

Politechnika Krakowska

Generator tytułów

Projekt z przedmiotu
Przetwarzanie Języka Naturalnego

Wykonali: Łukasz Białas, Kim Aloszyn

Prowadzący: dr. inż Radosław Kycia

W dobie dynamicznego rozwoju technologii informacyjnych i sztucznej inteligencji, generowanie tekstu przez modele językowe odgrywa kluczową rolę w wielu dziedzinach nauki i przemysłu. Niniejsza praca przedstawia dwa innowacyjne podejścia do generowania tytułów naukowych artykułów: jedno za pomocą modelu GPT-2, a drugie przy użyciu klasycznych metod przetwarzania języka naturalnego (NLP).

Wstęp

Cel pracy

Celem niniejszej pracy jest opracowanie dwóch podejść do generowania tytułów naukowych artykułów. Pierwsze podejście wykorzystuje model językowy GPT-2, który jest dostosowywany (fine-tuned) na podstawie tytułów z serwisu arXiv. Drugie podejście opiera się na klasycznych technikach przetwarzania języka naturalnego (NLP) i wykorzystuje bigramy do tworzenia nowych tytułów. Celem jest ocena jakości generowanych tytułów oraz zbadanie, które podejście lepiej wspiera proces tworzenia naukowych publikacji.

Zakres

Zakres projektu obejmuje:

1. Zbieranie danych: Pobranie tytułów artykułów naukowych z serwisu arXiv.
2. Fine-tuning modelu GPT-2: Dostosowanie modelu na podstawie zebranych tytułów oraz eksperymenty z parametrami generowania tekstu (temperatura, top-k, top-p).
3. Klasyczne techniki NLP: Zastosowanie metod takich jak tokenizacja i lematyzacja do przetwarzania tytułów.
4. Generowanie tytułów przy użyciu bigramów: Konstrukcja bigramów z przetworzonych tytułów oraz generowanie nowych tytułów na ich podstawie.
5. Ocena i porównanie: Analiza i porównanie jakości tytułów wygenerowanych przez oba podejścia.

Metodyka

W projekcie zastosowano następujące metody i narzędzia:

1. **Zbieranie danych:**
 - Wykorzystanie API serwisu arXiv do pobierania tytułów artykułów naukowych.
2. **Fine-tuning modelu GPT-2:**
 - Model GPT-2 z biblioteki Hugging Face Transformers.
 - Dostosowanie modelu przy użyciu tytułów z arXiv.
 - Eksperymentowanie z różnymi parametrami generowania tekstu.
3. **Klasyczne techniki NLP:**
 - Tokenizacja: Rozbijanie tytułów na poszczególne słowa.
 - Lematyzacja: Sprowadzanie słów do ich podstawowej formy gramatycznej.
4. **Generowanie tytułów przy użyciu bigramów:**
 - Konstrukcja bigramów z przetworzonych tytułów.
 - Generowanie nowych tytułów na podstawie zbudowanych bigramów.
5. **Ocena i porównanie:**
 - Ocena jakości wygenerowanych tytułów
 - Porównanie wyników obu podejść i wyciągnięcie wniosków na temat ich skuteczności.

Teoria

Sztuczna Inteligencja i Przetwarzanie Języka Naturalnego

Sztuczna inteligencja (SI) to dziedzina informatyki zajmująca się tworzeniem systemów zdolnych do wykonywania zadań, które normalnie wymagają ludzkiej inteligencji. Przetwarzanie języka naturalnego (NLP - Natural Language Processing) jest jednym z obszarów SI, koncentrującym się na interakcji między komputerami a ludzkim językiem naturalnym.

Modele Językowe

Modele językowe są rdzeniem NLP i są wykorzystywane do przewidywania następnych słów w sekwencji, generowania tekstu, tłumaczenia języka i wielu

innych zadań. Model językowy GPT-2 (Generative Pre-trained Transformer 2) jest jednym z modeli, który został opracowany przez OpenAI.

