Instructor: Vishnu Boddeti
CSE 891-001: Deep Learning
Programming Homework 1
October, 16

Instructions:

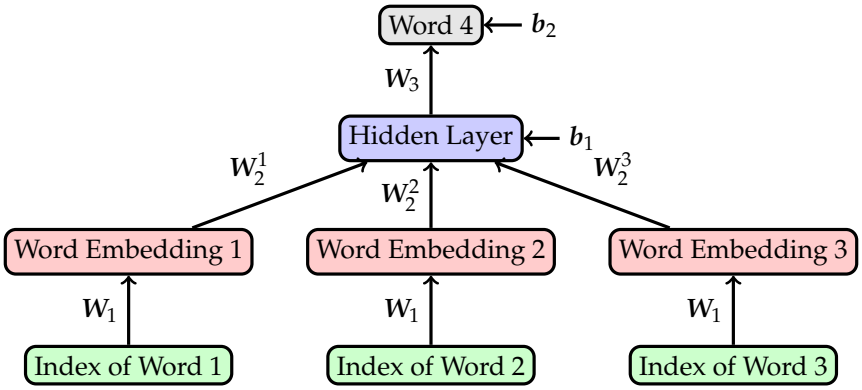
- **Submission:** Only homeworks uploaded to Google Classroom will be graded. Submit solutions in PDF format.
- **Integrity and Collaboration:** You are expected to work on the homeworks by yourself. You are not permitted to discuss them with anyone except the instructor. The homework that you hand in should be entirely your own work. You may be asked to demonstrate how you got any results that you report.
- **Clarifications:** If you have any question, please look at Google Classroom first. Other students may have encountered the same problem, and is solved already. If not, post your question there. We will respond as soon as possible.

In this assignment we will train a multi-layered perceptron for natural language processing. We will consider the task of predicting the next word in a sentence given a sequence of words. You will implement the backpropagation computations for the neural network and analyze the resulting learned word representations. The amount of code you have to write is very short – about 10 lines – but you need to think very carefully to write each line. You will need to first derive the updates mathematically and then implement them using matrix-vector operations in PyTorch.

0 Code and Software Setup
The first step is to install Python, PyTorch and other required libraries. After you install Python, see the instructions on this web page to install PyTorch (`https://pytorch.org`). PyTorch provides a tensor library just like Numpy, the code will be based on PyTorch, you are encouraged to look at the documentation.

1 Network Architecture (1pts)

You will train a neural language model using a multi-layered perceptron like the one below. It receives 3 consecutive words as the input and aims to predict a distribution over the next word. We will train the model using the cross-entropy criterion, which is equivalent to maximizing the probability it assigns to the *target* words in the training set. Hopefully, it will also learn to make sensible predictions for sequences it hasn't seen before.



The network consists of an input layer, embedding layer, hidden layer and output layer. The input consists of a sequence of 3 consecutive words, provided as integer valued indices, i.e., the 250 words in our dictionary are arbitrarily assigned a unique integer between 0 and 249. The embedding layer maps each word to its corresponding vector representation. This layer has $3 \times d$ units, where d is the embedding dimension, and functions as a look-up table. We will share the same look-up table for all the 3 positions, so we will learn a single common word embedding matrix for each context position. The embedding layer is connected to the hidden layer, which uses a sigmoid loss activation function. The hidden layer is connected to the output layer, and the output layer is a softmax over the 250 words in our dictionary.

1. The trainable parameters of the model consist of 3 weight matrices and two bias vectors. Assuming that we have 250 words in the dictionary, use three words as our input context, a 16-dimensional word embedding and a hidden layer with 128 units. What is the total number of trainable parameters in the model? Which part of the model has the highest number of parameters?

Starter Code and Data
Look at the sentences in `raw_sentences.txt`. It contains the sentences that we will be using for this assignment. These sentences are fairly simple ones and cover a vocabulary of only 250 words.
We extracted the sequence of *input* and *target* words from this dataset and divided them into training, validation and test sets. To inspect this data, run the following within IPython.

```
import _pickle
f = open('data.pk', 'rb')
data_obj = _pickle.load(f, encoding='latin1')
```

Here `data_obj` is a Python dictionary which contains the vocabulary, as well as the inputs and targets for all the three data splits. `data['vocab']` is a list of the 250 words in the dictionary. `data['train_inputs']` is a 372,500 $\times$ 3 matrix with each row corresponding to the indices of the 3 context words. `data['train_targets']` is a vector providing the index of the target word for each of the training sample. The validation and test splits also follow the same structure.

