Introduction to Deep Learning (CS474)

Lecture 5

Rupam

Outline

- Representing data through Pytorch **Tensor**
    - Representing text

Representing text

- Deep learning has taken the field of natural language processing (NLP) by storm, particularly using models that *repeatedly* consume a combination of new input and previous model output.

- Our goal is to turn **text** into something a neural network can process: a tensor of numbers.

- If we can do that and later choose the right architecture for our text-processing job, we'll be in the position of doing NLP with PyTorch.

Representing text

- There are **two** particularly intuitive levels at which networks operate on text:
  - at the character level, by processing one character at a time, and
  - at the word level, where individual words are the finest-grained entities to be seen by the network.
- The technique with which we encode text information into tensor form is the same whether we operate at the character level or the word level.

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMAN

Representing text: Example

- Let's start with a character-level example.

- First, let's get some text to process.

- Download and save the text file in your Google Drive from the following link.

  http://www.gutenberg.org/files/1342/1342-0.txt

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMAN

Representing text: Example

```python
from google.colab import drive
drive.mount("/content/drive/")

import numpy as np
import torch


with open('/content/drive/My Drive/Deep Learning (CS474)/1342-0.txt', encoding='utf8') as f:
    text = f.read()
```

- Every written character is represented by a **code**: a sequence of bits of appropriate length so that each character can be uniquely identified.
- The simplest such encoding is **ASCII**.
- Popular encodings are **UTF-8**, **UTF-16**, and **UTF-32**, in which the numbers are a sequence of 8-,16-, or 32-bit integers, respectively.

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMAN

Representing text: Example

- We are going to **one-hot encode** our characters.
- In our case, since we loaded text in English, it is safe to use ASCII and deal with a small encoding.
- At this point, we need to **parse** through the characters in the text and provide a **one-hot encoding** for each of them.
- Each character will be represented by a ***vector*** of length equal to the number of different characters in the encoding.
- This vector will contain **all zeros except a one** at the index corresponding to the location of the character in the encoding.

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMAN

Representing text: Example

- Continuing with the earlier Notebook:

```
# We first split our text into a list of lines and pick an arbitrary line to focus on

lines = text.split('\n')
line = lines[200]
line

#Let's create a tensor that can hold the total number of one-hot-encoded characters for the whole line:

letter_t = torch.zeros(len(line), 128)
letter_t.shape
```

```
'       Michaelmas, and some of his servants are to be in the house by'

torch.Size([68, 128])
```

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMAN

Representing text: Example

- Continuing with the earlier Notebook:
- Note that **`letter_t`** holds a one-hot-encoded character per row.
- Now we just have to set a **one** on each row in the correct position so that each row represents the correct character.

```
for i, letter in enumerate(line.lower().strip()):
    letter_index = ord(letter) if ord(letter) < 128 else 0  # <1>
    letter_t[i][letter_index] = 1
```

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMAN

Rupam

Representing text: Example

- Continuing with the earlier Notebook:

```
def clean_words(input_str):
    punctuation = '.,;:"!?”“_-'
    word_list = input_str.lower().replace('\n',' ').split()
    word_list = [word.strip(punctuation) for word in word_list]
    return word_list

words_in_line = clean_words(line)
line, words_in_line
```

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMAN

Representing text: Example

- Continuing with the earlier Notebook:

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mount("/content/drive/", force_remount=True).
('      Michaelmas, and some of his servants are to be in the house by',
 ['michaelmas',
  'and',
  'some',
  'of',
  'his',
  'servants',
  'are',
  'to',
  'be',
  'in',
  'the',
  'house',
  'by'])
```

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMAN

Representing text: Example

- Continuing with the earlier Notebook:

```
# Next, let's build a mapping of words to indexes in our encoding.

word_list = sorted(set(clean_words(text)))
word2index_dict = {word: i for (i, word) in enumerate(word_list)}

len(word2index_dict), word2index_dict['impossible']
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/aut

Enter your authorization code:
··········
Mounted at /content/drive/
(7278, 3383)
```

- Note that **word2index_dict** is now a dictionary with *words* as keys and an *integer* as a value.
- We will use it to efficiently find the index of a word as we one-hot encode it.

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMAN

Representing text: Example

- Let’s now focus on our sentence.
- We break it up into words and one-hot encode it.
- We populate a tensor with one **one-hot-encoded** vector per word.
- We create an empty vector and assign the one-hot-encoded values of the word in the sentence.

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMAN

Representing text: Example

- Continuing with the earlier *Notebook*:

```
word_t = torch.zeros(len(words_in_line), len(word2index_dict))
for i, word in enumerate(words_in_line):
    word_index = word2index_dict[word]
    word_t[i][word_index] = 1
    print('{:2} {:4} {}'.format(i, word_index, word))

print(word_t.shape)
```

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMAN

Representing text: Example

- Continuing with the earlier *Notebook*:

