CSC108H Winter 2024 Worksheet 27 : Tiny Language Model 1

Introduction

Large Language Models are computer systems that can generate new text based on huge amounts of existing information. An example of a Large Language Model (or LLM) is ChatGPT. Users interact with a LLM by giving it prompts, and the LLM produces output that (it "thinks") matches the prompt. Often (but not always!) it does a very good job.

Over the next few hours of class, we are going to develop a simpler version of a LLM. We'll call ours a "Tiny Language Model", or TLM. It will be able to read some data we provide, and generate output that sounds similar to the input data. For example, we might give our program Shakespeare's plays and then generate a new random sequence of words that sounds remarkably like Shakespeare, and is even somewhat coherent.

We strongly recommend that you work in a group with your classmates on this exercise so that you can help each other.

For this exercise, we will make a few design decisions:

- Capitalization matters. For example, **`what`** and **`What`** are considered to be different words.
- Punctuation is included as part of a word. For example, **`what`** and **`what,`** are considered to be different.

An Example: Understanding the Algorithm

Consider this old German saying: `What I spent, I had; what I saved, I lost; what I gave, I have.`

We're going to write a new saying using only those words, in somewhat random order. We will call the text we base the new saying on the *training text*.

Here are the rules we're going to follow:

- To start, choose from the training text a word such as **`I`** and print that word. We will use the term *current context* to refer to the word that was just printed.
- Repeatedly:
  - Make a list of the words that follow the current context. In our example, these words follow **`I`**:

    ```
    spent,  had;  saved,  lost;  gave,  have.
    ```

  - Choose one of the words that follow the current context. That word is the new current context.

  Continuing our example, let's choose to print **`lost;`**, making it our new current context. There is only one word that follows **`lost;`**, the word **`what`**. We choose **`what`** and print it, making it our new current context. So far, we have printed this:

  ```
  I lost; what
  ```

Now it's your turn: repeat the process. The current context is **`what`**. Write the word(s) that follow the current context (if the same word appears multiple times, write it down again):

Choose one, print it, and note the new current context. Continue following the steps until you are sure you understand the process.

A longer current context

In the previous example, the current context was only one word long, but you can imagine using 2 words for the current context. Every time you choose and print a new word, the new current context is the second word of the old current context followed by the new word.

For example, if the current context was **`what I`** then here are the words that follow this context:

```
saved,  gave,
```

Let's say we pick **`gave,`**. We print **`gave,`**, and the new 2-word current context is **`I gave,`**.

- Assume that the current context is **`I gave,`**. Write the word(s) that follow the current context, then write the new current context.

CSC108H Winter 2024 Worksheet 27 : Tiny Language Model 1

A Larger Example

Here is a longer training text. Treat newlines like spaces — the last word on each line is followed by the first word on the next line.

```
The sun did not shine.
It was too wet to play.
So we sat in the house
All that cold, cold, wet day.
I sat there with Sally,
We sat there we two.
And I said, "How I wish
We had something to do!"
Too wet to go out
And too cold to play ball.
So we sat in the house.
We did nothing at all.
So all we could do was to
Sit! Sit! Sit! Sit!
And we did not like it.
Not one little bit.
```

Using a 1-word context:

1. Assume that the current context is **`to`**. Write the word(s) that follow the current context:

2. Assume that the current context is **`wet`**. Write the word(s) that follow the current context:

Using a 2-word context:

1. Assume that the current context is **`did not`**. Write the word(s) that follow the current context:

2. Choose one of those words randomly. Now write the new current context:

If the length of the output to be generated is 11 words and there is one word of context, here are some possible outputs:

- cold, wet to play ball. So we two. And too cold
- could do was too wet to play. So we sat in
- said, "How I sat there with Sally, We did nothing at
- I said, "How I sat in the house. We had something
- Sit! Sit! Sit! Sit! Sit! Sit! Sit! Sit! Sit! Sit! Sit!

Finish writing the following output (11 words long, using one word of context):

We did not shine. It was ______________________________

CSC108H Winter 2024 Worksheet 28 : Tiny Language Model 2
Representing the Data

Now that we've figured out an algorithm for our Tiny Language Model, we need to decide how to represent the data in a Python program.

- **The new generated output**:

  Which Python type(s) could we use to represent this data? ________________________________

- **The number of words in the generated output**:

  Which Python type(s) could we use to represent this data? ________________________________

- **The number of words of context**:

  Which Python type(s) could we use to represent this data? ________________________________

- **The possible current contexts and the words that follow them**:

  To determine how to represent this information, let's start by writing down what we need to keep track of. We'll use this sample training text:

  So, as fast as I could, I went after my net.

  Using one word of context, complete the table below:

Context (one word)	**Possible next word(s)**
So,	as
as	fast I
fast	
I	
could,	
went	
after	
my	
net.	

  Using two words of context, complete the table below:

Context (two words)	**Possible next word(s)**
So, as	
as fast	
fast as	
as I	
I could,	
could, I	
I went	
went after	
after my	
my net.	

  Which Python type should we use to represent this data? **Write it as you would in a type contract.**

  ____

CSC108H Winter 2024 Worksheet 29 : Tiny Language Model 3
Reading and Storing the Training Text

In your program, you need to read the training text and then build the data structure you designed on the previous worksheet.

Storing the training text

You should represent the words in the training text as a list of strings. The length of the list is the number of words in the training text. Here is an example of a word list:

```
['And', 'the', 'fan,', 'and', 'the', 'cup,', 'And', 'the', 'ship,', 'and', 'the', 'fish.']
```

Your program will need to read data from a file and create the word list. This is surprisingly easy, provided that you use this method:

```
>>> help(str.split)
Help on method_descriptor:

split(self, /, sep=None, maxsplit=-1)
    Return a list of the words in the string, using sep as the delimiter string.

    sep
      The delimiter according which to split the string.
      None (the default value) means split according to any whitespace,
      and discard empty strings from the result.
    maxsplit
      Maximum number of splits to do.
      -1 (the default value) means no limit.
```

Given a variable `training_file` that refers to a file open for reading, write Python statement(s) to build the list, assigning that list to a variable called `words`. Which of the four file-reading approaches is the easiest in this situation? (The four approaches: `read(), readline(), readlines(), for line in training_file`)

```
training_file = open('original_data.txt', 'r')
```

Building the data structure

Once you have built your words list, you can accumulate the context information in a dictionary where the keys are tuples of strings and the values are lists of strings.

Given the words list from the first section of this worksheet and a context length of 2, complete the dictionary below. We have given you the first key `('And', 'the')`. Write the list of values for it, and then complete the dictionary with all the 2-word contexts and their values.

```
{ ('And', 'the') : [








}
```