B.TECH. (2020-24)
Artificial Intelligence

NATURAL LANGUAGE PROCESSING WITH DEEP LEARNING

[CSE468]

Submitted To
Prof. (Dr.) Archana Singh

Submitted By
NIKITA
A023119820034
7AI 1

DEPARTMENT OF ARTIFICIAL INTELLIGENCE
AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY
AMITY UNIVERSITY UTTAR PRADESH
NOIDA (U.P)

INDEX

Exp. No.	Name of Experiment	Date of Allotment of experiment	Date of Evaluation	Remarks	Signature of Faculty
1	To perform classification with word vectors.	18/07/2023	01/08/2023		
2	To implement Neural Network Bigram Model.	01/08/2023	08/08/2023		
3	Implementation of word2vec using NumPy.	08/08/2023	22/08/2023		
4	Implementation of word2vec using tensorflow.	22/08/2023	29/08/2023		
5	To implement GLOVE using numpy gradient descent.	29/08/2023	05/09/2023		
6	To implement GLOVE using Alternative Least Squares.	05/09/2023	12/09/2023		
7	Visualizing data with analogies with t-SNE.	12/09/2023	19/09/2023		
8	To visualize the data analogies using embedding projectors.	19/09/2023	26/09/2023		
9	Implement GloVe using tensorflow gradient descent.	26/09/2023	03/10/2023		
10	To perform point wise Mutual Information.	03/10/2023	10/10/2023		
11	Implement Recursive Neural Tensor Network using tensorflow.	10/10/2023	17/10/2023		

Experiment 1

AIM
To perform classification with word vectors.

SOFTWARE USED
Jupyter Notebook

DATASET USED
IMDB dataset from Hugging Face's datasets library

THEORY
A word vector is an attempt to mathematically represent the meaning of a word. In essence, a computer goes through some text (ideally a lot of text) and calculates how often words show up next to each other. These frequencies are represented with numbers. So, if the word 'good' always shows up next to the word 'friend' then part of the word vector for 'good' will reflect that connection. A given word will have a vast number of such values, usually in the hundreds and sometimes in the thousands.
Once you have this set of numbers for a word, you can compare the vectors of different words. For example, you could compare the different vectors for the words 'banana,' 'kiwi,' and 'raincloud.' Since the vectors are mathematical objects, you can calculate the numerical similarity of different vectors. And since 'banana' and 'kiwi' are used in a lot of similar contexts and are not used in the contexts where 'raincloud' shows up, their vectors will be closer to each other and farther from the vector for 'raincloud.'

CODE

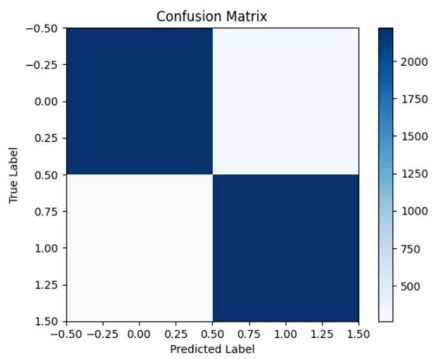
```
from datasets import load_dataset
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import random

# Load the IMDB dataset from Hugging Face's datasets library
dataset = load_dataset("imdb")

# Get the total number of examples in the dataset
total_examples = len(dataset["train"]["text"])

# Randomly pick 5 indices
random_indices = random.sample(range(total_examples), 5)

# Extract random reviews and labels
random_reviews = [dataset["train"]["text"][i] for i in random_indices]
random_labels = [dataset["train"]["label"][i] for i in random_indices]

# Display the randomly picked samples
print(f"Sample Dataset:")
for review, label in zip(random_reviews, random_labels):
    print(f"Label: {label}")
    print(f"Review: {review}")
    print("=" * 50)
print()
```

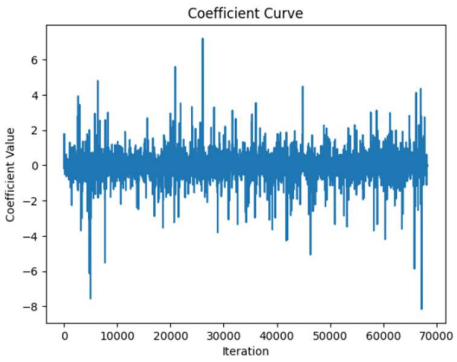
```
# Extract text and labels
texts = dataset["train"]["text"]
labels = dataset["train"]["label"]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(texts, labels, test_size=0.2,
random_state=42)

# Convert text data to feature vectors using TfidfVectorizer
vectorizer = TfidfVectorizer()
X_train_vectors = vectorizer.fit_transform(X_train)
X_test_vectors = vectorizer.transform(X_test)

# Build and train a logistic regression classifier with increased max_iter
classifier = LogisticRegression(max_iter=1000)  # Increase max_iter
classifier.fit(X_train_vectors, y_train)

# Make predictions on the test set
predictions = classifier.predict(X_test_vectors)

# Display a few samples with predictions and actual labels
print(f"Some Predictions:")
for i in range(5):
    print(f"Text: {X_test[i]}")
    print(f"Prediction: {predictions[i]}, Actual: {y_test[i]}")
    print()

# Plot confusion matrix
cm = confusion_matrix(y_test, predictions)
plt.imshow(cm, interpolation="nearest", cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.colorbar()
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

# Plot loss curve
plt.plot(range(len(classifier.coef_[0])), classifier.coef_[0])
plt.title("Coefficient Curve")
plt.xlabel("Iteration")
plt.ylabel("Coefficient Value")
plt.show()

# Calculate and plot accuracy
train_predictions = classifier.predict(X_train_vectors)
test_predictions = classifier.predict(X_test_vectors)

train_accuracy = accuracy_score(y_train, train_predictions)
test_accuracy = accuracy_score(y_test, test_predictions)

# Plot accuracy curves
plt.plot(range(len(classifier.coef_[0])), [train_accuracy] * len(classifier.coef_[0]),
label='Train', linestyle='--')
```

