

Generowanie tytułów publikacji naukowych

Projekt z przedmiotu: Przetwarzanie Języka
Naturalnego

Maciej Płachta

Szymon Rozmus

Łukasz Ryczek

Spis treści

Abstrakt.....	2
Wstęp.....	2
Cel.....	2
Zakres.....	2
Metodyka.....	2
Część teoretyczna.....	3
Model GPT Neo.....	3
Fine-tuning.....	3
Tokenizacja.....	3
Część praktyczna.....	3
Pobranie danych.....	3
Załadowanie modelu i tokenizera.....	4
Tokenizacja.....	4
Fine-tuning modelu.....	5
Generowanie przykładów na podstawie podanego abstraktu.....	5
Podsumowanie.....	6
Bibliografia.....	7

Abstrakt

Projekt ma za zadanie wykorzystanie technik przetwarzania języka naturalnego w celu wygenerowania tytułów naukowych na podstawie podanych abstraktów. W ramach projektu dokonano fine-tuningu modelu GPT Neo 125M zaprojektowanego przez EleutherAI jako replikacja architektury GPT-3. W tym celu wykorzystane zostały dane artykułów naukowych dotyczących sztucznej inteligencji z archiwum arXiv[1], język Python oraz Google Colab. Wzorem dla tego projektu był inny stworzony już projekt znajdujący się na githubie[2], którego twórcą jest Chandan Singh.

Wstęp

Cel

Celem projektu jest stworzenie skutecznego narzędzia pozwalającego na generowanie tytułów artykułów naukowych na podstawie podanego abstraktu opartego na fine-tuningu modelu GPT Neo 125M.

Zakres

Projekt obejmuje pobranie danych oraz modelu i wytrenowanie sieci neuronowej. Na wejściu sieć przyjmuje abstrakt, a na wyjściu ma podać żadaną ilość propozycji tytułów odpowiednich do artykułu zaczynającego się od tego abstraktu. Docelowo wszystkie propozycje powinny móc być użyte, jednak dopuszcza się niezbędność ludzkiej selekcji oraz redakcji.

Metodyka

- pobranie danych z archiwum arXiv,
- przygotowanie danych,
- dobór hiperparametrów,
- finetuning modelu,
- testowanie

Część teoretyczna

Model GPT Neo[3]

GPT Neo to rodzina modeli językowych o otwartym kodzie Źródłowym stworzonym przez EluetherAI. Modele są inspirowane architekturą GPT-3 opracowaną przez OpenAI. Modele w rodzinie GPT Neo różnią się skalą, gdzie liczba parametrów jest mniejsza niż w GPT-3, lecz nadal są one w stanie generować teksty wysokiej jakości.

Fine-tuning[4]

Fine-tuning jest techniką stosowaną w uczeniu maszynowym polegającą na dostosowaniu wcześniej przeszkolonego modelu do specyficznych zastosowań lub zbiorów danych, które mogą różnić się od danych użytych podczas pierwotnego treningu. Parametry modelu są aktualizowane, by dostosować się do nowych wzorców.

Tokenizacja

Tokenizacja jest powszechnym krokiem w przetwarzaniu języka naturalnego. Ponieważ modele językowe działają na małych częściach tekstu, zwanych tokenami, tekst musi zostać do takiej właśnie postaci przygotowany. Pod czas tokenizacji tekst dzielony jest na zdania, wyrazy i znaki interpunkcyjne. Takie podejście pozwala na bardziej efektywne uczenie maszynowe - pozwala zoptymalizować wykorzystanie zasobów systemu (w tym przypadku głównie pamięci).

Część praktyczna

Pobranie danych

Dane do dostrajania modelu pobrane zostały z arXiv poprzez bibliotekę arxivscraper. Dane zostały przefiltrowane pod względem kategorii – zostały wybrane tylko te dotyczące sztucznej inteligencji.

```
import arxivscraper as ax
import pandas as pd
import json
scraper = ax.Scraper(category='cs', date_from='2020-04-01',
                    date_until='2020-06-01', t=10,
                    filters={
                        'categories':['cs.AI']
                    })

output = scraper.scrape()

df = pd.DataFrame(output, columns=['abstract','title'])
```

Załadowanie modelu i tokenizera

```
checkpoint = "EleutherAI/gpt-neo-125M"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
tokenizer.pad_token = tokenizer.eos_token
model = AutoModelForCausalLM.from_pretrained(checkpoint)
```

Z pomocą biblioteki Hugging Face Transformers zostaje załadowany tokenizer oraz model GPT Neo 125M .

Tokenizacja

W naszym projekcie do tokenizacji został wykorzystany AutoTokenizer z transformers.

```
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
tokenizer.pad_token = tokenizer.eos_token

if os.path.exists('/content/dset_tokenized.pkl'):
    dset_tokenized = pickle.load(open('dset_tokenized.pkl', 'rb'))
else:
    df['text'] = df['abstract'] + '\n\n' + df['title']
    df = df.drop(columns=['abstract', 'title'])
    dset = datasets.Dataset.from_pandas(df)
    print(dset)
    # set up tokenized data
    print('tokenizing data...')

    def tokenize_function(ex):
        return tokenizer(ex["text"], padding='max_length', truncation=True)
    dset_tokenized = dset.map(tokenize_function, batched=True)
    dset_tokenized = dset_tokenized.add_column(
        'labels', dset_tokenized['input_ids'])
    pickle.dump(dset_tokenized, open('dset_tokenized.pkl', 'wb'))
print('training on', len(dset_tokenized), 'examples')
print('ex', repr(dset_tokenized[0]['text']))
print('\t(both prompt and text-gen should end with something specific, like newline)')
data_collator = DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm=False)
```

Dane są dzielone na tytuły oraz abstrakty, które umieszczane są w datasecie pandas. Następnie na tym datasecie wykonywana jest tokenizacja za pomocą AutoTokenizer'a. Na koniec wyświetlane jest krótkie podsumowanie przetworzonych danych.

