Demo 1: Deep Learning for NLP - Word Embeddings

REPORT PREPARED BY,

- Prashanth ASOK KUMAR
- Annanya CHITRA KANNAN

Demo 1: Deep Learning for NLP - Word Embeddings

REPORT OF ANSWERS

I. Train word embeddings

Exercice 1: Train two models for each dataset, with a vector dimension of 50, 5 iteration of training and a window size of 15.

From the above question we are able to understand that we need to make use of the fasttext to train model with the specified conditions:

- Dimension: implies the option "-dim"
- iteration of training: implies the option "-epoch"
- window size: implies the option "-ws"

Q1: Train a cbow model, explain your command line (1pt)

To train a cbow model with the requested conditions, we will be making use of the below command line:

```
fasttext cbow -input text8/text8.txt -output output/text8_cbow -dim 50 -ws 15 -epoch 5
```

where:

- the text followed by -input represents the data to be used for training the model
- the text followed by -output represents the path and the filenaming to be used for the output file.
- -dim 50 sets the dimension to 50, which is the dimension of the vector to be created upon reading the file.
- -ws 15 sets the window size to 15, which is the number of words the fasttext can lookup in both direction of a text in a line.

Q2: Train a skip-gram model, explain your command line (1pt)

To train a skip-gram model with the requested conditions, we will be making use of the below command line:

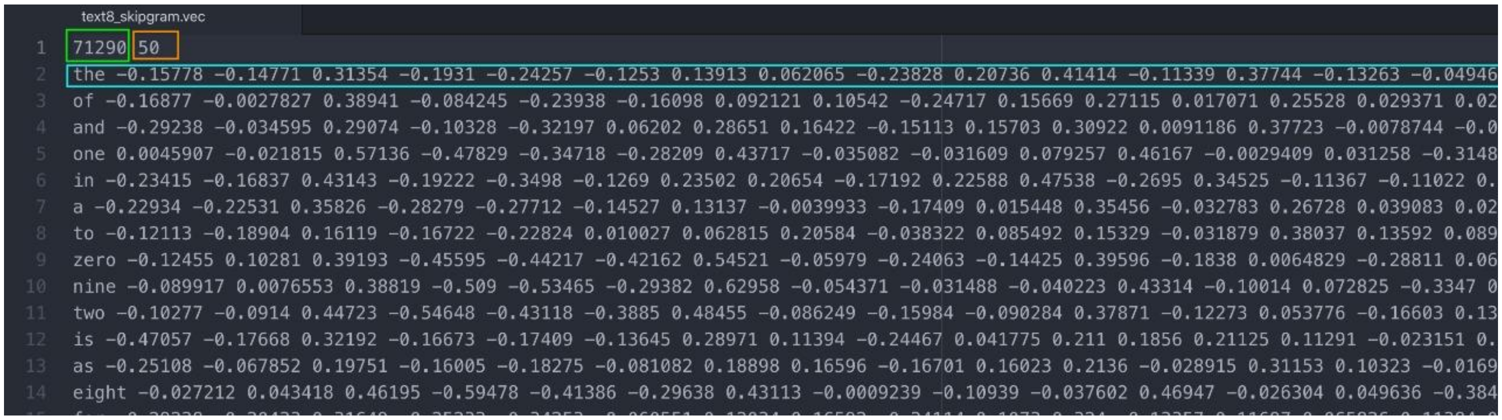
```
fasttext skipgram -input text8/text8.txt -output output/text8_skipgram -dim 50 -ws 15 -epoch 5
```

where:

- the text followed by -input represents the data to be used for training the model
- the text followed by -output represents the path and the filenaming to be used for the output file.
- -dim 50 sets the dimension to 50, which is the dimension of the vector to be created upon reading the file.
- -ws 15 sets the window size to 15, which is the number of words the fasttext can lookup in both direction of a text in a line.

