DAT494_Lab4

November 17, 2024

1 Lab 4: Basic Text Sequence Analysis with Recurrent Nets

This lab will largely mimic our work in recent lectures.

Our goals are to:

1. Create a vocabulary and train an embedding matrix using the **continuous bag of words model**.
2. **Text Classification:** Load the works of Shakespeare as well as War and Peace by Leo Tolstoy, create a joint vocabulary, and then classify sentences as either Shakespeare or Tolstoy. We will use the embedding matrix trained in the first step.
3. Make a **character generation model** using the works of Shakespeare (so extremely similar to our War and Peace text generator).

We will use long short term memory (LSTM) cells for all our recurrent layers (we generally used GRU cells for the in-class examples)

You should be able to mostly follow the work presented in the recent notebooks to complete this lab.

[1]:
```
#Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#PyTorch Libraries
import torch
import torch.nn as nn

from torch.utils.data import Dataset, DataLoader

#You will also likely need:
import re
from collections import Counter

from torch.nn.utils.rnn import pad_sequence, pack_padded_sequence,␣
↪pad_packed_sequence
```

2 Part 1: Word Embeddings with Word2Vec CBOW

2.1 Load our texts

Load the Complete Works of Shakespeare and War and Peace by Leo Tolstoy texts by running the following cell:

[2]:
```
## Get War and Peace:
with open('war_and_peace.txt', 'r', encoding="utf8") as file:
    war_txt = file.read()

with open('shakespeare.txt', 'r', encoding="utf8") as file:
    shakespeare_txt = file.read()
```

2.2 Build Tokenizer

- Write a function named `tokenizer()` that takes a text as its input, and:
  1. Removes any non-alphanumeric characters *except for periods*
  2. Pads any periods with spaces so that they treated as separate tokens
  3. Lowers the text
  4. Returns the text split into tokens

Confirm that it seems to tokenize our texts properly by giving the output on at least one of the texts.

2.3 Build the Vocabulary

Now, build a master vocabulary that includes all words from both of our texts:

- This vocabulary should be a dictionary with the word the key and an (arbitrary) scalar the value.
- Keep only words that appear at least 10 times
- Include two special tokens for padding and unknown words
- Also create a reverse vocabulary that maps from a scalar to the associated word
- Save the vocabulary length as a variable
- Write a helper function that yields an appropriate list of scalars given an input text

Then:

1. Display the first few entries of both your vocabulary and reverse vocabulary to confirm construction.
2. Give the output of your helper function for the test sentence `"To be or not to asdf"`

2.4 Word2Vec: Continuous Bag of Words Model

Now, let's prep our data for the continuous bag of words model. Perform the following steps:

1. Convert the two texts to lists of scalars using the vocabulary and functions developed above, and concatenate these.
2. Using a window length of **5** (so, you will 5 words to the left and 5 words to the right of a target words, for a total of 10 context words per target word), construct all possible input/output pairs for a **continuous bag of words model** (you can ignore the transients at the beginning

and end of the combined text). Recall that you will have 10 context words as input for each target output word.
3. Confirm your work by displaying the first three input/output pairs: This will be three lists of 10 context words and an associated target. An example is given below.
4. Put your data into a Dataset (call it `CBOWDataset`) that returns a PyTorch input (vector of context words) and output (target word) for a given index.
5. Finally, create a DataLoader for this dataset with `shuffle=True` and a batch size of **64**. Confirm your work by displaying a random batch.

2.4.1 Example input/output pairs:

- You should get something with this format (examples shown either as scalars or as the corresponding text):

```
CONTEXT:  [3, 838, 861, 4491, 6, 2306, 1533, 6, 1723, 1]
 TARGET:  3

CONTEXT:  [838, 861, 4491, 6, 3, 1533, 6, 1723, 1, 28]
 TARGET:  2306

CONTEXT:  [861, 4491, 6, 3, 2306, 6, 1723, 1, 28, 4491]
 TARGET:  1533


CONTEXT:  the project gutenberg ebook of complete works of william <unk>
 TARGET:  the

CONTEXT:  project gutenberg ebook of the works of william <unk> this
 TARGET:  complete

CONTEXT:  gutenberg ebook of the complete of william <unk> this ebook
 TARGET:  works
```

2.5 Define and Train Continuous Bag of Words Model

Now, create the model, `CBOWModel`, inheriting from the `nn.Module` class. Recall the basic flow:

1. Embed inputs using `nn.Embedding` (equivalent to passing through linear layer with linear activation)
2. Sum over first dimension
3. Pass through output linear layer
4. (Softmax: can put in the model or handle as appropriate during training)

Your model should be declared with format:

```
class CBOWModel(nn.Module):

    def __init__(self, vocab_size, embedding_dim):
        ...
```

```
    def forward(self, x):
        ...
```

where `vocab_size` is the size of the vocabulary, and `embedding_dim` is the desired dimension of the embedding matrix.

