Published in Geek Culture

Lucas Soares Follow
Apr 27, 2021 · 6 min read · · Listen

Save

Photo by Aleks Dorohovich on Unsplash

Summarizing Papers With Python and GPT-3

Using python and openai's GPT-3 to summarize AI papers

When access to the GPT-3 API was released to researchers and engineers (upon request) I immediately requested it so I could see what kind of interesting tools one could write and what kind of interesting research questions could be asked.

Upon trying the API and the awesome tools that come with it, I realized that one of my favorite applications of GPT-3 was for ***paper summarization,*** so I decided to test it out.

Today I will show you how to build a simple python tool to summarize papers directly from their arxiv address.

Photo by Seven Shooter on Unsplash

Steps to summarize a paper with GPT-3

The process itself is quite simple:

1. Download the paper

2. Convert from pdf to text

3. Feed the text to the GPT-3 model using the openai api

4. Show the summary

1. Download the paper

First let's import our dependencies

```
import openai
import wget
import pathlib
import pdfplumber
import numpy as np
```

[Updated 2021 — the GPT-3 model is now available for all]

Here you will have to install `openai` for interfacing with GPT-3 in case you have an API key, if you don't have it you can get started here.

You will also need `wget` for downloading the pdf from the arxiv page and `pdfplumber` for converting the pdf to text:

Now, let's write a function that downloads a pdf from an arxiv address, the paper I will be using is *'Understanding training and generalization in deep learning by Fourier analysis'* by Zhi-Qin John Xu.

```
def getPaper(paper_url, filename="random_paper.pdf"):
    """
    Downloads a paper from it's arxiv page and returns
    the local path to that file.
    """
    downloadedPaper = wget.download(paper_url, filename)
    downloadedPaperFilePath = pathlib.Path(downloadedPaper)

    return downloadedPaperFilePath
```

Here, I am using the `wget` package to directly download the pdf and return a path to the downloaded file.

2. Convert from pdf to text

Now, I will write another function to convert the pdf to text so that GPT-3 can actually read it.

```
paperFilePath = "random_paper.pdf"
paperContent = pdfplumber.open(paperFilePath).pages

def displayPaperContent(paperContent, page_start=0, page_end=5):
    for page in paperContent[page_start:page_end]:
        print(page.extract_text())
displayPaperContent(paperContent)

# Output
Understanding training and generalization in deeplearning by Fourier
analysisZhi-
QinJohnXu∗8NewYorkUniversityAbuDhabi1AbuDhabi129188,UnitedArabEmirat
es0zhiqinxu@nyu.edu2 voN Abstract 92 Background: It is still an open
research area to theoretically understand why
DeepNeuralNetworks(DNNs)—equippedwithmanymoreparametersthantrain- ]
ing data and trained by (stochastic) gradient-based methods—often
achieve re-Gmarkably low generalization error. Contribution: We
study DNN training byL Fourier analysis. Our theoretical
frameworkexplains: i) DNN w[illegible]c). gradient-based methods
often endows low-frequency [illegible]he targetsc function with
a higher priority during the training; ii) Small initialization
leads[ to good generalizationability of DNN while preservingthe
DNN’s ability to ﬁt   any function. These results are further
conﬁrmed by experiments of DNNs
...
...
```

Here, I extracted the text per page from the paper and wrote a function `displayPaperContent` to show the corresponding content. There are issues with the conversion that I won’t address in this article because I want to focus just on the summarization pipeline.

3. Feed the text to the GPT-3 model using the openai api

Now, for the fun stuff, let’s write a function that get’s the paper content and feeds it to the GPT-3 model using openai’s api:

```
def showPaperSummary(paperContent):
    tldr_tag = "\n tl;dr:"
    openai.organization = 'organization key'
    openai.api_key = "your api key"
    engine_list = openai.Engine.list() # calling the engines
available from the openai api

    for page in paperContent:
        text = page.extract_text() + tldr_tag
        response =
openai.Completion.create(engine="davinci",prompt=text,temperature=0.
3,
            max_tokens=140,
            top_p=1,
            frequency_penalty=0,
            presence_penalty=0,
            stop=["\n"]
        )
        print(response["choices"][0]["text"])
```

