Lab Task-3: Training and Visualizing Word Embeddings and Using them for Spelling Error Detection

Word embeddings are a modern approach for representing text in natural language processing. Word embedding algorithms like word2vec and GloVe are key to the state-of-the-art results achieved by neural network models on natural language processing problems like machine translation.

The Main Folder in this Repo consists of 4 sub-folders : Word2Vec(Skip-gram), Word2Vec(CBOW), GloVe, CNN model for Spelling Error Detection

Word2Vec Embeddings

There are two main training algorithms that can be used to learn the embedding from text; they are a continuous bag of words (CBOW) and skip grams.

Skip-gram

In the Skip-gram model, the distributed representation of the input word is used to predict the context.

Word2Vec Skipgram folder consists of Data, Model, Jupyter notebook consisting of all the code used to perform the task and Plots obtained.

- train.txt consists of the clean dataset of words seperated by spaces which were stores in the list by iterating the loop over the sentences using the following command:

```
file = open("/content/train.txt", 'r') lines = file.readlines() list_word = [] for l in lines: list_word.append(l.split(" "))
```

- I then installed the gensim library and trained the model on the train.txt corpus using the following command:

```
model_sg = Word2Vec(list_word,min_count=1, sg=1)
```

- The above trained model was saved and used to find most similar words and to display Visualization plots.

- To check the model and display the most similar words, I stored some 4-5 words in a list and iterated over them to output their most similar words(the output is shown in the notebook i uploaded)
- These word embeddings are visualized in 4 types of plots:

1. PCA(Principal Component Analysis) for complete embedding space
2. PCA(Principal Component Analysis) for some most similar words
3. t-SNE(T- distributed Stochastic Neighbor Embedding) for complete embedding space
4. t-SNE(T- distributed Stochastic Neighbor Embedding) for some most similar words

CBOW (Continuous Bag Of Words)

In the CBOW model, the distributed representations of context (or surrounding words) are combined to predict the word in the middle.

Word2Vec CBOW folder consists of Data, Model, Jupyter notebook consisting of all the code used to perform the task and Plots obtained.

- train.txt consists of the clean dataset of words seperated by spaces which were stores in the list by iterating the loop over the sentences using the following command:

```
file = open("/content/train.txt", 'r') lines = file.readlines() list_word = [] for l in lines: list_word.append(l.split(" "))
```

- I then installed the gensim library and trained the model on the train.txt corpus using the following command:

```
model_sg = Word2Vec(list_word,min_count=1)
```

- The above trained model was saved and used to find most similar words and to display Visualization plots.
- To check the model and display the most similar words, I stored some 4-5 words in a list and iterated over them to output their most similar words(the output is shown in the notebook i uploaded)
- These word embeddings are visualized in 4 types of plots:

1. PCA(Principal Component Analysis) for complete embedding space
2. PCA(Principal Component Analysis) for some most similar words
3. t-SNE(T- distributed Stochastic Neighbor Embedding) for complete embedding space
4. t-SNE(T- distributed Stochastic Neighbor Embedding) for some most similar words

GloVe Embeddings

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

I cloned the stanford glove repository and build it using ./demo.sh using the following commands:

```
git clone https://github.com/stanfordnlp/glove %cd glove make ./demo.sh
```

I used the inbuilt vector.txt corpus and loaded glove embeddings from stanford glove embeddings using gensim library and converted GloVe file format to word2vec file format. The code used to do these tasks was as follows:

```
from gensim.test.utils import datapath, get_tmpfile
from gensim.models import KeyedVectors
from gensim.scripts.glove2word2vec import glove2word2vec

glove_file = "/content/glove/vectors.txt"
tmp_file = get_tmpfile("vectors-word2vec.txt")
_ = glove2word2vec(glove_file, tmp_file)
model = KeyedVectors.load_word2vec_format(tmp_file)
model.vocab
model.save('model_glove.bin')
```

- Saved the model with the name model_glove.bin and use it to find out most similar words of some words and For Visualizing plots.
- These word embeddings are visualized in 4 types of plots:

1. PCA(Principal Component Analysis) for complete embedding space
2. PCA(Principal Component Analysis) for some most similar words
3. t-SNE(T- distributed Stochastic Neighbor Embedding) for complete embedding space
4. t-SNE(T- distributed Stochastic Neighbor Embedding) for some most similar words

Spelling Error Detection with CNN

In this directory data folder consists of incorrect/misspelt words, correcct words and the complete dataset consisting the incorrect and correct words in 1:10 ratio.

The complete code of making the above corpus is given in the jupyter notebook uploaded there.

- this dataset was divided into testing and training data and a model was trained on the given data using the below code:

```
i = Input(shape=(CT,))
x = Embedding(V,20)(i)
x = Conv1D(1024,7,activation="relu")(x)
x = MaxPooling1D(3)(x)
x = Conv1D(1024,3,activation="relu")(x)
x = MaxPooling1D(3)(x)
x = Conv1D(1024,1,activation="relu")(x)
x = Conv1D(1024,1,activation="relu")(x)
x = Conv1D(1024,1,activation="relu")(x)
x = Conv1D(1024,1,activation="relu")(x)
x = MaxPooling1D(1)(x)
x = Dense(2048,activation="relu")(x)
x = Dense(2048,activation="relu")(x)
x = Dense(1,activation="sigmoid")(x)

model = Model(i,x)
model.compile(optimizer="adam",loss="binary_crossentropy",metrics=["accuracy"
])
```

- This model is placed in the CNN_spellcheck folder itself.
- Accuracy: This model is tested on the test data which gives out the accuracy of about 90%.