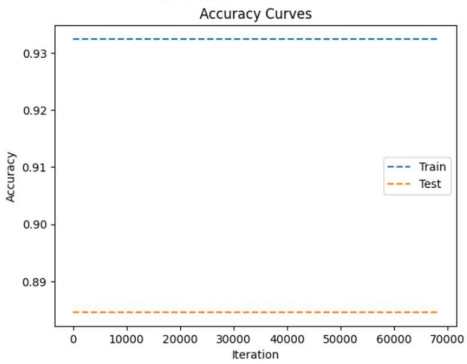
```
plt.plot(range(len(classifier.coef_[0])), [test_accuracy] * len(classifier.coef_[0]),
label='Test', linestyle='--')
plt.title("Accuracy Curves")
plt.xlabel("Iteration")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

# Print accuracy values
print(f'train_accuracy: {train_accuracy}')
print(f'test_accuracy: {test_accuracy}')
```

OUTPUT

```
Sample Dataset:
Label: 1
Review: Russian emigrant director in Hollywood in 1928 (William Powell) is casting his epic about the Russian revolution, and hires an old ex-general from the Czarist regime (Emi
==================================================
Label: 1
Review: This is the epitome of fairytale! The villains are completely wicked and the heroes are refreshingly pure. Danes, Deniro,and Pfieffer are wonderful as well as the new act
==================================================
Label: 0
Review: OK, I bought this film from Woolworths for my friend for a joke present on his birthday, because the front cover had a sexual innuendo in it.<br /><br />But we decided it
==================================================
Label: 0
Review: Of all the movies I've seen, this one rates almost at the bottom (Haunted Mansion, Nothing but Trouble and a few others keep it from reaching rock bottom.) It is hasty, tl
==================================================
Label: 1
Review: "Spaced Invaders" is one of the funniest movies, I´ve ever seen. I don´t understand, why this movie didn´t get better critics, it´s funny, harmless and sweet. I first wat
==================================================


Some Predictions:
Text: Dumb is as dumb does, in this thoroughly uninteresting, supposed black comedy. Essentially what starts out as Chris Klein trying to maintain a low prof
Prediction: 0, Actual: 0

Text: I dug out from my garage some old musicals and this is another one of my favorites. It was written by Jay Alan Lerner and directed by Vincent Minelli.
Prediction: 1, Actual: 1

Text: After watching this movie I was honestly disappointed - not because of the actors, story or directing - I was disappointed by this film advertisements.
Prediction: 0, Actual: 0

Text: This movie was nominated for best picture but lost out to Casablanca but Paul Lukas beat out Humphrey Bogart for best actor. I don't see why Lucile Wat
Prediction: 1, Actual: 1

Text: Just like Al Gore shook us up with his painfully honest and cleverly presented documentary-movie "An inconvenient truth", directors Alastair Fothergill
Prediction: 1, Actual: 1
```








```
train_accuracy: 0.9324
test_accuracy: 0.8846
```

RESULT

The implementation was successful.

Experiment 2

AIM
To implement Neural Network Bigram Model.

SOFTWARE USED
Jupyter Notebook

DATASET USED
NLTK's Reuters corpus

THEORY
The bigram model approximates the probability of a word given all the previous words by using only the conditional probability of one preceding word.
And so, when you use a bigram model to predict the conditional probability of the next word, you are thus making the following approximation:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1})$$

CODE

```
# imports
import string
import random
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('reuters')
from nltk.corpus import reuters
from nltk import FreqDist

# input the reuters sentences
sents  =reuters.sents()

# write the removal characters such as : Stopwords and punctuation
stop_words = set(stopwords.words('english'))
string.punctuation = string.punctuation +'"'+'"'+'-'+'''+'''+'—'
string.punctuation
removal_list = list(stop_words) + list(string.punctuation)+ ['lt','rt']
removal_list

# generate unigrams bigrams trigrams
unigram=[]
bigram=[]
trigram=[]
tokenized_text=[]
for sentence in sents:
  sentence = list(map(lambda x:x.lower(),sentence))
  for word in sentence:
        if word== '.':
            sentence.remove(word)
        else:
            unigram.append(word)

  tokenized_text.append(sentence)
  bigram.extend(list(ngrams(sentence, 2,pad_left=True, pad_right=True)))
  trigram.extend(list(ngrams(sentence, 3, pad_left=True, pad_right=True)))

# remove the n-grams with removable words
def remove_stopwords(x):
    y = []
    for pair in x:
```

```
        count = 0
        for word in pair:
            if word in removal_list:
                count = count or 0
            else:
                count = count or 1
        if (count==1):
            y.append(pair)
    return (y)
unigram = remove_stopwords(unigram)
bigram = remove_stopwords(bigram)
trigram = remove_stopwords(trigram)

# generate frequency of n-grams
freq_bi = FreqDist(bigram)
freq_tri = FreqDist(trigram)

d = defaultdict(Counter)
for a, b, c in freq_tri:
    if(a != None and b!= None and c!= None):
      d[a, b] += freq_tri[a, b, c]


# Next word prediction
s=''
def pick_word(counter):
    "Chooses a random element."
    return random.choice(list(counter.elements()))
prefix = "he", "said"
print(" ".join(prefix))
s = " ".join(prefix)
for i in range(19):
    suffix = pick_word(d[prefix])
    s=s+' '+suffix
    print(s)
    prefix = prefix[1], suffix
```

OUTPUT

```
he said
he said kotc
he said kotc made
he said kotc made profits
he said kotc made profits of
he said kotc made profits of 265
he said kotc made profits of 265 ,
he said kotc made profits of 265 , 457
he said kotc made profits of 265 , 457 vs
he said kotc made profits of 265 , 457 vs loss
he said kotc made profits of 265 , 457 vs loss eight
he said kotc made profits of 265 , 457 vs loss eight cts
he said kotc made profits of 265 , 457 vs loss eight cts net
he said kotc made profits of 265 , 457 vs loss eight cts net loss
he said kotc made profits of 265 , 457 vs loss eight cts net loss 343
he said kotc made profits of 265 , 457 vs loss eight cts net loss 343 ,
he said kotc made profits of 265 , 457 vs loss eight cts net loss 343 , 266
he said kotc made profits of 265 , 457 vs loss eight cts net loss 343 , 266 ,
he said kotc made profits of 265 , 457 vs loss eight cts net loss 343 , 266 , 000
he said kotc made profits of 265 , 457 vs loss eight cts net loss 343 , 266 , 000 shares
```

RESULT

The implementation was successful.

