadenia, na ktoré chcete modely načítať, alebo je možné použiť hodnotu "auto" v argumente *device_map*, a tým sa model automaticky rozdelí medzi dostupné zdroje.

```
model = BertForSequenceClassification.from_pretrained("bert_
sentiment_model", device_map = "auto")
```

Ďalšie možnosti distribuovaného trénovania sú podrobne popísané v dokumentácii Hugging Face. Táto stránka poskytuje prehľad rôznych techník paralelizácie, ktoré môžu byť využité na efektívne rozdelenie trénovania na viacero GPU akcelerátorov.

Všetky programy a potrebné pomocné skripty k ďalším príkladom, vrátane inštrukcií na spustenie, sú dostupné na našom repozitári github.com/NSCC-Slovakia/NCC_HPC-FineTuning-Examples. Alternatívne, ak by ste potrebovali ďalšiu pomoc alebo konzultácie ohľadom trénovania modelov na HPC systéme Devana, srdečne Vás pozývame na osobné alebo online stretnutie v NSCC.

LITERATÚRA

[1] Bibiána Lajčinová, Patrik Valábek, and Michal Spišiak. Named entity recognition for address extraction in speech-to-text transcriptions using synthetic data, 2024.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[3] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[4] Ahmet ¨Ust¨un, Viraat Aryabumi, Zheng-Xin Yong, Wei-Yin Ko, Daniel D'souza, Gbemileke Onilude, Neel Bhandari, Shivalika Singh, Hui-Lee Ooi, Amr Kayid, Freddie Vargus, Phil Blunsom, Shayne Longpre, Niklas Muennighoff, Marzieh Fadaee, Julia Kreutzer, and Sara Hooker. Aya model: An instruction finetuned open-access multilingual language model. *arXiv preprint arXiv:2402.07827*, 2024.

[5] Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14):6421, 2021.

[6] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108, 2019.

[7] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147, 2003.

[8] Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation, 2024.

[9] Ken Lang. Newsweeder: Learning to filter netnews. In *Armand Prieditis and Stuart Russell, editors, Machine Learning Proceedings 1995*, pages 331–339. Morgan Kaufmann, San Francisco(CA), 1995.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

Pri trénovaní veľkých jazykových modelov s miliardami parametrov, je paralelizácia kľúčová pre efektívne využitie dostupných výpočtových zdrojov a zníženie celkovej doby výpočtu.