<u>**Q3:** Give example of the vector file, describe it (1pt)</u>

- The vector file is one of the files generated as the result of training Fasttext model with an input file.
- The first line of the vector file contains the number of words in the file followed by the dimension of the vector present in it.
- The rest of the lines contain the word vectors of all the words in the input file, each line contains a word vector and they are arranged in a descending of their frequency.


text8_skipgram.vec

```
71290 50
the -0.15778 -0.14771 0.31354 -0.1931 -0.24257 -0.1253 0.13913 0.062065 -0.23828 0.20736 0.41414 -0.11339 0.37744 -0.13263 -0.04946
of -0.16877 -0.0027827 0.38941 -0.084245 -0.23938 -0.16098 0.092121 0.10542 -0.24717 0.15669 0.27115 0.017071 0.25528 0.029371 0.02
and -0.29238 -0.034595 0.29074 -0.10328 -0.32197 0.06202 0.28651 0.16422 -0.15113 0.15703 0.30922 0.0091186 0.37723 -0.0078744 -0.0
one 0.0045907 -0.021815 0.57136 -0.47829 -0.34718 -0.28209 0.43717 -0.035082 -0.031609 0.079257 0.46167 -0.0029409 0.031258 -0.3148
in -0.23415 -0.16837 0.43143 -0.19222 -0.3498 -0.1269 0.23502 0.20654 -0.17192 0.22588 0.47538 -0.2695 0.34525 -0.11367 -0.11022 0.
a -0.22934 -0.22531 0.35826 -0.28279 -0.27712 -0.14527 0.13137 -0.0039933 -0.17409 0.015448 0.35456 -0.032783 0.26728 0.039083 0.02
to -0.12113 -0.18904 0.16119 -0.16722 -0.22824 0.010027 0.062815 0.20584 -0.038322 0.085492 0.15329 -0.031879 0.38037 0.13592 0.089
zero -0.12455 0.10281 0.39193 -0.45595 -0.44217 -0.42162 0.54521 -0.05979 -0.24063 -0.14425 0.39596 -0.1838 0.0064829 -0.28811 0.06
nine -0.089917 0.0076553 0.38819 -0.509 -0.53465 -0.29382 0.62958 -0.054371 -0.031488 -0.040223 0.43314 -0.10014 0.072825 -0.3347 0
two -0.10277 -0.0914 0.44723 -0.54648 -0.43118 -0.3885 0.48455 -0.086249 -0.15984 -0.090284 0.37871 -0.12273 0.053776 -0.16603 0.13
is -0.47057 -0.17668 0.32192 -0.16673 -0.17409 -0.13645 0.28971 0.11394 -0.24467 0.041775 0.211 0.1856 0.21125 0.11291 -0.023151 0.
as -0.25108 -0.067852 0.19751 -0.16005 -0.18275 -0.081082 0.18898 0.16596 -0.16701 0.16023 0.2136 -0.028915 0.31153 0.10323 -0.0169
eight -0.027212 0.043418 0.46195 -0.59478 -0.41386 -0.29638 0.43113 -0.0009239 -0.10939 -0.037602 0.46947 -0.026304 0.049636 -0.384
```

II. Loading word embeddings

Loading the WE is not difficult in itself. To make computations fast, we store the whole set of embeddings in a numpy array of shape (num words, dimensions).

We also build a dictionary (vocab) for mapping words to row index in this array, and another (rev vocab) for mapping indexes back to word forms.

```
def load(filename):
    vocab = {}
    rev_vocab = []
    lines = open(filename).readlines()
    vectors = np.zeros((int(lines[0].split()[0]), int(lines[0].split()[1])))
    for i, line in enumerate(lines):
        tokens = line.strip().split()
        if (i > 0) :
            vocab[tokens[0]] = i-1
            rev_vocab.append(tokens[0])
            vectors[i-1] = [float(value) for value in tokens[1:]]
    return vocab, rev_vocab, vectors
```

Exercice 1: Loading

<u>**Q1:** write in Python the necessary script to load the WE (1pt)</u>

- We can load the WE and work with it by making use of the generated vector file.
- With the help of the provided function "load" we can load the WE by feeding the vector filename (along with the path) and get the resultant variables with which we can perform the word embedding tasks as expected.

<u>**Q2:** what "vocab", "rev_vocab" and "vectors" stand for? (1pt)</u>

- ***vocab* -** is a dictionary which has the words present in the vector file has keys and its corresponding row number in the "vectors" variable as the value
- ***rev_vocab* -** is a dictionary which has the words present in the vector file has values and its corresponding row number in the vectors variable as key. It is an inverse of key value pairs of the "vocab" dictionary.
- ***vectors* -** is an array variable which has the vector value of the words in the vector file. Each row in this array denotes the vector form of a word.