Fine-tuning modelu

```
# set up trainer
training_args = TrainingArguments(
    output_dir='title-' + checkpoint.replace('/', '__'),
    # 1e-05 was used for 2.7B model
    # warmup_steps=100,
    learning_rate=5e-06,
    weight_decay=0.01, # 0.01 was used for 20B
    per_device_train_batch_size=2,
    gradient_accumulation_steps=5,
    num_train_epochs=4,
    save_strategy='epoch',
)
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=dset_tokenized,
    data_collator=data_collator
)

# launch training
trainer.train()
```

Klasy 'TrainingArguments' i 'Trainer' z biblioteki Transformers są wykorzystywane do konfiguracji i uruchomienia procesu dostrajania modelu.

Określone zostały :

- katalog wyjściowy,
- wartość szybkości uczenia,
- wartość spadku wagi,
- kroki treningowe przed aktualizacją wag,
- ilość epok treningowych

Do Trainera przekazane zostały: model, argumenty treningowe, ztokenizowany zbiór danych, obiekt do kolekcji danych łączący dane w batche. Metoda train() rozpoczyna proces treningowy.

Generowanie przykładów na podstawie podanego abstraktu

```
def generate_samples(model, tokenizer, prompt, num_return_sequences=1):
    # decode some examples
    eos_token_id = tokenizer('\n')['input_ids'][0]
    inputs = tokenizer(prompt, return_tensors='pt').to(model.device)
```

```

outputs = model.generate(
    **inputs,
    do_sample=True,
    eos_token_id=eos_token_id,
    max_new_tokens=30,
    num_return_sequences=num_return_sequences,
    #temperature=0.8,
)
return tokenizer.batch_decode(outputs, skip_special_tokens=True)

checkpoint = "EleutherAI/gpt-neo-125M"
checkpoint_dir = "/content/title-EleutherAI__gpt-neo-125M/checkpoint-258"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
tokenizer.pad_token = tokenizer.eos_token

# load model
mod = AutoModelForCausalLM.from_pretrained(
    # checkpoint,
    checkpoint_dir, local_files_only=True
)

abstract = 'Parameter sharing, as an important technique in multi-agent systems, can effectively solve the scalability issue in largescale agent problems. However, the effectiveness of parameter sharing largely depends on the environment setting. When agents have different identities or tasks, naive parameter sharing makes it difficult to generate sufficiently differentiated strategies for agents. Inspired by research pertaining to the brain in biology, we propose a novel parameter sharing method. It maps each type of agent to different regions within a shared network based on their identity, resulting in distinct subnetworks. Therefore, our method can increase the diversity of strategies among different agents without introducing additional training parameters. Through experiments conducted in multiple environments, our method has shown better performance than other parameter sharing methods.\n\n'
samples =generate_samples(
    mod,
    tokenizer,
    prompt=abstract,
    num_return_sequences=20,
)
save_str = '\n\n'.join([repr(x[len(abstract):].strip()) for x in samples])
open(f'./testgeneration.txt', 'w').write(save_str)

```

Podsumowanie

Na podstawie podanego abstraktu model stworzył bardziej lub mniej zadowalające tytuły.

Najgorsze:

- 'p.\xa0experiment'
- 'background description'

Ciekawe:

- 'useragent and parameter sharing for the agent with human biases (a) in two and five agents (b) in three agents, and (c) in five agents.'
- 'para-method-sharing-a;'

Filozoficzne:

- 'user, task, environment'
- '10 10-2'
- 'Introducing different identities'

Najlepsze:

- '- Parameter sharing for multi-agent systems'
- 'evaluation of parameter sharing for agent problems.'
- 'algorithm for parameter sharing in largescale agents'

Zadowalające tytuły były w znacznej mniejszości. Większość otrzymanych wyników nie nadawała się do użycia ani redagowania przez człowieka.

Wynika to najprawdopodobniej z wielkości użytego modelu. Z powodu braku sprzętu niezbędnego do trenowania zaawansowanego modelu i podobnych ograniczeń wynikających z Google Colab użyty został model mniejszy niż pierwotnie zaplanowany.

Osiągnięto jednak pewien sukces. Wśród otrzymanych wyników znajdują się tytuły które po ludzkiej selekcji mogłyby zostać użyte.

Bibliografia

- [1] <https://arxiv.org>
- [2] <https://github.com/csinva/gpt-paper-title-generator>
- [3] https://huggingface.co/docs/transformers/model_doc/gpt_neo
- [4] [https://en.wikipedia.org/wiki/Fine-tuning_\(deep_learning\)](https://en.wikipedia.org/wiki/Fine-tuning_(deep_learning))