The starter code for this assignment contains the following files:

- `main.py`: The main file that you need to run to train and evaluate your language models. This file also defines the model that you will be training.
- `data.py`: This file loads the data and provides a batch of data, given the batch size. Randomly shuffling the data after each epoch is also taken care of in this file.
- `optim.py`: Different optimization routines are implemented in this file. Each method is implemented as a class with same API.
- `layer.py`: Different types of layers in our neural network are implemented here. Each layer is implemented as a class, and each class consists of three functions, `forward`, `backward` and `zero_grad`. You will be implementing the backward functions for all the layers that we need for the language model.

2 Training the Model (5pts) In the first part of the assignment, you will implement the methods to compute the gradients of the parameters of the model using backpropagation. In order to implement backpropagation efficiently, you need to express the computations in terms of matrix operations, rather than `for` loops. You should compute the derivatives on pencil and paper using chain rule and then express them in terms of matrix operations. Check out the `forward` functions within `layer.py` to understand how each layer can be implemented using matrix operations.

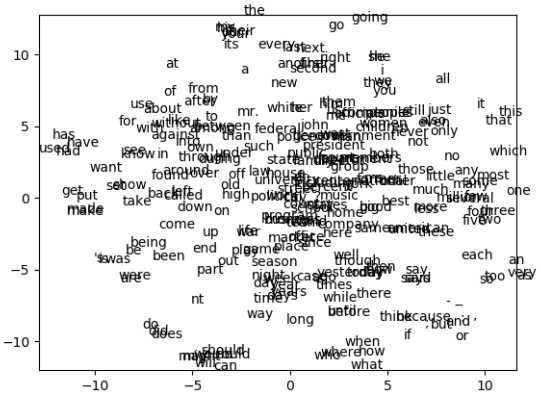
Once you have implemented the gradient computation, you will need to train the model. The training procedure is implemented in `main.py`. Be sure to set the appropriate parameter values in `args`. As the model trains it will print the aggregate loss and accuracy of the trained model at the end of each epoch.

To demonstrate that you have correctly implemented the gradient computation, please include the following with your assignment submission:

- You will submit the `layer.py` file, you do not need to modify any of the code except the parts that we have asked you to implement.
- We will check your gradients on the first batch of 100 training samples using the provided semi-trained model.

3 Analysis (4pts) In this part, you will analyze the trained language representations. You should first train a model with a 16-dimensional embedding and 128 hidden units, as discussed in the previous question. You will use this trained model for the remainder of this section. Be sure to load your trained model weights for this analysis. The following methods can be used to analyze the model after the training is done:

- `tsne_plot`: Creates a 2-D plot of the distributed representation space using an algorithm called t-SNE (you do not know what this is for the assignment, you are encouraged to read more about it on your own). Nearby points in the 2-D space are meant to correspond to nearby points in the 16-D word embedding space. From the learned model, you can create pictures of the kind shown below.



- `display_nearest_words`: Shows the words whose embedding vectors are nearest to the given word
- `word_distance`: Computes the distance between the embeddings of two given words.
- `predict_next_word`: Shows the most likely words as predicted by the model, along with their probabilities.

Using these methods, answer the following questions, each of which is worth 1 point.

1. Pick three words from the vocabulary that go well together (for instance, `'government of united'`, `'city of new'`, `'life in the'`, `'he is the'` etc.). Use the model to predict the next word. Does the model give sensible predictions? Try to find an example where it makes a plausible prediction even though the 4-gram wasn't present in the dataset (`raw_sentences.txt`).
2. Plot the 2-D visualization using the method `Model.tsne_plot.py`. Look at the plot and find a few clusters of related words. What do the words in each cluster have in common?
3. Are the words `'new'` and `'york'` close together in the learned representation? Why or why not?
4. Which pair of words is closer together in the learned representation: `('government', 'political')` or `('government','university')`? Why do you think that is?

Submission Here is everything you need to submit.

- A PDF file titled `programming-assignment-1-{msunetid}.pdf` containing the following:
  - The answer for Q1.
  - Answers to all questions in Q3.
  - Optional: What part of this homework was the most illuminating, and the part that was difficult?
- Your code file `layer.py`