Exercice 2: Compute the cosine similarity for each model and each datasets

<u>**Q1:** Explain what is the cosine similarity. (1pt)</u>

- Cosine Similarity is the method used to identify how much a vector is similar to the other vector, irrespective of it's magnitude.
- This is used in identifying how much a word or document when represented as a vector is similar to the other word or document.

Q2: How to compute it in python using numpy? and using scipy? (2pt)

Using Numpy:

We can calculate cosine similarity by making use of numpy.dot() function to find the dot product of the two vectors and dividing the result by the product of individual vector's normalized value with the help of numpy.linalg.norm() function

Cosine Similarity= numpy.dot(vector1, vector2)/ numpy.linalg.norm(vector1) numpy.linalg.norm(vector2)*

Using Scipy:

We can calculate the cosine similarity by making use of the function scipy.spatial.distance.cosine() to get the distance and subtracting this value from 1 to get the cosine similarity.

Cosine Similarity= 1- scipy.spatial.distance.cosine(vector1, vector2)

Q3: What is the cosine similarity between the vectors representing "dog" and "cat", and what about "dog" and "dentist"? (1pt)

Using test8.txt skip-gram model with dim 50,

- Cosine Similarity for dog and cat is **0.8150036990132189**
- Cosine Similarity for dog and dentist is **0.4987285494582308**

Q4: What is the closest word to "bank": "river" or "trade"? (1pt)

- Cosine Similarity of bank and river is 0.5072638871962184
- Cosine Similarity of bank and trade is 0.766917610616357
- Hence, the closest Word to bank is **trade**

III. Closest words

Now you have your WE model, you will use a tric to compute all the closest words.
This can be done once for all, by computing the dot product between he matrix containing all vectors and the transpose of the target word vector (np.dot(vectors, v.T)).
Then we can use some numpy trick to recover the indices of the n highest scores.

```
def closest(vectors, vector, n=10):
    n=n+1
    scores = np.dot(vectors, vector.T)
    indices = np.argpartition(scores, -n)[-n:]
    indices = indices[np.argsort(scores[indices])]
    output = []
    for i in [int(x) for x in indices]:
        output.append((scores[i], i))
    return reversed(output)
```

Exercice 1: Code the function

<u>**Q1:** What pre-process to all the vectors do you NEED to do before using the dot product instead of the cosine? (3pt)</u>

- We need to normalize the vectors before using the dot product instead of cosine.
- We can check this by making use of numpy.allclose() function to check if the dot product of vectors and cosine of the vectors are same. We will get the result as true when we are making use of the normalized vectors for the dot products.
- Consider we have 2 vectors A and B which are not normalized.
- If we make use of numpy.allclose() function to compare dot product and cosine, we will see that the result is false.
    - i.e. numpy.allclose(numpy.dot(A, B.T), (1-scipy.spatial.distance.cosine(A, B))) --> False

- But when we normalize these vectors A and B before feeding it to numpy.dot() function we will see that we have similar value to cosine.
    - i.e.
        - normalizedA= A/numpy.linalg.norm(A)
        - normalizedB= A/numpy.linalg.norm(B)
        - numpy.allclose(numpy.dot(normalizedA, normalizedB.T), (1-scipy.spatial.distance.cosine(A, B))) --> True

<u>**Q2:** What "argpartition" stand for? (1pt)</u>

- "argpartition" means partition the arguments provided, it will partition the input array based on the index value provided such a way that arguments smaller than the index element are moved before this element and all equal or greater are moved behind it.
- It returns the index value of the array based on the partition.

<u>**Q3:** Add Comments to the code. (1pt)</u>

```
def closest(vectors, vector, n=10):
    n=n+1 # it is kth value to be fed to the argpartition
    scores = np.dot(vectors, vector.T)
    # performs dot product between the vectors and store the result in the array "scores"
    indices = np.argpartition(scores, -n)[-n:]
    # make use of argpartition on the array "scores" and kth value as n.
    # The resultant index of the partitioned array is assigned to indices array in the reverse order from the index
value of n.
    indices = indices[np.argsort(scores[indices])]
    #sorts each list in the indices array based on the scores array values.
    output = []
    for i in [int(x) for x in indices]:
        output.append((scores[i], i)) #each scores value and it's corresponding index is mapped to the list output
    return reversed(output) #output list is sorted in descending order based on the scores value in it.
```