GPT-2: Generative Pre-trained Transformer 2

GPT-2 jest dużym modelem językowym opartym na architekturze transformera, który został wstępnie wytrenowany na ogromnych zbiorach danych tekstowych z internetu. Model ten składa się z wielu warstw przekształceń i jest zdolny do generowania tekstu, który jest trudny do odróżnienia od tekstu napisanego przez człowieka. Kilka kluczowych pojęć związanych z GPT-2 to:

- **Architektura Transformera:** Model GPT-2 wykorzystuje architekturę transformera, która pozwala na równoległe przetwarzanie sekwencji danych, co znacznie przyspiesza proces trenowania i generowania tekstu.
- **Self-Attention:** Mechanizm self-attention umożliwia modelowi koncentrację na różnych częściach sekwencji wejściowej przy generowaniu każdego tokena, co pozwala na bardziej kontekstowe rozumienie tekstu.
- **Tokenizacja:** Proces tokenizacji polega na dzieleniu tekstu na mniejsze jednostki, zwane tokenami, które mogą być pojedynczymi słowami, częściami słów lub znakami. GPT-2 używa tokenizera, który zamienia tekst na liczby, które mogą być przetwarzane przez model.

Fine-Tuning Modelu GPT-2

Fine-tuning jest procesem dalszego trenowania wstępnie wytrenowanego modelu na specyficznym zbiorze danych, aby dostosować go do konkretnego zadania. W kontekście naszej pracy, fine-tuning modelu GPT-2 na zbiorze tytułów z serwisu arXiv pozwala modelowi na lepsze zrozumienie struktury i stylu tytułów naukowych artykułów. Kroki związane z fine-tuningiem obejmują:

- **Zbieranie Danych:** Pobieranie tytułów artykułów z serwisu arXiv.
- **Przygotowanie Danych:** Tokenizacja tytułów i przekształcenie ich w format zrozumiały dla modelu GPT-2.
- **Trenowanie Modelu:** Dalsze trenowanie modelu GPT-2 na przygotowanych tytułach, dostosowywanie parametrów takich jak temperatura, top-k, top-p.

Generowanie Tytułów za Pomocą GPT

Proces generowania tytułów za pomocą modelu GPT-2 lub na wersjach po fine-tuningu polega na dostarczeniu modelowi sekwencji początkowych tytułów, które służą jako kontekst, a następnie wygenerowaniu nowego tytułu na podstawie tego kontekstu. Parametry, które wpływają na jakość generowanego tekstu to:

- **Temperatura:** Kontroluje losowość generowania tekstu; niższe wartości prowadzą do bardziej deterministycznych wyników, podczas gdy wyższe wartości wprowadzają większą różnorodność.
- **Top-k Sampling:** Ogranicza wybór kolejnego tokena do k najbardziej prawdopodobnych opcji, co zapobiega generowaniu mniej prawdopodobnych tokenów.
- **Top-p (Nucleus) Sampling:** Ogranicza wybór tokenów do tych, których skumulowana prawdopodobieństwo wynosi co najmniej p, co balansuje między różnorodnością a spójnością generowanego tekstu.

Część praktyczna

W tej sekcji spróbujemy wygenerować tytuły korzystając z modeli pgt oryginalnego i po fine-tuningu oraz opierając się na klasycznych technikach NLP.

Używamy API arXiv do pobrania tytułów artykułów autorstwa Yoshua Bengio z kategorii uczenia maszynowego (cs.LG). Wyniki są zapisywane do listy **titles** oraz pliku .csv.

```
import arxiv

from transformers import GPT2LMHeadModel, GPT2Tokenizer

import pandas as pd

import re

client = arxiv.Client()

search = arxiv.Search(

    query = 'au:"Yoshua Bengio" AND cat:cs.LG',
```

```

max_results = 1000,

sort_by = arxiv.SortCriterion.SubmittedDate
)

papers = list(client.results(search))

papers[0]

titles = [paper.title for paper in papers]

titles_df = pd.DataFrame(titles, columns=["Title"])

titles_df.to_csv("yoshua_bengio_titles.csv", index=False)