Set `embedding_dim = 32`.

Training

- Create and train a model using an appropriate loss function and optimizer.
- Train until you begin to see meaningful relationships in the word embeddings, perhaps 5 epochs.

2.6 Explore Trained Embedding Matrix

Follow our work in class to determine the 10 most similar words to the following:

1. noble
2. war
3. sword
4. love
5. mother

Also, project all words into a two-dimensional space using t-SNE. Visualize the top 500 most common words. Comment briefly on any notable patterns you notice.

2.6.1 Example Results

The following are some example results for the 10 most similar words. Yours will likely vary, but you should observe at least some degree of semantic relatededness among the word lists.

```
noble
['honourable', 'worthy', 'good', 'great', 'royal', 'warlike', 'condemned',
'gracious', 'dread', 'princely']

war
['action', 'soil', 'purpose', 'chance', 'peace', 'pomp', 'quarrel', 'wars',
'malice', 'loss']

sword
['knife', 'arm', 'foot', 'body', 'horse', 'head', 'dog', 'dagger', 'finger',
'foe']

love
['praise', 'youth', 'desire', 'virtue', 'pity', 'bliss', 'joys', 'fortune',
'bounty', 'pride']

mother
['father', 'husband', 'sister', 'brother', 'wife', 'daughter', 'son', 'child',
```

```
'grave', 'aunt']
```

3 Part 2: Classify Sentences as Shakespeare or Tolstoy

3.1 Data for classification

We will want the following:

1. A list of all sentences in the Shakespearean text
2. A list of all sentences in War and Peace

Let's just naively use periods to divide sentences, and also enfore a minimum sentence length of three tokens, and a maximum length of 128 tokens.

- Use the vocabulary developed above, and construct the indicated lists (as lists of lists of scalars).
- Print out several sentences from each to confirm your work.

3.1.1 Some example sentences

A few examples as either lists of scalars or as words:

```
Shakespeare

[1537, 124, 17, 5712, 24, 1, 37, 151, 1769, 3531, 934, 42, 66, 62, 167, 21, 51,
70, 203, 128, 2384, 6, 50, 199, 960, 1, 16, 5, 26]

[28, 278, 7, 584, 21, 68, 43, 592, 657, 7, 530, 61, 203, 128, 4, 653, 914, 638]

[1, 149, 320, 4, 147, 47, 34, 8, 1217, 308, 69, 5, 4162, 3, 3329, 6, 50, 553,
149, 320, 14, 147, 50, 2247, 17, 553, 58, 3336, 14, 147, 149, 320, 3, 1072, 6,
10, 219, 14, 147, 183, 2809, 10, 40, 9, 23, 400, 453, 8, 4389, 127, 7716, 15,
5156, 122, 21, 3, 1, 183, 10, 6187, 1108, 4, 480, 9, 23, 50, 199, 2695, 559]

until life s composition be <unk> by those swift messengers returned from thee
who even but now come back again assured of thy fair health <unk> it to me

this told i joy but then no longer glad i send them back again and straight grow
sad

<unk> mine eye and heart are at a mortal war how to divide the conquest of thy
sight mine eye my heart thy picture s sight would bar my heart mine eye the
freedom of that right my heart doth plead that thou in him dost lie a closet
never pierced with crystal eyes but the <unk> doth that plea deny and says in
him thy fair appearance lies


Tolstoy
```

```
[4237, 10, 1783, 20, 26, 4, 7, 55, 339, 24, 30, 125, 4193, 1076, 1, 15, 64,
2468, 22, 8, 1068, 1890, 6, 149, 3214, 9, 13, 2230]

[38, 19, 726, 4, 6, 65, 1060, 4, 10, 17, 35, 7, 439]

[4, 15, 3, 7783, 4, 1356, 289, 3370, 5, 23, 12, 1285, 3, 620, 6, 1024, 17, 135,
5, 13, 605, 1169, 16, 4, 1, 16, 5, 4, 6894, 22, 12, 354, 203, 9, 13, 3943, 394,
9, 158, 1623]

arrange that affair for me and i shall always be your most devoted slave <unk>
with an f as a village elder of mine writes in his reports

she is rich and of good family and that s all i want

and with the familiarity and easy grace peculiar to him he raised the maid of
honor s hand to his lips kissed it and <unk> it to and fro as he lay back in his
armchair looking in another direction
```

