#######################################

Neural Networks for Machine Learning

Programming Assignment 2

Learning word representations.

#######################################

In this assignment, you will design a neural net language model that will

learn to predict the next word, given previous three words.

The data set consists of 4-grams (A 4-gram is a sequence of 4 adjacent words

in a sentence). These 4-grams were extracted from a large collection of text.

The 4-grams are chosen so that all the words involved come

from a small vocabulary of 250 words. Note that for the purposes of this

assignment special characters such as commas, full-stops, parentheses etc

are also considered words. The training set consists of 372,550 4-grams. The

validation and test sets have 46,568 4-grams each.

### GETTING STARTED. ###

Look at the file raw\_sentences.txt. It contains the raw sentences from which

these 4-grams were extracted. Take a look at the kind of sentences we are

dealing with here. They are fairly simple ones.

To load the data set, go to an octave terminal and cd to the directory where the

downloaded data is located. Type

> load data.mat

This will load a struct called 'data' with 4 fields in it.

You can see them by typing

> fieldnames(data)

'data.vocab' contains the vocabulary of 250 words. Training, validation and

test sets are in 'data.trainData', 'data.validData' and 'data.testData' respectively.

To see the list of words in the vocabulary, type -

> data.vocab

'data.trainData' is a matrix of 372550 X 4. This means there are 372550

training cases and 4 words per training case. Each entry is an integer that is

the index of a word in the vocabulary. So each row represents a sequence of 4

words. 'data.validData' and 'data.testData' are also similar. They contain

46,568 4-grams each. All three need to be separated into inputs and targets

and the training set needs to be split into mini-batches. The file load\_data.m

provides code for doing that. To run it type:

>[train\_x, train\_t, valid\_x, valid\_t, test\_x, test\_t, vocab] = load\_data(100);

This will load the data, separate it into inputs and target, and make

mini-batches of size 100 for the training set.

train.m implements the function that trains a neural net language model.

To run the training, execute the following -

> model = train(1);

This will train the model for one epoch (one pass through the training set).

Currently, the training is not implemented and the cross entropy will not

decrease. You have to fill in parts of the code in fprop.m and train.m.

Once the code is correctly filled-in, you will see that the cross entropy

starts decreasing. At this point, try changing the hyperparameters (number

of epochs, number of hidden units, learning rates, momentum, etc) and see

what effect that has on the training and validation cross entropy. The

questions in the assignment will ask you try out specific values of these.

The training method will output a 'model' (a struct containing weights, biases

and a list of words). Now it's time to play around with the learned model

and answer the questions in the assignment.

### DESCRIPTION OF THE NETWORK. ###

The network consists of an input layer, embedding layer, hidden layer and output

layer. The input layer consists of three word indices. The same

'word\_embedding\_weights' are used to map each index to a distributed feature

representation. These mapped features constitute the embedding layer. This layer

is connected to the hidden layer, which in turn is connected to the output

layer. The output layer is a softmax over the 250 words.

### THINGS YOU SEE WHEN THE MODEL IS TRAINING. ###

As the model trains it prints out some numbers that tell you how well the

training is going.

(1) The model shows the average per-case cross entropy (CE) obtained

on the training set. The average CE is computed every 100 mini-batches. The

average CE over the entire training set is reported at the end of every epoch.

(2) After every 1000 mini-batches of training, the model is run on the

validation set. Recall, that the validation set consists of data that is not

used for training. It is used to see how well the model does on unseen data. The

cross entropy on validation set is reported.

(3) At the end of training, the model is run both on the validation set and on

the test set and the cross entropy on both is reported.

You are welcome to change these numbers (100 and 1000) to see the CE's more

frequently if you want to.

### SOME USEFUL FUNCTIONS. ###

These functions are meant to be used for analyzing the model after the training

is done.

display\_nearest\_words.m : This method will display the words closest to a

given word in the word representation space.

word\_distance.m : This method will compute the distance between two given

words.

predict\_next\_word.m : This method will produce some predictions for the next

word given 3 previous words.

Take a look at the documentation inside these functions to see how to use them.

### THINGS TO TRY. ###

Choose some words from the vocabulary and make a list. Find the words that

the model thinks are close to words in this list (for example, find the words

closest to 'companies', 'president', 'day', 'could', etc). Do the outputs make

sense ?

Pick three words from the vocabulary that go well together (for example,

'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?

Which words would you expect to be closer together than others ? For example,

'he' should be closer to 'she' than to 'federal', or 'companies' should be

closer to 'business' than 'political'. Find the distances using the model.

Do the distances that the model predicts make sense ?

You are welcome to try other things with this model and post any interesting

observations on the forums!