```

W tym kroku pobraliśmy tytuły autora Yoshua Bengio z dziedziny uczenia maszynowego i zapisaliśmy do pliku .csv.

```

yoshua_bengio_titles.csv
1  Title
2  Learning diverse attacks on large language models for robust red-teaming and safety tuning
3  Attention as an RNN
4  Metacognitive Capabilities of LLMs: An Exploration in Mathematical Problem Solving
5  Generative Active Learning for the Search of Small-molecule Protein Binders
6  Towards DNA-Encoded Library Generation with GFlowNets
7  Foundational Challenges in Assuring Alignment and Safety of Large Language Models
8  Language Models Can Reduce Asymmetry in Information Markets
9  Ant Colony Sampling with GFlowNets for Combinatorial Optimization
10 Discrete Probabilistic Inference as Control in Multi-path Environments
11 Iterated Denoising Energy Matching for Sampling from Boltzmann Densities
12 Improved off-policy training of diffusion samplers
13 Efficient Causal Graph Discovery Using Large Language Models
14 A Hitchhiker's Guide to Geometric GNNs for 3D Atomic Systems
15 Improving Gradient-guided Nested Sampling for Posterior Inference
16 Unlearning via Sparse Representations
17 Mitigating Biases with Diverse Ensembles and Diffusion Models

```

Generowanie tytułów za pomocą GPT

Łaadowaliśmy model GPT-2 oraz jego tokenizer.

```

model = GPT2LMHeadModel.from_pretrained('gpt2')

tokenizer = GPT2Tokenizer.from_pretrained('gpt2')

```

Utworzyliśmy metodę generującą nowe tytuły na podstawie dostarczonych tytułów kontekstowych. Używa ona różnych parametrów, takich jak `top_k`, `top_p` i `temperature`, aby kontrolować proces generowania tekstu. Dekoduje wygenerowany tekst.

```
def generate_titles(titles, max_new_tokens=15, num_titles=5,
num_return_sequences=3):

    context_titles = ' '.join(titles[:num_titles])

    prompt = f"{context_titles}\nGenerated Title: "

    inputs = tokenizer.encode(prompt, return_tensors='pt')

    outputs = model.generate(

        inputs,

        max_new_tokens=max_new_tokens,

        num_return_sequences=num_return_sequences,

        pad_token_id=tokenizer.eos_token_id,

        do_sample=True,

        top_k=10,

        top_p=0.95, # Nucleus sampling for better results

        temperature=0.7 # Controls the randomness

    )

    # Decodes the generated output and clean up the titles
    generated_titles = []

    for output in outputs:

        decoded_output = tokenizer.decode(output,
skip_special_tokens=True)

        new_title = decoded_output.split("Generated
Title:")[-1].strip().split('\n')[0] # Handle multiple lines

        cleaned_title = clean_text(new_title)
```

```
generated_titles.append(cleaned_title)

return generated_titles
```

Po wywołaniu metody generującej otrzymaliśmy następujące wyniki:

```
Generated Title 1: A Randomized, Multivariate, and Randomized Model for the Search of Small-mole
Generated Title 2: The Search for Small-molecule Protein Binders Towards DNA-Encoded Library Generation with
Generated Title 3: The Future of Learning
Generated Title 4: _____. A New Approach to the Search for Small-molecule Protein Binders. J.
Generated Title 5:
```

Dokonałiśmy fine-tuning modelu dwa razy na różnego rozmiaru zbiorach danych (1208 tytułów oraz 9429).

Pierwszy model, wytrenowany na mniejszej bazie zwrócił następujące wyniki:

```
model = GPT2LMHeadModel.from_pretrained('./gpt2-finetuned')

tokenizer = GPT2Tokenizer.from_pretrained('./gpt2-finetuned')
```

```
Generated Title 1: Coding with Textures and Identifiers for Self-Supervised Learning of Satellite Imagery
Generated Title 2: vernacular Chinese: Mandarin Long-form English Corpus for Speech Recognition with GFlowNets
Generated Title 3: Supervised Sequence Generation from Sequence Data via Reinforcement Learning Theory of Causal Models in Rein
Generated Title 4: ersatz-sequencing of sequence data for class-generic linear algebraic models using partial class-
Generated Title 5: ixML: A Model-agnostic Deep Reinforcement Learning Approach for Non-linear Action Space Discovery
```

