Deep Learning 1

Section – 1

Purpose of the Script

This Python script is designed to scrape random WikiHow articles, extract important details like subheadings and paragraphs, and save this information into a CSV file. Here's a step-by-step breakdown of how it works.

Step-by-Step Explanation

1. **Importing Libraries**

```
import requests
from bs4 import BeautifulSoup
import csv
import re
```

The script starts by importing necessary libraries:

- requests to handle HTTP requests and fetch the webpage content.
- BeautifulSoup from the bs4 package to parse and navigate through the HTML content.
- csv to write the extracted data into a CSV file.
- re (regular expressions) to clean and format text.

2. **Setting Up the URL**

```
url = 'https://www.wikihow.com/Special:Randomizer'
```

This URL points to a special WikiHow page that redirects you to a random article each time you visit. By using this URL, we can scrape different articles repeatedly.

3. **Opening the CSV File**

```
with open('wikiHow.csv', mode='a', newline='', encoding='utf-8') as csv_file:
    writer = csv.writer(csv_file)
```

The script opens a CSV file for writing. The mode='a' means the file will be opened in append mode, so new data will be added without overwriting existing data. The newline='' and encoding='utf-8' parameters ensure the file is written correctly.

4. **Looping to Scrape Multiple Articles**

```
    for count in range(30):
```

This loop runs 30 times to scrape 30 different random articles. Each iteration of the loop handles one article.

5. **Fetching and Parsing the Webpage**

```
        response = requests.get(url)
        html_content = response.content
        soup = BeautifulSoup(html_content, 'html.parser')
```

Inside the loop, the script sends an HTTP GET request to the WikiHow URL to fetch the webpage content. It then uses BeautifulSoup to parse this HTML content, allowing us to extract the information we need.

6. **Extracting the Article Title**

```
    article_title = soup.find('title').text.strip()
     print(article_title + " " + str(count))
```

The script finds the article's title from the HTML using the <title> tag and prints it along with the current count. This helps track which article is being processed.

7. **Extracting Subheadings and Paragraphs**

```
subheadings = []
paragraphs = []

steps = soup.find_all('div', {'class': 'step'})
for step in steps:
   subheading_element = step.find('b')
   if subheading_element is not None:
                subheading_text = subheading_element.text.strip().replace('\n', '')
                subheading_text = subheading_text.encode('ascii',
errors='ignore').decode()
                subheading_text = re.sub(r'\s+', ' ', subheading_text)
                print(subheading_text)
                subheadings.append(subheading_text)
                subheading_element.extract()
```

Here, the script initializes two lists to store subheadings and paragraphs. It then finds all the sections in the article marked with the class step. For each section, it looks for subheadings inside <b> tags, cleans up the text, and adds it to the subheadings list.

```
               span_tag.extract()
               paragraph_text = step.text.strip().replace('\n', ' ').replace('\r', '')
               paragraph_text = paragraph_text.encode('ascii',
errors='ignore').decode()
               paragraph_text = re.sub(r'\s+', ' ', paragraph_text)
               print(paragraph_text)
               paragraphs.append(paragraph_text)
```

It also removes unwanted tags and extracts the text from the remaining content, formatting it by removing newlines and extra spaces. This text is added to the paragraphs list.

8. **Saving the Extracted Data**

```
if len(subheadings):
            for i in range(len(subheadings)):
                writer.writerow([article_title, subheadings[i], paragraphs[i]])
```

Finally, if there are any subheadings, the script writes each subheading and its corresponding paragraph, along with the article title, into the CSV file.