Exercice 2: Analysis

Q1: What are the closest words to "apple"? (1pt)

- The closest words to "apple" are,
    - apple
    - macintoshes
    - macintosh
    - applesoft
    - appleworks
    - amiga
    - amigabasic
    - visicalc
    - applescript
    - wordperfect
    - workstation

Q2: What about other words (close to "apple")? (1pt)

- The other words close to "apple" are,
    - scp
    - workbench
    - ported
    - compaq
    - powerpc
    - microsoft
    - workstations
    - dreamcast
    - iic

- windows

<u>**Q3:** Can you find words which have a strange neighborhood? (1pt)</u>

<u>**Q4:** Check your answers using scipy.spatial.distance.cosine regarding the scores obtained what can you conclude about the dot product?</u>

IV. Analogy

Word analogies can be exposed by translating a word vector in a direction that corresponds to a linguistic or semantic relationship between two other words.
So if we have w1 and w2 in relation R(w1,w2), we can compute the relation r=vec(w2)-vec(W1), and then apply this relation to the vector of another word vec(w3).
The word closest to vec(w3)+r should also exhibit the same relation.
Therefore, the idea is to use the closest function to find vectors similar to vec(w2)-vec(w1)+vec(w3).

Exercice 1: Code the analogy function

<u>**Q1:** Can you use the previous code? How far? (1pt)</u>

- We can use the previous code to get the list of words which are close to the required output.
- We could only be able to get the list of closest words and not the exact answer from it.
- Also, getting exact answer depends upon the training model, size of the window screen and the dimension of the vector used while training the fasttext model.

Exercice 2: Solve the analogies

<u>**Q1:** "paris" is to "france" what "delhi" is to ... ? (1pt)</u>

- Using test8.txt skip-gram model with dim 50, we received the following list of closest words,
    - delhi
    - mysore
    - india
    - gujarat
    - jayavarman
    - vijayanagara
    - vijayanagar
    - punjab
    - maratha
    - qinghai
    - warangal
- We could see that, the expected result “india” is among this list.

Q2: "gates" - "microsoft" + "apple" = ... ? (1pt)

- Using test8.txt skip-gram model with dim 50, we received the following list of closest words,
    - gates
    - gate
    - fountain
    - invents
    - mansion
    - delicatessen
    - menor
    - doorstep
    - bearden
    - storey
    - beard

Q3: "king" - "man" + "woman" = ... ? (1pt)

- Using test8.txt skip-gram model with dim 50, we received the following list of closest words,
    - regnant
    - daughters
    - eldest
    - llywelyn
    - concubine
    - prince
    - queen
    - coemperor
    - betrothed
    - yolande
    - throne
- We could see that, the expected result "queen" is among this list.

Q4: "slow" - "slower" + "fast" = ... ? (1pt)

- Using test8.txt skip-gram model with dim 50, we received the following list of closest words,
    - fast
    - stinking
    - relaying
    - off
    - relive
    - dispatching
    - ready
    - devasting
    - deadly
    - attack
    - slow

Exercice 3: Bonus

<u>**Q1:** Augment the dimension of the WE to 300 and retrain them (1pt)</u>

- Using test8.txt skip-gram model with dim 300, we received the following list of closest words,
  - paris,france,delhi
    - delhi
    - india
    - warangal
    - hyderabad
    - indian
    - shahi
    - haryana
    - lahore
    - subcontinent
    - mughal
    - ajaigarh
  - microsoft,gates,apple
    - gates
    - apple
    - apples
    - at
    - gated
    - shines
    - grated
    - gate
    - with
    - dames
    - ran
  - man,king,woman
    - king
    - kings
    - noblewoman
    - woman
    - stepdaughter
    - daughter
    - marries
    - queen
    - reigning
    - archduchess
    - kingu
  - slower,slow,fast
    - fast
    - slow
    - fasts
    - food

- yet
- too
- few
- catch
- way
- fastow
- off

Q2: What are the impact on the analogies? (1pt)

- Since we are having enriched data due to increase of the dimension of vectors for each word, we are able to see that the list of closest words obtained seems to be slightly better when compared to the previous list of closest words.

THANK YOU