Experiment 3

AIM

Implementation of word2vec using NumPy.

SOFTWARE USED

Jupyter Notebook

DATASET/CORPUS USED

```
corpus = [['this','mobile','is','good','not','affordable']]
```

THEORY

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
Word2Vec is a method to construct such an embedding. It can be obtained using two methods (both involving Neural Networks): Skip Gram and Common Bag Of Words (CBOW). word2vec is not a singular algorithm, rather, it is a family of model architectures and optimizations that can be used to learn word embedding's from large datasets. Embeddings learned through word2vec have proven to be successful on a variety of downstream natural language processing tasks.

- ***Continuous bag-of-words model:*** predicts the middle word based on surrounding context words. The context consists of a few words before and after the current (middle) word. This architecture is called a bag-of-words model as the order of words in the context is not important.
- ***Continuous skip-gram model:*** predicts words within a certain range before and after the current word in the same sentence.

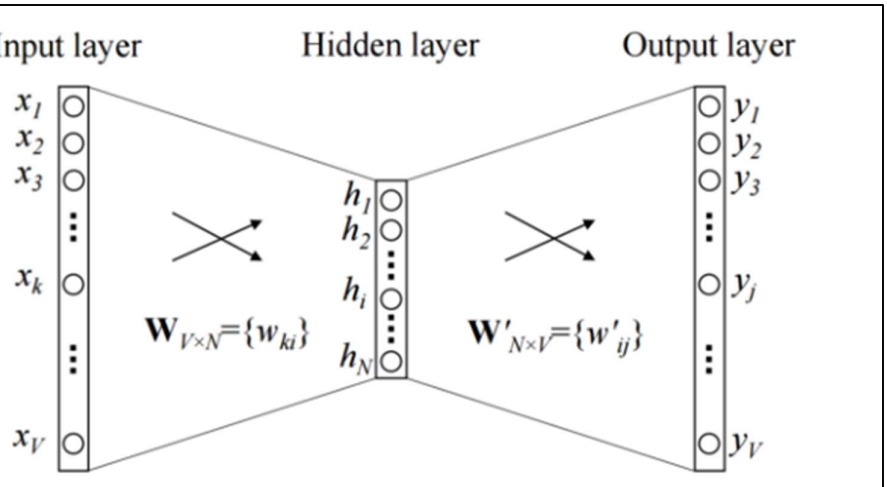


Figure 1: A simple CBOW model with only one word in the context

CODE

```
import numpy as np
import pandas as pd
import re
from collections import defaultdict
class word2vec():
    def _init_ (self):
        self.n = settings['n']
        self.eta = settings['learning_rate']
        self.epochs = settings['epochs']
        self.window = settings['window_size']
        pass
    # GENERATE TRAINING DATA
    def generate_training_data(self, settings, corpus):
        # GENERATE WORD COUNTS
```

```
        word_counts = defaultdict(int)
        for row in corpus:
            for word in row:
                word_counts[word] += 1
        self.v_count = len(word_counts.keys())
        # GENERATE LOOKUP DICTIONARIES
        self.words_list = sorted(list(word_counts.keys()),reverse=False)
        self.word_index = dict((word, i) for i, word in enumerate(self.words_list))
        self.index_word = dict((i, word) for i, word in enumerate(self.words_list))
        training_data = []
        # CYCLE THROUGH EACH SENTENCE IN CORPUS
        for sentence in corpus:
            sent_len = len(sentence)
            # CYCLE THROUGH EACH WORD IN SENTENCE
            for i, word in enumerate(sentence):
                #w_target  = sentence[i]
                w_target = self.word2onehot(sentence[i])
                # CYCLE THROUGH CONTEXT WINDOW
                w_context = []
                for j in range(i-self.window, i+self.window+1):
                    if j!=i and j<=sent_len-1 and j>=0:
                        w_context.append(self.word2onehot(sentence[j]))
                training_data.append([w_target, w_context])
        return np.array(training_data)
    # SOFTMAX ACTIVATION FUNCTION
    def softmax(self, x):
        e_x = np.exp(x - np.max(x))
        return e_x / e_x.sum(axis=0)
    # CONVERT WORD TO ONE HOT ENCODING
    def word2onehot(self, word):
        word_vec = [0 for i in range(0, self.v_count)]
        word_index = self.word_index[word]
        word_vec[word_index] = 1
        return word_vec
    # FORWARD PASS
    def forward_pass(self, x):
        h = np.dot(self.w1.T, x)
        u = np.dot(self.w2.T, h)
        y_c = self.softmax(u)
        return y_c, h, u
    # BACKPROPAGATION
    def backprop(self, e, h, x):
        dl_dw2 = np.outer(h, e)
        dl_dw1 = np.outer(x, np.dot(self.w2, e.T))
        # UPDATE WEIGHTS
        self.w1 = self.w1 - (self.eta * dl_dw1)
        self.w2 = self.w2 - (self.eta * dl_dw2)
        pass
    # TRAIN W2V model
    def train(self, training_data):
        # INITIALIZE WEIGHT MATRICES
        self.w1 = np.random.uniform(-0.8, 0.8, (self.v_count, self.n))     # embedding matrix
        self.w2 = np.random.uniform(-0.8, 0.8, (self.n, self.v_count))     # context matrix
        # CYCLE THROUGH EACH EPOCH
        for i in range(0, self.epochs):
            self.loss = 0
            # CYCLE THROUGH EACH TRAINING SAMPLE
            for w_t, w_c in training_data:

                # FORWARD PASS
                y_pred, h, u = self.forward_pass(w_t)

                # CALCULATE ERROR
                EI = np.sum([np.subtract(y_pred, word) for word in w_c], axis=0)
```

```
                # BACKPROPAGATION
                self.backprop(EI, h, w_t)

                # CALCULATE LOSS
                self.loss  +=  -np.sum([u[word.index(1)]  for  word  in  w_c])  +  len(w_c)  *
np.log(np.sum(np.exp(u)))
                #self.loss += -2*np.log(len(w_c)) -np.sum([u[word.index(1)] for word in w_c])
+ (len(w_c) * np.log(np.sum(np.exp(u))))

            print('EPOCH:',i, 'LOSS:', self.loss)

    # input a word, returns a vector (if available)
    def word_vec(self, word):
        w_index = self.word_index[word]
        v_w = self.w1[w_index]
        return v_w

    # input a vector, returns nearest word(s)
    def vec_sim(self, vec, top_n):

        # CYCLE THROUGH VOCAB
        word_sim = {}
        for i in range(self.v_count):
            v_w2 = self.w1[i]
            theta_num = np.dot(vec, v_w2)
            theta_den = np.linalg.norm(vec) * np.linalg.norm(v_w2)
            theta = theta_num / theta_den

            word = self.index_word[i]
            word_sim[word] = theta

        words_sorted = sorted(word_sim.items(), key=lambda x: x[1], reverse=True)

        for word, sim in words_sorted[:top_n]:
            print(word, sim)

    # input word, returns top [n] most similar words
    def word_sim(self, word, top_n):

        w1_index = self.word_index[word]
        v_w1 = self.w1[w1_index]

        # CYCLE THROUGH VOCAB
        word_sim = {}
        for i in range(self.v_count):
            v_w2 = self.w1[i]
            theta_num = np.dot(v_w1, v_w2)
            theta_den = np.linalg.norm(v_w1) * np.linalg.norm(v_w2)
            theta = theta_num / theta_den

            word = self.index_word[i]
            word_sim[word] = theta

        words_sorted = sorted(word_sim.items(), key=lambda x: x[1], reverse=True)

        for word, sim in words_sorted[:top_n]:
            print(word, sim)
```