Let's unpack what is happening here:

```
tldr_tag = "\n tl;dr:"
```

In this step I am writing the tag so that the GPT-3 model knows when the text stops and when it should start the completion (which in this case is a summary).

```
openai.organization = "the openai organization key"
openai.api_key = "your api key"
engine_list = openai.Engine.list() # calling the engines available
# from the openai api
```

Here, I am setting up the environment for using the openai API.

```
for page in paperContent:
        text = page.extract_text() + tldr_tag
```

Now, I am extracting the text from each page and feeding it to the GPT-3 model. In the future I want to write an extension to this script so that it can get paragraph by paragraph, so that the model could see semantically connected chunks from the text.

```
response =
openai.Completion.create(engine="davinci",prompt=text,temperature=0.
3,
            max_tokens=140,
            top_p=1,
            frequency_penalty=0,
            presence_penalty=0,
            stop=["\n"]
        )
        print(response["choices"][0]["text"])
```

Finally, I am feeding the text to the model setting a max tokens of 140 for the response (half the size of a tweet) for each page and printing that to the terminal.

4. Show the summary

Ok great! Now that we have our model set up, let’s run it and see the results:

```
paperContent = pdfplumber.open(paperFilePath).pages
showPaperSummary(paperContent)

# Output:

The power spectrum of the tanh function exponentially decays w.r.t.
frequency.

Thegradient-
basedtrainingwiththeDNNwithonehiddenlayerwithtanhfunction
Thetotallossfunctionofasinglehiddenlayer DNNis


 The initial weights of a deep neural network are more important
than the initial biases.
 We can use Fourier analysis to understand the gradient-based
optimization of DNNs on real data and pure noise. We found that DNNs
have a better generalization ability when the loss function is
ﬂatter. We also found that the DNN generalization ability is better
when the DNN parameters are smaller.

 The code is available at https://github.com/yuexin-
wang/DeepLearning-Music .

 Theorem 3 is true.
 We can use the same method to prove the theorem.
 The DNN is a very powerful tool that can be used for many things.
It is not a panacea, but it is a very powerful tool.
```

```
 Deep learning is only one of many tools for building intelligent
systems.
```

And there it is! This is so cool! Due to issues with the pdf conversion, two page summaries appeared glued together but overall it gives an ok description of the paper page by page, even highlighting the source code link!

Putting it all together

The full code is this:

```
import openai
import wget
import pathlib
import pdfplumber
import numpy as np

def getPaper(paper_url, filename="random_paper.pdf"):
    """
    Downloads a paper from it's arxiv page and returns
    the local path to that file.
    """
    downloadedPaper = wget.download(paper_url, filename)
    downloadedPaperFilePath = pathlib.Path(downloadedPaper)

    return downloadedPaperFilePath


def showPaperSummary(paperContent):
    tldr_tag = "\n tl;dr:"
    openai.organization = 'API KEY org'
    openai.api_key = "your openAI key"
    engine_list = openai.Engine.list()

    for page in paperContent:
        text = page.extract_text() + tldr_tag
        response =
openai.Completion.create(engine="davinci",prompt=text,temperature=0.
3,
            max_tokens=140,
            top_p=1,
            frequency_penalty=0,
            presence_penalty=0,
            stop=["\n"]
        )
        print(response["choices"][0]["text"])
paperContent = pdfplumber.open(paperFilePath).pages
showPaperSummary(paperContent)
```

Thoughts on Summarization

I think summarization models are great. They will never replace the important process of actually reading an entire paper, but they can serve as a ***tool to explore a wider range of interesting scientific discoveries***.

This post was more a proof of concept, mostly because the issues with formatting the text converted from the pdf and the actual quality of the summarization are still things to tweak quite a bit. However, I feel like we have come a long way and now, more than ever, we are getting closer to having something that will actually allow us to process a bigger array of scientific information.

[Update 22/11/2021]

I made a youtube video about this that you can check out here:

If you want to dive deeper into natural language processing and text summarization, check out this course:

> *This is an affiliate link, if you use the course I get a small commission, cheers! :)*

- **Natural Language Processing for Text Summarization**

If you liked this post connect with me on Twitter, LinkedIn and follow me on Medium. Thanks and see you next time! :)

Data Science Artificial Intelligence Python Programming

Software Development