Drugi model, wytrenowany na mniejszej bazie zwrócił następujące wyniki:

```
model = GPT2LMHeadModel.from_pretrained('./gpt2-big-finetuned')

tokenizer = GPT2Tokenizer.from_pretrained('./gpt2-big-finetuned')
```

```
Generated Title 1: Theoretical and Recent Results of a Deep Convolutional Neural Network for Human-AI
Generated Title 2: A Study in Mathematical Reasoning and Accuracy of Neural Networks from Video Using Differentiable Networks
Generated Title 3: Deep Reinforcement Learning for Vision-Based Robotic Manipulation of Objects via Self-Att
Generated Title 4: VIDE-Inspired Attacks on Language Models for Model-Based Control of Worms or
Generated Title 5: Coupled Representations for Model-Based Control of Large Language Models with Differentiable Dynamics
```

Wyraźnie widać, że trenowany model **gpt-finetuned** na 1208 tytułach zwrócił bardziej sensowne wyniki niż oryginalny model **gpt-2**, a model **gpt-big-finetuned** wygenerował tytuły jeszcze sensowniejsze, lecz w dalszym ciągu nie doskonałe. Na przykład tytuł czwarty kończy się słowem “or” (lub).

Teraz wyjaśnijmy jak przebiegał Fine-tuning.

Najpierw tak samo pobieramy dane z arXiv i zapisujemy je do pliku csv. Model **gpt-big-finetuned** został zapoznany z tytułami wielu autorów z wielu dziedzin naukowych.

```
authors = [
    "Yoshua Bengio", "Geoffrey Hinton", "Andrew Ng", "Yann LeCun", "Ian Goodfellow",
    "Jürgen Schmidhuber", "Sebastian Thrun", "Alex Graves", "Pieter Abbeel", "Sergey Levine",
    "Fei-Fei Li", "Trevor Darrell", "Daphne Koller", "Ruslan Salakhutdinov", "Zoubin Ghahramani",
    "Max Welling", "Bernhard Schölkopf", "Antonio Torralba", "Samy Bengio", "Andrej Karpathy",
    "Christopher Manning", "Richard Socher", "Shai Shalev-Shwartz", "Amos Storkey", "Nando de Freitas",
    "Christopher Bishop", "Tom Mitchell", "Michael Jordan", "David Silver", "Demis Hassabis"
]

categories = [
    "cs.LG", "cs.AI", "stat.ML", "cs.CV", "cs.CL", "cs.NE", "cs.RO", "cs.IR", "cs.DS", "cs.SD",
    "cs.CR", "cs.CY", "cs.DB", "cs.HC", "cs.IT", "cs.LG", "cs.MA", "cs.NI", "cs.PL", "cs.SE"
]
```

załadowaliśmy wejściowy model gpt-2 oraz tokenizer

```
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')

model = GPT2LMHeadModel.from_pretrained('gpt2')

tokenizer.pad_token = tokenizer.eos_token
```

załadowaliśmy zbiór danych z pliku CSV za pomocą bibliotek pandas i datasets.

```
def load_dataset_from_csv(file_path):

    df = pd.read_csv(file_path)

    return Dataset.from_pandas(df)

raw_datasets = load_dataset_from_csv("data_titles.csv")
```

Dokonaaliśmy tokenizacji zbioru danych, aby dostosować go do modelu GPT-2.

```
def tokenize_function(examples):

    return tokenizer(examples['Title'], padding='max_length',
truncation=True, max_length=128)

tokenized_datasets = raw_datasets.map(tokenize_function, batched=True)
```

Utworzyliśmy Data Collator, który jest używany do dynamicznego wstawiania tokenów wypełniających podczas trenowania modelu.

```
data_collator = DataCollatorForLanguageModeling(
```

```
tokenizer=tokenizer,  
  
mlm=False)
```