EXAMPLE RUN and OUTPUT

```
settings = {}
settings['n'] = 5                   # dimension of word embeddings
settings['window_size'] = 2         # context window +/- center word
settings['min_count'] = 0           # minimum word count
settings['epochs'] = 600           # number of training epochs
settings['neg_samp'] = 10           # number of negative words to use during training
```

```
settings['learning_rate'] = 0.01    # learning rate
np.random.seed(0)                   # set the seed for reproducibility

corpus = [['this','mobile','is','good','not','affordable']]

# INITIALIZE W2V MODEL
w2v = word2vec()

# generate training data
training_data = w2v.generate_training_data(settings, corpus)

# train word2vec model
w2v.train(training_data)
```

```
array([ 0.62697258, -0.8435938 ,  2.10190231,  0.26548012,  0.2082425 ])
```

```
EPOCH: 584 LOSS: 20.631999407572565
EPOCH: 585 LOSS: 20.631520106423608
EPOCH: 586 LOSS: 20.631043111386806
EPOCH: 587 LOSS: 20.63056840670071
EPOCH: 588 LOSS: 20.630095976742535
EPOCH: 589 LOSS: 20.629625806026706
EPOCH: 590 LOSS: 20.629157879203383
EPOCH: 591 LOSS: 20.628692181056984
EPOCH: 592 LOSS: 20.628228696504777
EPOCH: 593 LOSS: 20.62776741059548
EPOCH: 594 LOSS: 20.627308308507835
EPOCH: 595 LOSS: 20.626851375549258
EPOCH: 596 LOSS: 20.626396597154457
EPOCH: 597 LOSS: 20.625943958884108
EPOCH: 598 LOSS: 20.625493446423523
EPOCH: 599 LOSS: 20.625045045581 32
```

RESULT

The implementation was successful.

Experiment 4

AIM

Implementation of word2vec using tensorflow.

SOFTWARE USED

Jupyter Notebook

DATASET/CORPUS USED

```
corpus_raw = 'this mobile is good this mobile is not good this mobile is affordable'
```

THEORY

Word2Vec is a method to construct such an embedding. It can be obtained using two methods (both involving Neural Networks): Skip Gram and Common Bag Of Words (CBOW). word2vec is not a singular algorithm, rather, it is a family of model architectures and optimizations that can be used to learn word embedding's from large datasets. Embedding's learned through word2vec have proven to be successful on a variety of downstream natural language processing tasks.

TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

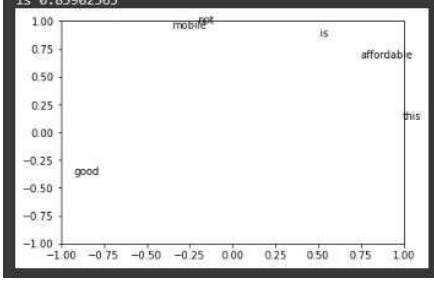
CODE

```
from _future_ import absolute_import, division, print_function, unicode_literals
import tensorflow as tf
import numpy as np

class Word2Vec:
    def _init_(self, vocab_size=0, embedding_dim=16, optimizer='sgd', epochs=10000):
        self.vocab_size=vocab_size
        self.embedding_dim=5
        self.epochs=epochs
        if optimizer=='adam':
            self.optimizer = tf.optimizers.Adam()
        else:
            self.optimizer = tf.optimizers.SGD(learning_rate=0.1)
    def train(self, x_train=None, y_train=None):
        self.W1 = tf.Variable(tf.random.normal([self.vocab_size, self.embedding_dim]))
        self.b1 = tf.Variable(tf.random.normal([self.embedding_dim])) #bias
        self.W2 = tf.Variable(tf.random.normal([self.embedding_dim, self.vocab_size]))
        self.b2 = tf.Variable(tf.random.normal([self.vocab_size]))
        for _ in range(self.epochs):
            with tf.GradientTape() as t:
                hidden_layer = tf.add(tf.matmul(x_train,self.W1),self.b1)
                output_layer   =   tf.nn.softmax(tf.add(   tf.matmul(hidden_layer,   self.W2),
self.b2))
                cross_entropy_loss     =     tf.reduce_mean(-tf.math.reduce_sum(y_train     *
tf.math.log(output_layer), axis=[1]))
            grads = t.gradient(cross_entropy_loss, [self.W1, self.b1, self.W2, self.b2])
            self.optimizer.apply_gradients(zip(grads,[self.W1, self.b1, self.W2, self.b2]))
            if(_ % 1000 == 0):
                print(cross_entropy_loss)
    def vectorized(self, word_idx):
        return (self.W1+self.b1)[word_idx]

corpus_raw = 'this mobile is good this mobile is not good this mobile is affordable'
# convert to lower case
corpus_raw = corpus_raw.lower()
# raw sentences is a list of sentences.
raw_sentences = corpus_raw.split('.')
sentences = []
for sentence in raw_sentences:
    sentences.append(sentence.split())
```

```
data = []
WINDOW_SIZE = 2
for sentence in sentences:
    for word_index, word in enumerate(sentence):
        for nb_word in sentence[max(word_index - WINDOW_SIZE, 0) : min(word_index +
WINDOW_SIZE, len(sentence)) + 1] :
            if nb_word != word:
                data.append([word, nb_word])

words = []
for word in corpus_raw.split():
    if word != '.': # because we don't want to treat . as a word
        words.append(word)
words = set(words) # so that all duplicate words are removed
word2int = {}
int2word = {}
vocab_size = len(words) # gives the total number of unique words
for i,word in enumerate(words):
    word2int[word] = i
    int2word[i] = word

# function to convert numbers to one hot vectors
def to_one_hot(data_point_index, vocab_size):
    temp = np.zeros(vocab_size)
    temp[data_point_index] = 1
    return temp
x_train = [] # input word
y_train = [] # output word
for data_word in data:
    x_train.append(to_one_hot(word2int[ data_word[0] ], vocab_size))
    y_train.append(to_one_hot(word2int[ data_word[1] ], vocab_size))
# convert them to numpy arrays
x_train = np.asarray(x_train, dtype='float32')
y_train = np.asarray(y_train, dtype='float32')

w2v = Word2Vec(vocab_size=vocab_size, optimizer='adam', epochs=10000)
w2v.train(x_train, y_train)

w2v.vectorized(word2int['mobile'])

sentences = sentences[0]

vectors = []
for i in sentences:
    vectors.append(w2v.vectorized(word2int[i]))

from sklearn.manifold import TSNE
from sklearn import preprocessing
model = TSNE(n_components=2, random_state=0)
np.set_printoptions(suppress=True)
vectors = model.fit_transform(vectors)
normalizer = preprocessing.Normalizer()
vectors = normalizer.fit_transform(vectors, 'l2')

import matplotlib.pyplot as plt
fig, ax = plt.subplots()
ax.set_xlim(left=-1, right=1)
ax.set_ylim(bottom=-1, top=1)
for word in words:
    print(word, vectors[word2int[word]][1])
    ax.annotate(word, (vectors[word2int[word]][0],vectors[word2int[word]][1] ))
plt.show()
```

OUTPUT

```
tf.Tensor(3.3884168, shape=(), dtype=float32)
tf.Tensor(1.4622096, shape=(), dtype=float32)
tf.Tensor(1.4024959, shape=(), dtype=float32)
tf.Tensor(1.376369, shape=(), dtype=float32)
tf.Tensor(1.3706317, shape=(), dtype=float32)
tf.Tensor(1.3691655, shape=(), dtype=float32)
tf.Tensor(1.3686327, shape=(), dtype=float32)
tf.Tensor(1.3684037, shape=(), dtype=float32)
tf.Tensor(1.3682947, shape=(), dtype=float32)
tf.Tensor(1.3682394, shape=(), dtype=float32)
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:783: FutureWarning: The default initialization in TSNE will change from 'random' to 'pca' in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:793: FutureWarning: The default learning rate in TSNE will change from 200.0 to 'auto' in 1.2.
  FutureWarning,
mobile 0.9364252
affordable 0.6647415
good -0.38220012
not 0.9795167
this 0.11703096
is 0.85902363
```



RESULT

The implementation was successful.

Experiment 5

AIM

To implement GLOVE using numpy gradient descent.

SOFTWARE USED

Jupyter Notebook

DATASET/CORPUS USED

Genism Data – "text8"

THEORY

GloVe is a word vector technique that rode the wave of word vectors after a brief silence. Just to refresh, word vectors put words to a nice vector space, where similar words cluster together and different words repel. The advantage of GloVe is that, unlike Word2vec, GloVe does not rely just on local statistics (local context information of words) but incorporates global statistics (word co-occurrence) to obtain word vectors.

