Table of Contents

1. Introduction to AI for Generating Text

1.1. Welcome!

This workshop is a gentle introduction to using advanced AI tools, for beginners.

We will work on a popular platform for machine learning, the HuggingFace🤗platform, which offers tools uploaded by a community of AI developers. We will explore and play around with these tools, which range over various tasks and topics.

Then, we will shift to running these tools with coding, with the Python programming language on Google Colab.

How many people have some background with Python?

- This workshop will be a gentle introduction to people who have never coded with Python before.

1.2. goals:

- understanding of language models available to you and how to start using them on their own.
  - build toward skills for using them in programming, even if you don't have a background
  - how program code abstracts certain processes
- familiarity of underlying technical processes that power the tools, like word vectors.
- exposure to ethical issues with training and producing LLMs, the ways they perpetuate bias and discrimination.
- all of that within the larger goal of seeing what you can do with these AI tools on your own.

1.3. introduction to LLMs

Before starting, let's talk a bit about how they work.

The goal is to de-mystify how Large Language Models (a kind of AI software) work under the hood.

Will be helpful when we get to the next section, when we move to the HF website.

And will help understand some of the ethical issues with the way these models are trained.

1.3.1. How does ChatGPT work?

How does it know what to respond when someone asks it a question?

More specifically, how does it know what language to generate, what words follow other words?

- by prediction.
- it learns by reading. Gains an understanding of language from processing massive amounts of text, deriving patterns.

Still, how does it know what certain words mean? How to use them in a sentence?

How do we turn words into something that a computer can understand?

It quantifies words. Language into numerical forms, representations.

1.3.2. Word Embeddings / Word Vectors

One key development in history of AI, 2013 paper.

We use "word embeddings", "word vectors."

- technically: representations of language in vector space, graphical space.
- Each word is represented by a series of numbers, with each of those numbers representing it's relationship to another word. How closely they are related.
- show vector for "cat" vs "dog":

word	tiger	cute	bones	wolf
cat	.90	.99	.40	.35
dog	.35	.99	.85	.90

Each word is expressed as a series of numbers. Each number a different dimension of the vector.

We can plot "cat" and "dog" according to x and y axes, for example, "tiger" and "cute"

- x is tiger
- y is cute

So while both dog and cat would be high on the cute axis, on the Y axis, only cat would be higher up on the X axis, the tiger axis. Because cats are like tigers, whereas dogs are not.

Except we don't have just x, y, or z. We have hundreds, thousands of vectors.

Why do this to words? Why represent them as numbers?

- So we can do math!
- We can do linear algebra.
    - Cat and kitten are close to each other, that means they have similar meanings.
    - We can do cosine similarity, finding out what two vectors are similar to each other in shape/direction actually gives us a sense of their semantic similarity.
    - Opens up a world of algebra, calculus, that we can do with language.

1.3.3. king - man + woman = queen

Famous formula:

- King - Man + Woman = Queen
  - we can "do language" using math!
- vector(”King”) - vector(”Man”) + vector(”Woman”) = vector("Queen")
- Notice for a moment all the assumptions being made about gender here.
  - That the difference between a king and a queen has to do with gender.
  - What exactly is being calculated when we subtract "man" and add "woman"?
    - Is it biological sex that's being substracted?
    - Is it gender conventions, femininity and masculinity? Kings are embody a masculine ideal, and queens a feminine one?
    - What qualities are being assumed to pertain to each gender and each

    role?
- Not a massive deal, but interesting, because this is the formula that introduced the power of word vectors to the world. So the assumptions it plays on must be deeply embedded across society.

Why am I saying all this about word vectors?

- to de-mystify the tool.
  - these tools are not magic, they are not intuitive, possibly not even "intelligent", they can just do a lot of math.

It's alright if this doesn't make sense. It's advanced ML.

See more:

- Word2Vec paper, 2013.
- (and great explanation by Jay Alammar)

1.4. Huggingface🤗platform

Now let's move to the platform we will be using, HuggingFace.

- HF is an AI research and development company based in Brooklyn, New York City.
- A platform for Machine Learning: compute & collaborative spaces for AI models, datasets, and more.
  - like a github for ML, if github had additional "hubs" for things besides just code (datasets, papers, apps).
  - also can run software directly on the website, "platform"

1.4.1. "models hub"

Start with the "models hub"

Here contains AI models created by the community/ HF users.

- a little overwhelming interface, I will explain it in a moment.
- navigation goes from left to right
  - on left side, there's tasks, like text classification.
  - on right side, there's models. We are going to narrow down the models.

Search for "gpt-neo" in the text box.

- gpt-neo-125m
    - a model developed by EleutherAI, a non-profit research lab.
    - part of a larger family of models named "gpt-neo" with the size at the end.
- notice "**model size**". How big is it?
    - 125m parameters. That's how many inputs goes into inference. Includes things like word vectors, but also different kinds of inputs.
        - size is an indication of complexity. The larger the size, the more likely that the model will preform well.
- notice the "**license**":
    - MIT license. Very permissive, part of the "Open Source" licenses.
        - the model is totally open to download and modify as you wish, even for commercial purposes.

Practice running inference here for a mintue. Anything that you notice about the results?

- it's repetitive.
    - the repetition problem is caused by the traits of our language itself.
    - it generates words that have the highest likelihood. The words that have this likelihood tend to be the same ones, over and over again.

Let's look at one more model, to start a conversation about how these products are created and then disributed.

Go back to most download, select Llama,

- by Meta, aka Facebook.
- in terms of licensing, this is the most restrictive, by far.
    - Meta champions this model as "open source" but it is nothing like that. The license prevents you from making anything that can compete with them.
        - "open source" vs open access models: not everything open access is open source!
        - not sharing the model's training data or the code used to train it unless you sign their agreement.