Dodaliśmy argumenty treningowe, które definiują parametry trenowania modelu.

```
training_args = TrainingArguments(  
  
    output_dir="./gpt2-big-finetuned",  
    overwrite_output_dir=True,  
    num_train_epochs=10,  
    per_device_train_batch_size=4,  
    evaluation_strategy="epoch",  
    save_strategy="epoch",  
    save_total_limit=2,  
    learning_rate=5e-5,  
    weight_decay=0.01,  
    logging_dir='./logs',  
    logging_steps=100,  
    fp16=True,  
    load_best_model_at_end=True,  
    metric_for_best_model="eval_loss",  
    greater_is_better=False,  
)
```

Opis argumentów:

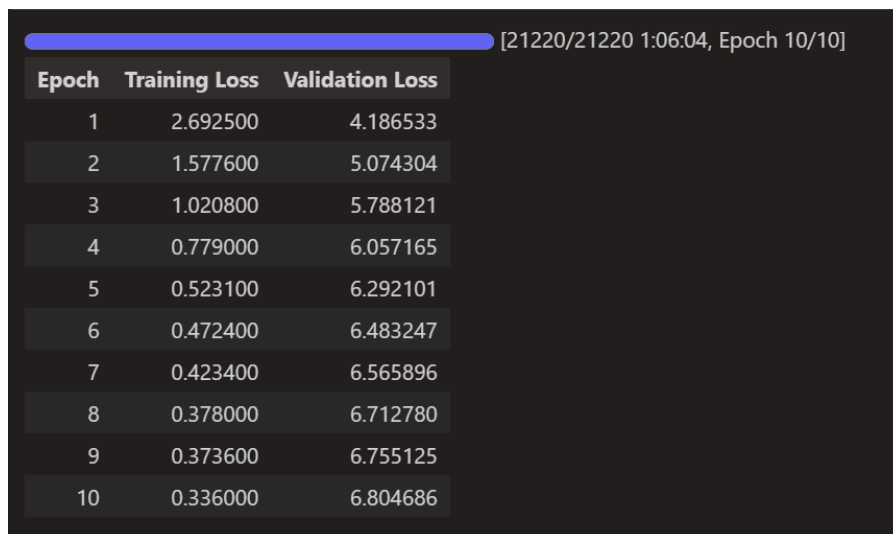
- *output_dir*: Katalog, w którym będą zapisywane wyniki trenowania.
- *overwrite_output_dir*: Jeśli ustawione na **True**, nadpisuje zawartość katalogu **output_dir**.
- *num_train_epochs*: Liczba epok trenowania.
- *per_device_train_batch_size*: Liczba próbek w jednej partii trenowania na jedno urządzenie (GPU/CPU).
- *evaluation_strategy*: Strategia walidacji modelu (np. **epoch** oznacza walidację po każdej epoce).
- *save_strategy*: Strategia zapisywania modelu (np. **epoch** oznacza zapis po każdej epoce).
- *save_total_limit*: Maksymalna liczba zapisanych modeli.
- *learning_rate*: Szybkość uczenia się.
- *weight_decay*: Współczynnik zanikania wag.
- *logging_dir*: Katalog, w którym będą zapisywane logi treningowe.
- *logging_steps*: Liczba kroków między zapisami logów.

- *fp16*: Jeśli ustawione na **True**, używa połowicznej precyzji (16-bit) do trenowania.
- *load_best_model_at_end*: Jeśli ustawione na **True**, ładuje najlepszy model na końcu trenowania.
- *metric_for_best_model*: Metryka używana do wyboru najlepszego modelu.
- *greater_is_better*: Jeśli ustawione na **False**, niższa wartość **metric_for_best_model** oznacza lepszy model.

Zainicjalizowaliśmy trenera, dokonaliśmy samego treningu i zapisaliśmy ulepszony model:

```
trainer = Trainer(  
    model=model,  
    args=training_args,  
    data_collator=data_collator,  
    train_dataset=train_dataset,  
    eval_dataset=val_dataset,  
)  
trainer.train()  
trainer.save_model("./gpt2-big-finetuned")  
tokenizer.save_pretrained("./gpt2-big-finetuned")
```

Cały trening modelu gpt2-big-finetuned zajął ponad godzinę, podzielony na 10 epok.