CODE

```
import gensim.downloader as api
dataset = api.load("text8")
import itertools
corpus = list(itertools.chain.from_iterable(dataset))
```

```
def load_config():
    config_filepath = "config.yaml's file path"
    with config_filepath.open() as f:
        config_dict = yaml.load(f, Loader=yaml.FullLoader)
    config = argparse.Namespace()
    for key, value in config_dict.items():
        setattr(config, key, value)
    return config
```

```
@dataclass
class Vocabulary:
    token2index: dict = field(default_factory=dict)
    index2token: dict = field(default_factory=dict)
    token_counts: list = field(default_factory=list)
    _unk_token: int = field(init=False, default=-1)

    def add(self, token):
        if token not in self.token2index:
            index = len(self)
            self.token2index[token] = index
            self.index2token[index] = token
            self.token_counts.append(0)
        self.token_counts[self.token2index[token]] += 1

    def get_topk_subset(self, k):
        tokens = sorted(
            list(self.token2index.keys()),
            key=lambda token: self.token_counts[self[token]],
            reverse=True
        )
        return type(self)(
            token2index={token: index for index, token in enumerate(tokens[:k])},
            index2token={index: token for index, token in enumerate(tokens[:k])},
            token_counts=[
                self.token_counts[self.token2index[token]] for token in tokens[:k]
            ]
        )
```

```
    def shuffle(self):
        new_index = [_ for _ in range(len(self))]
        random.shuffle(new_index)
        new_token_counts = [None] * len(self)
        for token, index in zip(list(self.token2index.keys()), new_index):
            new_token_counts[index] = self.token_counts[self[token]]
            self.token2index[token] = index
            self.index2token[index] = token
        self.token_counts = new_token_counts

    def get_index(self, token):
        return self[token]

    def get_token(self, index):
        if not index in self.index2token:
            raise Exception("Invalid index.")
        return self.index2token[index]

    @property
    def unk_token(self):
        return self._unk_token

    def __getitem__(self, token):
        if token not in self.token2index:
            return self._unk_token
        return self.token2index[token]

    def __len__(self):
        return len(self.token2index)
```

```
@dataclass
class Vectorizer:
    vocab: Vocabulary

    @classmethod
    def from_corpus(cls, corpus, vocab_size):
        vocab = Vocabulary()
        for token in corpus:
            vocab.add(token)
        vocab_subset = vocab.get_topk_subset(vocab_size)
        vocab_subset.shuffle()
        return cls(vocab_subset)

    def vectorize(self, corpus):
        return [self.vocab[token] for token in corpus]
```

```
@dataclass
class CooccurrenceEntries:
    vectorized_corpus: list
    vectorizer: Vectorizer

    @classmethod
    def setup(cls, corpus, vectorizer):
        return cls(
            vectorized_corpus=vectorizer.vectorize(corpus),
            vectorizer=vectorizer
        )
```

```
    def validate_index(self, index, lower, upper):
        is_unk = index == self.vectorizer.vocab.unk_token
        if lower < 0:
            return not is_unk
        return not is_unk and index >= lower and index <= upper

    def build(
        self,
        window_size,
        num_partitions,
        chunk_size,
        output_directory="."
    ):
        partition_step = len(self.vectorizer.vocab) // num_partitions
        split_points = [0]
        while split_points[-1] + partition_step <= len(self.vectorizer.vocab):
            split_points.append(split_points[-1] + partition_step)
        split_points[-1] = len(self.vectorizer.vocab)

        for partition_id in tqdm(range(len(split_points) - 1)):
            index_lower = split_points[partition_id]
            index_upper = split_points[partition_id + 1] - 1
            cooccurr_counts = Counter()
            for i in tqdm(range(len(self.vectorized_corpus))):
                if not self.validate_index(
                    self.vectorized_corpus[i],
                    index_lower,
                    index_upper
                ):
                    continue

                context_lower = max(i - window_size, 0)
                context_upper = min(i + window_size + 1, len(self.vectorized_corpus))
                for j in range(context_lower, context_upper):
                    if i == j or not self.validate_index(
                        self.vectorized_corpus[j],
                        -1,
                        -1
            ):
                        continue
                    cooccurr_counts[(self.vectorized_corpus[i], self.vectorized_corpus[j])] += 1 / abs(i - j)

    cooccurr_dataset = np.zeros((len(cooccurr_counts), 3))
    for index, ((i, j), cooccurr_count) in enumerate(cooccurr_counts.items()):
        cooccurr_dataset[index] = (i, j, cooccurr_count)
    if partition_id == 0:
        file = h5py.File(
            os.path.join(
                output_directory,
                "cooccurrence.hdf5"
            ),
            "w"
        )
        dataset = file.create_dataset(
            "cooccurrence",
            (len(cooccurr_counts), 3),
            maxshape=(None, 3),
            chunks=(chunk_size, 3)
        )
        prev_len = 0
    else:
        prev_len = dataset.len()
        dataset.resize(dataset.len() + len(cooccurr_counts), axis=0)
    dataset[prev_len: dataset.len()] = cooccurr_dataset
```