3.2 Dataset + Collate Function

- Now, construct a Dataset that consists of all sentences with a corresponding label (use 0 for Tolstoy, 1 for Shakespeare)
- Split this data into 80% training and 20% testing set. Further split 10% of the training set off for validation.
- Also, construct a collate function that returns a batch of padded sequences (since our sentences will all be different lengths), the associated labels, as well as the unpadded sentence lengths
- Finally, use your Datasets and collate function to create training, validation, and testing DataLoaders. For the training and validation loaders, use a batch size of 64 and `shuffle=True`

3.3 Classification Model

Construct a classification model with the following characteristics:

1. Pass input through an embedding layer (with dimension 32)
2. Pass through a recurrent component consisting of **two hidden layers of LSTMs** with **64 neurons each**,
3. Pass the **final hidden state** a final fully connected component with a single output neuron
4. (Sigmoid layer; you may put the sigmoid activation function in the model or deal with it separately)

Create your model class with the format:

```
class ClassificationModel(nn.Module):

    def __init__(self, vocab_size, embedding_dim):

    def forward(self, x, lengths):
        ...
```

3.3.1 Construct and Train

Initialize a model: **Use the trained embedding matrix from your CBOW model as the initial weights for the embedding layer**

Once you've created the model, use an appropriate loss function and an Adam optimizer with a learning rate of `.001` to train the model for a single epoch; for each training iteration, determine the model accuracy on the training batch as well as a random validation batch.

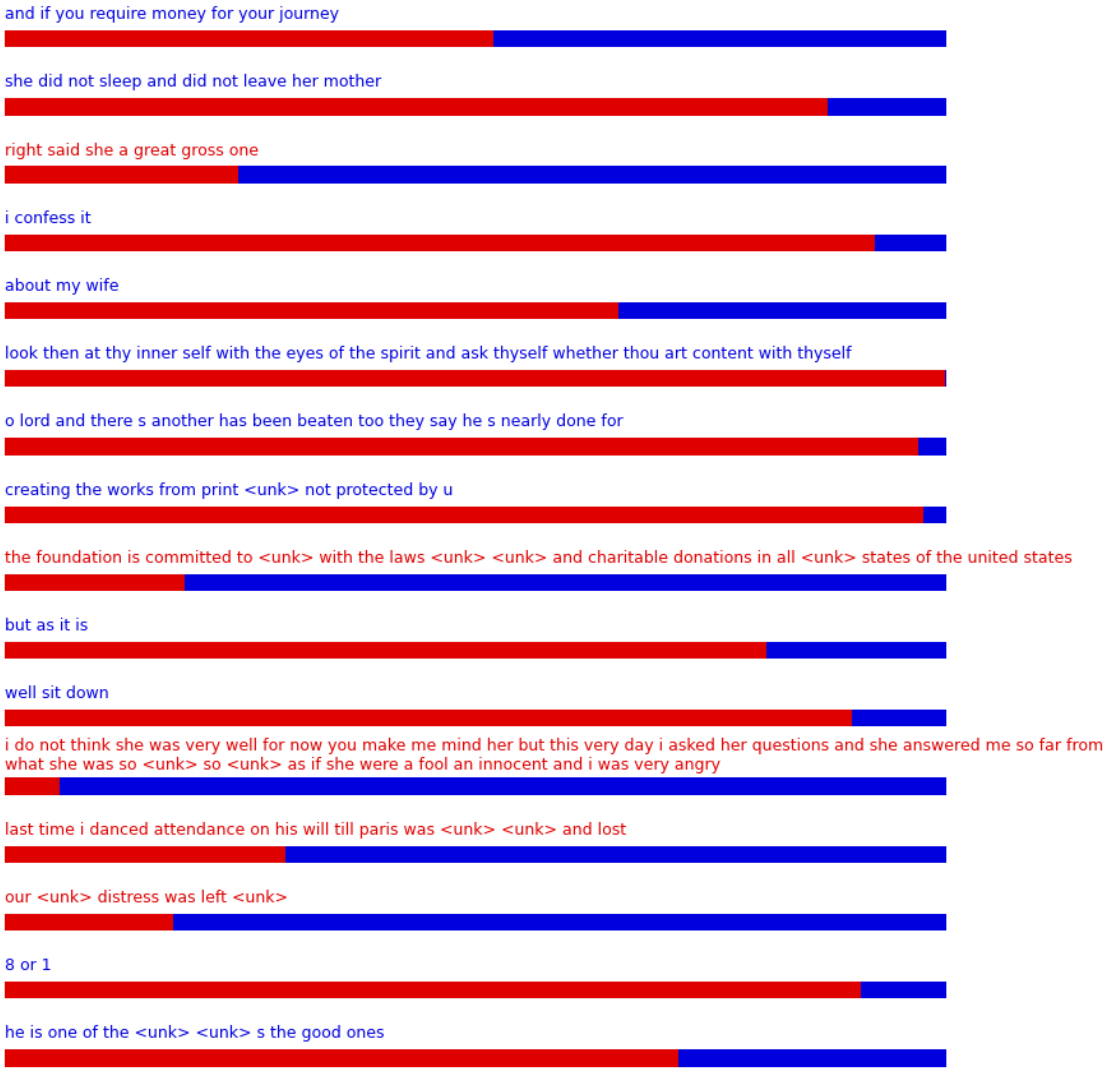
- Plot the training and validation accuracy as a function of batch number. You should see both accuracies rapidly approach and oscillate around 95% or more.
- Also determine the model accuracy on the testing dataset. It should be over 95%.