[21220/21220 1:06:04, Epoch 10/10]

Epoch	Training Loss	Validation Loss
1	2.692500	4.186533
2	1.577600	5.074304
3	1.020800	5.788121
4	0.779000	6.057165
5	0.523100	6.292101
6	0.472400	6.483247
7	0.423400	6.565896
8	0.378000	6.712780
9	0.373600	6.755125
10	0.336000	6.804686

Klasyczne techniki NLP

wykorzystane technologie:

-biblioteka NLTK - jest wszechstronną biblioteką języka Python, używaną do przetwarzania i analizy języka naturalnego. Jest ona często stosowana w badaniach nad językiem, edukacji i aplikacjach komercyjnych, których elementem jest analiza tekstu.

Na początek pobierana jest lista tytułów z serwisu ARxiv za pomocą biblioteki w języku Python udostępnianej przez ten serwis. W tym wypadku umożliwiamy użytkownikowi wprowadzenie autora i kategorii prac oraz ilość prac pobieraną do analizy:

Narzędzie do generacji tytułów publikacji naukowych w oparciu o istniejące tytuły

1. Wybierz autora i kategorię i ilość tytułów na wejściu:

```
[103]: author = input("Autor:")
      Autor: Yoshua Bengio

[100]: category = input("Kategoria:")
      Category: cs.LG

[118]: maxresultsinput = int(input("ilość tytułów"))
      ilość tytułów 1000
```

2. Pobieranie tytułów z Arxiv:

```
[119]: import arxiv

      client = arxiv.Client()
      search = arxiv.Search(
          query = f'au:"{author}" AND cat:{category}',
          max_results = maxresultsinput,
          sort_by = arxiv.SortCriterion.SubmittedDate
      )

      papers = list(client.results(search))
```

Następnie wykonujemy tokenizację tytułów na poszczególne słowa w oparciu o moduł “tokenize” biblioteki “nltk” w języku Python.

3.tokenizacja:

```
from nltk.tokenize import word_tokenize

tokenized_titles = [word_tokenize(title) for title in titles]
```

Stemming oraz usuwanie znaków stopu zostało przez nas pominięte ze względu na dużą “agresywność” metod stemmingu biblioteki nltk w przypadku słownictwa stosowanego w analizowanych tytułach. Po porównaniu wyników stwierdziliśmy że pominięcie tych kroków daje lepsze rezultaty dla wygenerowanych tytułów.

Kolejnym krokiem jest lematyzacja otrzymanych tokenów. Operacja ta jest wykonywana przy użyciu obiektu kalsy WordNetLemmatizer modułu stem.

4. Lematyzacja

```
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

lemmatized_titles = []

for title in tokenized_titles:
    lemmatized_title = [lemmatizer.lemmatize(word.lower()) for word in title]
    lemmatized_titles.append(lemmatized_title)
```

W oparciu o listę zlematyzowanych tytułów tworzone są “bigramy”:

5. Konstruowanie bigramów

```
from collections import defaultdict
import random

def construct_bigrams(titles):
    bigrams = defaultdict(list)
    for title in titles:
        for i in range(len(title)-1):
            bigrams[title[i]].append(title[i+1])
    return bigrams

bigrams = construct_bigrams(lemmatized_titles)
```

Na koniec przy użyciu bigramów tworzone są nowe tytuły:

6. Tworzenie nowych tytułów

```
def generate_title(bigrams, length=6):
    current_word = random.choice(list(bigrams.keys()))
    title = [current_word]
    while len(title) < length:
        next_words = bigrams.get(current_word, [])
        if not next_words:
            break
        next_word = random.choice(next_words)
        title.append(next_word)
        current_word = next_word
    return ' '.join(title)

# Generate and print new titles
for _ in range(5):
    print(generate_title(bigrams))

robotics : mesh agnostic neural network
red-teaming and fine-tuning generative adversarial learning
design of independent component estimation for
a3t : a single nucleotide resolution
denoising energy matching based sample from
```