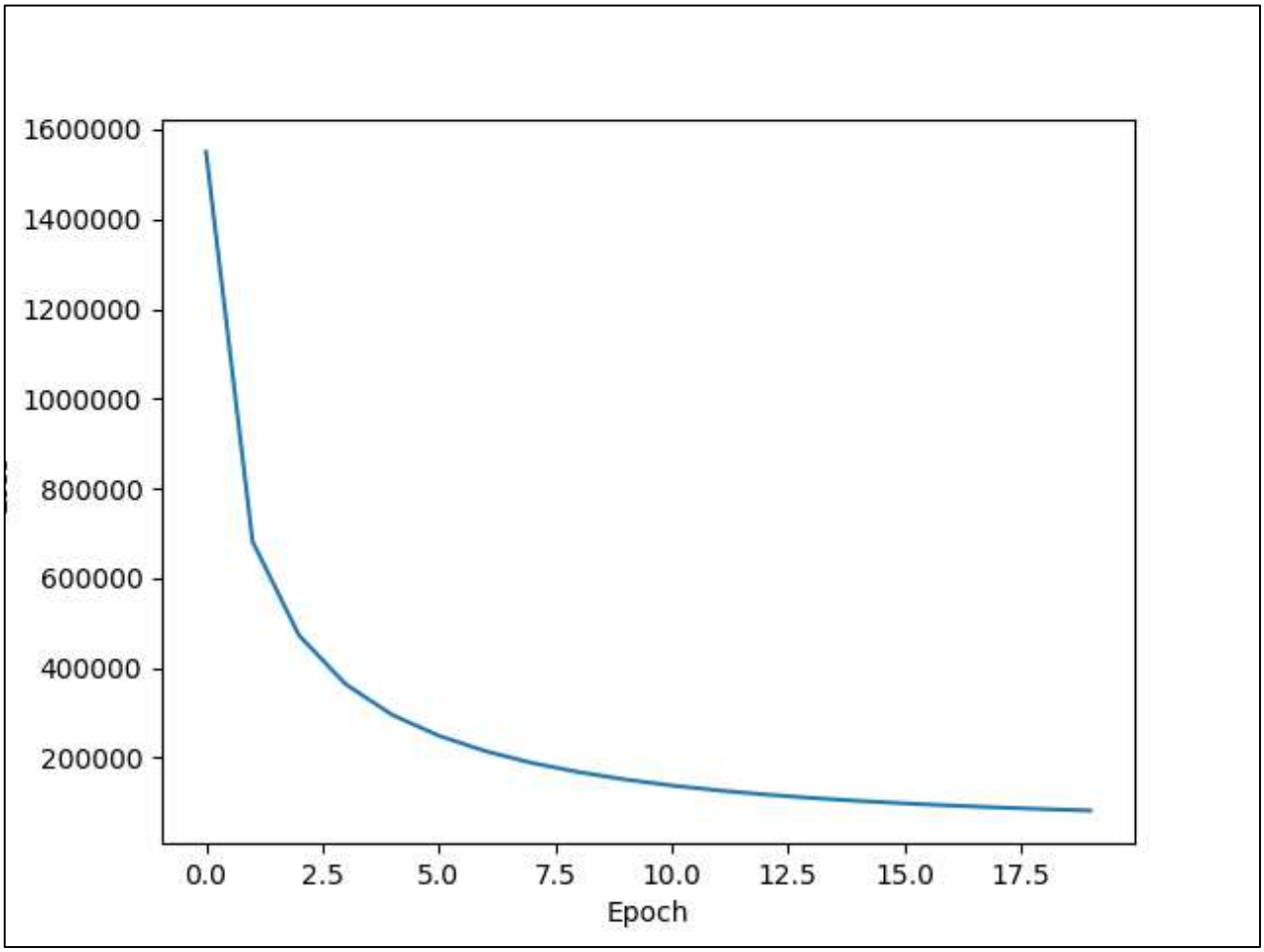
```
file.close()
with open(os.path.join(output_directory, "vocab.pkl"), "wb") as file:
    pickle.dump(self.vectorizer.vocab, file)
```

```
class GloVe(nn.Module):

    def __init__(self, vocab_size, embedding_size, x_max, alpha):
        super().__init__()
        self.weight = nn.Embedding(
            num_embeddings=vocab_size,
            embedding_dim=embedding_size,
            sparse=True
        )
        self.weight_tilde = nn.Embedding(
            num_embeddings=vocab_size,
            embedding_dim=embedding_size,
            sparse=True
        )
        self.bias = nn.Parameter(
            torch.randn(
                vocab_size,
                dtype=torch.float,
            )
        )
        self.bias_tilde = nn.Parameter(
            torch.randn(
                vocab_size,
                dtype=torch.float,
            )
        )
        self.weighting_func = lambda x: (x / x_max).float_power(alpha).clamp(0, 1)

    def forward(self, i, j, x):
        loss = torch.mul(self.weight(i), self.weight_tilde(j)).sum(dim=1)
        loss = (loss + self.bias[i] + self.bias_tilde[j] - x.log()).square()
        loss = torch.mul(self.weighting_func(x), loss).mean()
        return loss
```

```
dataloader = HDF5DataLoader(
    filepath=os.path.join(config.cooccurrence_dir, "cooccurrence.hdf5"),
    dataset_name="cooccurrence",
    batch_size=config.batch_size,
    device=config.device
)
model = GloVe(
    vocab_size=config.vocab_size,
    embedding_size=config.embedding_size,
    x_max=config.x_max,
    alpha=config.alpha
)
model.to(config.device)
optimizer = torch.optim.Adagrad(
    model.parameters(),
    lr=config.learning_rate
)
```

```
with dataloader.open():
    model.train()
    losses = []
    for epoch in tqdm(range(config.num_epochs)):
        epoch_loss = 0
        for batch in tqdm(dataloader.iter_batches()):
            loss = model(
                batch[0][:, 0],
                batch[0][:, 1],
                batch[1]
            )
            epoch_loss += loss.detach().item()
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()

        losses.append(epoch_loss)
        print(f"Epoch {epoch}: loss = {epoch_loss}")
        torch.save(model.state_dict(), config.output_filepath)
```

OUTPUT



```
How similar is man and woman:
0.65315914
How similar is man and apple:
0.25869069
How similar is woman and apple:
0.040839735
Most similar words of computer:
['book', 'early', 'game', 'country', 'today', 'particular', 'program', 'death', 'within', 'many']
Most similar words of united:
['states', 'under', 'world', 'during', 'nine', 'eight', 'seven', 'following', 'first', 'american']
Most similar words of early:
['first', 'after', 'only', 'several', 'had', 'he', 'when', 'before', 'even', 'their']
```

RESULT

The implementation was successful.

Experiment 6

AIM
To implement GLOVE using Alternative Least Squares.

SOFTWARE USED
Google Colaboratory

DATASET/CORPUS USED
Kaggle Retail Rocket recommender system dataset – events.csv

THEORY
Alternating Least Square (ALS) is also a matrix factorization algorithm and it runs itself in a parallel fashion. ALS is implemented in Apache Spark ML and built for a larges-scale collaborative filtering problems. ALS is doing a pretty good job at solving scalability and sparseness of the Ratings data, and it's simple and scales well to very large datasets.

CODE

```
!pip install implicit

#import libraries
import sys
import pandas as pd
import numpy as np
import scipy.sparse as sparse
from scipy.sparse.linalg import spsolve
import random

from sklearn.preprocessing import MinMaxScaler

import implicit
from datetime import datetime, timedelta

#Data Preprocessing
def create_data(datapath,start_date,end_date):
    df=pd.read_csv(datapath)
    df=df.assign(date=pd.Series(datetime.fromtimestamp(a/1000).date() for a in df.timestamp))
    df=df.sort_values(by='date').reset_index(drop=True) # for some reasons RetailRocket did
NOT sort data by date
    df=df[(df.date>=datetime.strptime(start_date,'%Y-%m-
%d').date())&(df.date<=datetime.strptime(end_date,'%Y-%m-%d').date())]
    df=df[['visitorid','itemid','event']]
    return df
#Download the kaggle RetailRocket data and give the events.csv file path
datapath= '/content/drive/MyDrive/dataset/events.csv'
data=create_data(datapath,'2015-5-3','2015-5-18')
data['visitorid'] = data['visitorid'].astype("category")
data['itemid'] = data['itemid'].astype("category")
data['visitor_id'] = data['visitorid'].cat.codes
data['item_id'] = data['itemid'].cat.codes

data['event']=data['event'].astype('category')
data['event']=data['event'].cat.codes
```

```
sparse_item_user = sparse.csr_matrix((data['event'].astype(float), (data['item_id'],
data['visitor_id'])))

sparse_user_item = sparse.csr_matrix((data['event'].astype(float), (data['visitor_id'],
data['item_id'])))

#Building the model
model = implicit.als.AlternatingLeastSquares(factors=20, regularization=0.1, iterations=20)

alpha_val = 40
data_conf = (sparse_item_user * alpha_val).astype('double')

model.fit(data_conf)

###USING THE MODEL

#Get Recommendations
user_id =   14
recommended = model.recommend(user_id, sparse_user_item[user_id])

recommended = pd.DataFrame(recommended)
recommended = recommended.T
recommended.columns = ['items', 'scores']

print(recommended)
#Get similar items
item_id = 7
n_similar = 4
similar = model.similar_items(item_id, n_similar)

similar = pd.DataFrame(similar)
similar = similar.T
similar.columns = ['items', 'scores']

print(similar)
```