```
 0 4167 michaelmas
 1  429 and
 2 6045 some
 3 4511 of
 4 3216 his
 5 5842 servants
 6  531 are
 7 6546 to
 8  728 be
 9 3409 in
10 6466 the
11 3253 house
12  981 by
torch.Size([13, 7278])
```

- At this point, tensor represents one sentence of length 13 in an encoding space of size 7,278, the number of words in our dictionary.

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMAN

Representing text: Analyzing available choices.

- The choice between **character-level** and **word-level** encoding leaves us to make a trade-off.

- In many languages, there are significantly fewer characters than words: representing characters has us representing just a few classes, while representing words requires us to represent a very large number of classes and, in any practical application, deal with words that are not in the dictionary.

- On the other hand, words convey much more meaning than individual characters, so a representation of words is considerably more informative by itself.

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMAN

Rupam

Representing text: Embedding

- One-hot encoding is a very useful technique for representing categorical data in tensors.
- However, as we have anticipated, one-hot encoding starts to break down when the number of items to encode is effectively unbound.
- How can we compress our encoding down to a more manageable size and put a cap on the size growth?
- Well, instead of vectors of many zeros and a single one, we can use vectors of floating-point numbers.
- A vector of, say, 100 floating-point numbers can indeed represent a large number of words. This is called an **embedding**.

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMAN

Representing text: Embedding

- To generate the embedding in such a way that **words** used in **similar contexts** mapped to **nearby regions** of the embedding.

- Let us consider an example regarding Embedding.

- We can generate a 2D space where axes map to **nouns**—fruit (0.0-0.33), flower (0.33-0.66), and dog (0.66-1.0)—and **adjectives**—red (0.0-0.2), orange (0.2-0.4), yellow (0.4-0.6), white (0.6-0.8), and brown (0.8-1.0).

- Our goal is to take actual fruit, flowers, and dogs and lay them out in the embedding.

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMAN

Representing text: Embedding

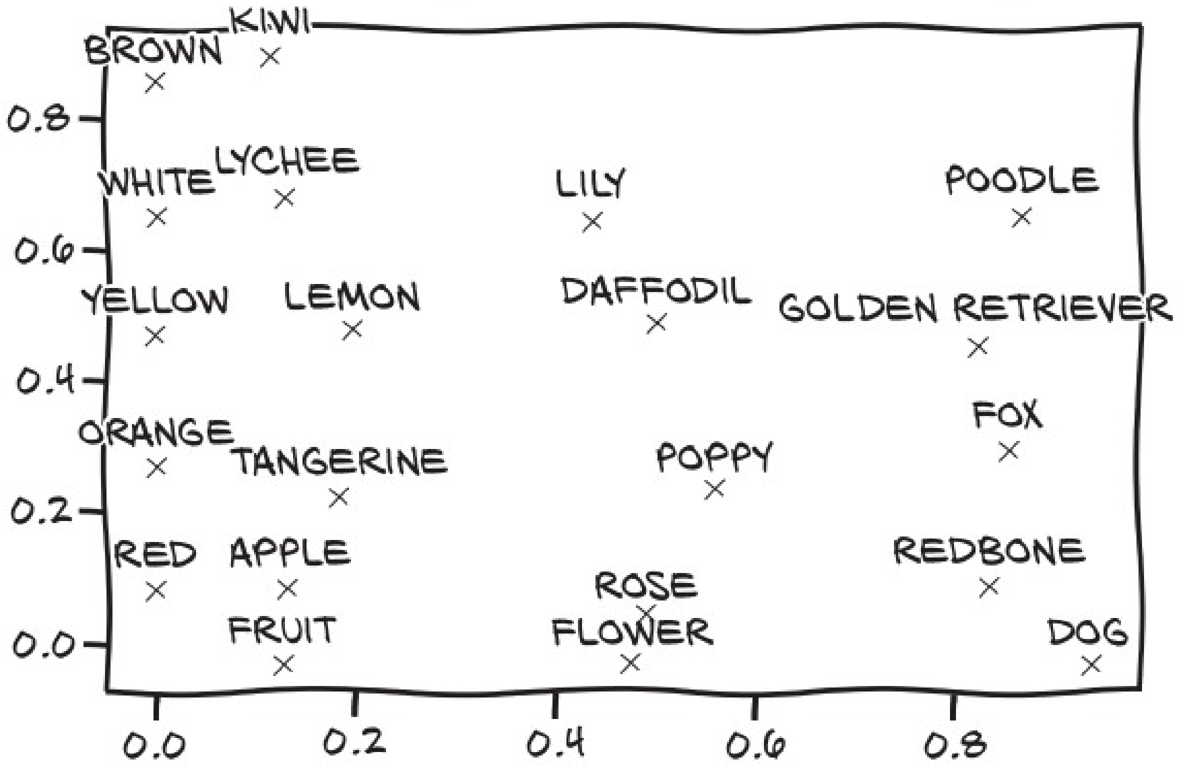

Figure 1: Manual Word Embedding

Slide Credit: E. STEVENS, L. ANTIGA, and T. VIEHMAN

References

- All the contents present in the slides are taken from various online resources. Due credit is given in the respective slides.These slides are used for *academic* purposes only.