Just to be aware of some of the terminology, marketing terms, used to promote this technology, which is really misleading.

1.4.2. training process

:notes: It's important to consider how these models were created, and what's going into the training process.

In addition to potential violations of copyright, issues with bias and discrimination.

- The ways that training data is cleaned (or not cleaned).

Where do we get most of the data used to train these models?

- scraped from the internet, most of them.
    - contains all the worst parts of the internet, too. All of the discrimination and violence.

Crucially, you cannot automate the removal of bias and discrimination, because t can be situational, nuanced.

- attempts to automate this have failed:

See "List of Dirty Obscene…"

- used to filter out any web pages that contained these words. Just remove the whole page.

- but it didn't work. GPT-2 was still dirty and obscene.

1.4.3. think/pair/share activity

Why do you think this method didn't work. Think about it for 2 minutes.

- Position: I might say something that is offensive, whereas if someone else, from different background or in a different context says the same thing, it's not offensive.
- Context: reclaiming the term, explaining why it's hurtful?

There's a race to get these products out there, so people aren't taking the time needed to adequately clean the data and make sure it's safe. That's just a fact.

- RLHF - "reinforcement learning from human feedback"

Something to keep in mind! There's a lot of work to be done in bias and discrimination

:end:

1.5. inference with python on colab: inference & abstraction:

Now that you have a sense of how inference works on the HF website, we are going to practice running inference on Google Colab.

Our goal is to create a text generator, using Python code, taking the following steps:

- Will use the model, "gpt-neo-125m", and write code that imports this model into the colab coding space.
- Then we will write code to process an input text, and generate an output, a continuation.

We'll talk about some programming concepts along the way.

- how programming languages abstract data through variables, learning how to read these layers of abstraction.

1.5.1. google colab, REPL, variables:

https://colab.research.google.com/

A cloud computing platform, where you can run code directly in the browser.

Python can be difficult to install and configure, it's system specific. Also distributions are large and take up space on your laptop. Cloud computing takes away these issues.

- one particular plus is the colab offers computing power that is strong enough to handle these models.

Basic interface:

- cells to run the code, an "expression"

```
1 + 1
# run code by pressing shift-return, or the play button.

x = 5

y = 7

x + y
```

```
# all variables saved.

# "interactive mode" - evaluate the expression, print result, back to
# prompt for more expressions.
```

1.5.2. import the models

on the toolbar, where it says RAM DISK, change the hardware accelator to GPU.

Then go back to the models page.

Search for gpt-neo, select 125m. On the top right, click on "Use in Transformers."

Copy that code, and paste it to your google colab cell.

```
# Use a pipeline as a high-level helper
from transformers import pipeline

pipe = pipeline("text-generation", model="EleutherAI/gpt-neo-125m")
```

Here we have a function, called `pipeline()`, which takes parameters (a fancy word for input).

The parameters specify the task and the model that we will be using.

We save the function to a variable called `pipe`, which we will later use to process our prompt.

1.5.3. run inference

Now we are going to "run inference."

First, we will type up a prompt, and save it to a variable `prompt`. Then we will pass that prompt to the `pipe` variable

```
prompt = "Hello, my name is Filipa and"

pipe(prompt)
```

Congratulations! We have just ran inference in Python.

What if we want to take this to the next level. What if we want to save the output.

How would we save the output?

```
response = pipe(prompt)
```

Running the same thing that we created before, saving the output to a new variable, called `response`.

Here we see the levels of abstraction at play.

- saved the prompt text to a variable, and passing that prompt into the pipe.
- then saving the pipeline function to a new variable

Why would you want to do this? Why would you want to save it?

Abstraction is the foundation for all programs.

It allows us to then *do things* with our code. Maybe make a chatbot interface. Maybe make a personal assistant. We've saved the output, we can now use it to do other things.

Basic concept of programing is creating running functions, saving the results, passing those results to new functions, and so on.

These abstractions of processes are what makes software.

1.6. IF TIME: data structures.

Now let's examine more closely our response.

```
output
# [{'generated_text': "Hello, my name is Filipa and I'm a newbie in
# the world of web development."}]

type(output)
# list
```

What kind of data is this?

- `list` is a collection of objects, or bits of information. So our output is saved as this collection type of object.

What if we wanted to extract just the output text, not the rest of the data, how would we go about it?

We are going to examine this list to see what else is contained inside. For that we will use "indexing."

Indexing is picking out object by their position within another object, like a list (though it also works for strings). The first item is zero, the second item is 1, and so on.

Does anybody know what we start with zero? (Because it is based on offsets. Think like a computer. The first item is the starting place, we don't have to move anywhere to access it. But the second item, we have to move one place to the right, so it's 1).

```
name = Filipa
name[0]
# F
name[1]
# i

output[0]
# {'generated_text': "Hello, my name is Filipa and I'm a newbie in the
# world of web development."}

type(output[0])
# dict
```

Now we are getting closer, we got rid of the brackets. Inside this list, we actually have a new data type, called a `dict`. This stands for data structured into key:value pairs.

Let's look at an example:

```
# key, value pairs

filipa = {
    'first_name': 'filipa'
```

```
    'last_name': 'calado',
    'job': 'library',
    'age': '34',
    'degree': 'literature'
}


type(filipa)
# dict
```

To get items from a dict, you use a different method, accessing them by their keys.

```
filipa['first_name']
# filipa

filipa['degree']
# literature
```

So, we can combine what we know about list indexing and accessing items in a dict by keys to pull out just the response text

```
output[0]['generated_text']

# then we can save it to a variable!

text = output[0]['generated_text']
```

Author: CaladoF
Created: 2024-02-25 Sun 18:16
Validate