3.3.2 Visualize some predictions

Take 16 random sentences from the testing dataset and print, along with the author source and the model-predicted probabilities they are Tolstoy vs. Shakespeare. The following is an example using horizontal bar plots to visualize this probability, where: 1. Blue text indicates a Tolstoy sentence; Red text indicates a Shakespeare sentence 2. The width of the blue bar is proportional to the model-predicted probability the sentence is Tolstoy, and a similar pattern holds for the red bar and Shakespeare.

3.3.3 Visualize some incorrect predictions

Do the same thing as above, but now give 16 examples where the model gave the incorrect (binary) prediction. An example follows.



4 Part 3: Shakespearean Text Generator

4.0.1 1. Create a new vocabulary consisting of the unique characters in our Shakespearean text

- Tokenize texts to individual characters
- Determine all unique characters
- Create a vocabulary dictionary that maps individual characters to an arbitrary scalar
- Also create a reverse vocabulary dictionary that maps from the scalar to the character
- Save the vocabulary size

4.0.2 2. Prepare data for character prediction

- For our model, we want sequences of length 128 as input, and length 128 as output

- That is, we will make a sequence-to-sequence model, with the ouput the input shifted by 1 word (plus the next word)
- To prep our data, construct all possible sub-sequences of length 129 (the first 128 characters will be the input, the last 128 the output)
- Wrap these sequences in a PyTorch `DataSet` that returns an input and output sequence as torch tensors
- Also create a DataLoader with batch size 64

4.1 Character Prediction Model

Now, create a sequence-to-sequence model that predicts an output sequence from an input sequence (again, the output is the input shifted by one to the right).

The model should have the following structure:

1. Embedding layer
2. Recurrent component consisting of two **LSTM** layers with 64 neurons each, **as well as 20% dropout**
3. An output layer with `vocab_size` neurons, where this is the size of your vocabulary

Call this model `ShakespeareModel()`, and define using the format:

```
class ShakespeareModel(nn.Module):

    def __init__(self, vocab_size, embedding_dim, rnn_size=64):
        ...

    def forward(self, x, hidden, cell):
        ...

    #OPTIONALLY:
    def init_hidden(self, batch_size):
        ...
```

Use an embedding matrix dimension of 32.

4.1.1 Train the model

Now, using an appropriate loss function and optimizer, train the model until, in your judgement, you get reasonable results when creating Shakespearean text autoregressively.

4.2 Autoregressively Generate Text

Using a starting sentence seed, generate 1,000 characters autoregressively:

1. Given the logits for the next character, use a `Categorical` object to draw the next letter from a categoral distribution. Additionally, you can multiply the logits by a scaling factor to make the text more or less random.
2. Give three example results from using the seed sentence "Cowards die many times before their deaths; the valiant never taste of death but once"

One example follows.

Cowards die many times before their deaths; the valiant never taste of death but oncell thou art the street of truth.

ANTONIO.
I may be extray, and make make to your more mouth take the see of the past,
And the new my soldiers you the well be bones. A wair.

[_Exeunt Prace be Clocess Soldiers.

KING HENRY.
That stand stand behild the times, and like the some constray
To a deep of the tall a nefford.

[_Exeunt._]

SCENE II. Countrunis to the fallow.

HAMLET.
I have thou she heaven so.

KING RICHARD.
Why of the which I have the distress, my stands
To be she hear me for the second them to be more his death.

KING RICHARD.
O mean every would their thenerved with my chance.

MESSENGER.
He do not show in the countered and weep
The scan the heaven, and heaven then? When I hold, my lord, thou thinks, but hear thee I must more of destrike behindness for your been day.

[_Exeunt._]

SCENE II. Henry, and Porace at him and Lisider and Nends of Marior.

SILVIA.
That so the rather to the one hath love my false.

KING.
What shall shall you must danger, and me
keep the she with the