OUTPUT

```
0s
1 #Get Recommendations
2 user_id =   14
3 recommended = model.recommend(user_id, sparse_user_item[user_id])
4
5 recommended = pd.DataFrame(recommended)
6 recommended = recommended.T
7 recommended.columns = ['items', 'scores']
8
9 print(recommended)
```

```
      items       scores
0   27681.0  1.220773e-12
1   42185.0  1.044047e-12
2  115528.0  9.106946e-13
3    7842.0  8.595597e-13
4   26717.0  7.493854e-13
5   19762.0  7.466673e-13
6   30619.0  7.413609e-13
7  103203.0  7.381930e-13
8   41991.0  7.261463e-13
9   44878.0  7.224949e-13
```

```
#Get similar items
item_id = 7
n_similar = 4
similar = model.similar_items(item_id, n_similar)

similar = pd.DataFrame(similar)
similar = similar.T
similar.columns = ['items', 'scores']

print(similar)
```

```
      items    scores
0       7.0  1.000000
1  164779.0  0.999999
2   80629.0  0.999999
3  130467.0  0.999992
```

RESULT

The implementation was successful.

Experiment 7

AIM
Visualizing data with analogies with t-SNE.

SOFTWARE USED
Jupyter Notebook

DATASET/CORPUS USED
MNIST iris dataset

THEORY
t-Distributed Stochastic Neighbor Embedding (t-SNE) is an unsupervised, non-linear technique primarily used for data exploration and visualizing high-dimensional data. In simpler terms, t-SNE gives you a feel or intuition of how the data is arranged in a high-dimensional space.
t-SNE is a tool to visualize high-dimensional data. It converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data. t-SNE has a cost function that is not convex, i.e. with different initializations we can get different results.
It is highly recommended to use another dimensionality reduction method (e.g. PCA for dense data or TruncatedSVD for sparse data) to reduce the number of dimensions to a reasonable amount (e.g. 50) if the number of features is very high. This will suppress some noise and speed up the computation of pairwise distances between samples.

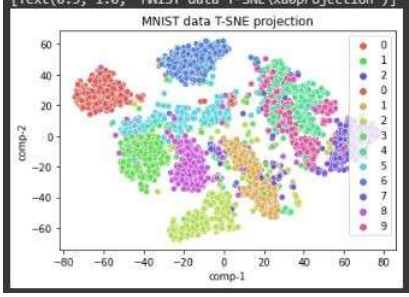
CODE

```
from sklearn.manifold import TSNE
from keras.datasets import mnist
from sklearn.datasets import load_iris
from numpy import reshape
import seaborn as sns
import pandas as pd

iris = load_iris()
x = iris.data
y = iris.target

tsne = TSNE(n_components=2, verbose=1, random_state=123)
z = tsne.fit_transform(x)
df = pd.DataFrame()
df["y"] = y
df["comp-1"] = z[:,0]
df["comp-2"] = z[:,1]

sns.scatterplot(x="comp-1", y="comp-2", hue=df.y.tolist(),
                palette=sns.color_palette("hls", 3),
                data=df).set(title="Iris data T-SNE projection")

(x_train, y_train), (_ , _) = mnist.load_data()
x_train = x_train[:3000]
y_train = y_train[:3000]
print(x_train.shape)

x_mnist = reshape(x_train, [x_train.shape[0], x_train.shape[1]*x_train.shape[2]])
print(x_mnist.shape)

tsne = TSNE(n_components=2, verbose=1, random_state=123)
z = tsne.fit_transform(x_mnist)
```

```
df = pd.DataFrame()
df["y"] = y_train
df["comp-1"] = z[:,0]
df["comp-2"] = z[:,1]

sns.scatterplot(x="comp-1", y="comp-2", hue=df.y.tolist(),
                palette=sns.color_palette("hls", 10),
                data=df).set(title="MNIST data T-SNE projection")
```

OUTPUT

```
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:783: FutureWarning: The default initialization in TSNE will change from 'random' to 'pca' in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:793: FutureWarning: The default learning rate in TSNE will change from 200.0 to 'auto' in 1.2.
  FutureWarning,
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 150 samples in 0.000s...
[t-SNE] Computed neighbors for 150 samples in 0.013s...
[t-SNE] Computed conditional probabilities for sample 150 / 150
[t-SNE] Mean sigma: 0.509910
[t-SNE] KL divergence after 250 iterations with early exaggeration: 50.387669
[t-SNE] KL divergence after 1000 iterations: 0.129141
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
(3000, 28, 28)
(3000, 784)
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 3000 samples in 0.001s...
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:783: FutureWarning: The default initialization in TSNE will change from 'random' to 'pca' in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:793: FutureWarning: The default learning rate in TSNE will change from 200.0 to 'auto' in 1.2.
  FutureWarning,
[t-SNE] Computed neighbors for 3000 samples in 0.413s...
[t-SNE] Computed conditional probabilities for sample 1000 / 3000
[t-SNE] Computed conditional probabilities for sample 2000 / 3000
[t-SNE] Computed conditional probabilities for sample 3000 / 3000
[t-SNE] Mean sigma: 607.882413
[t-SNE] KL divergence after 250 iterations with early exaggeration: 78.365952
[t-SNE] KL divergence after 1000 iterations: 1.271625
[Text(0.5, 1.0, 'MNIST data T-SNE\xa0projection')]
```



RESULT

The implementation was successful.

Experiment 8

AIM
To visualize the data analogies using embedding projectors.

SOFTWARE USED
Google Colaboratory

DATASET USED
IMDB reviews dataset – subwords8k

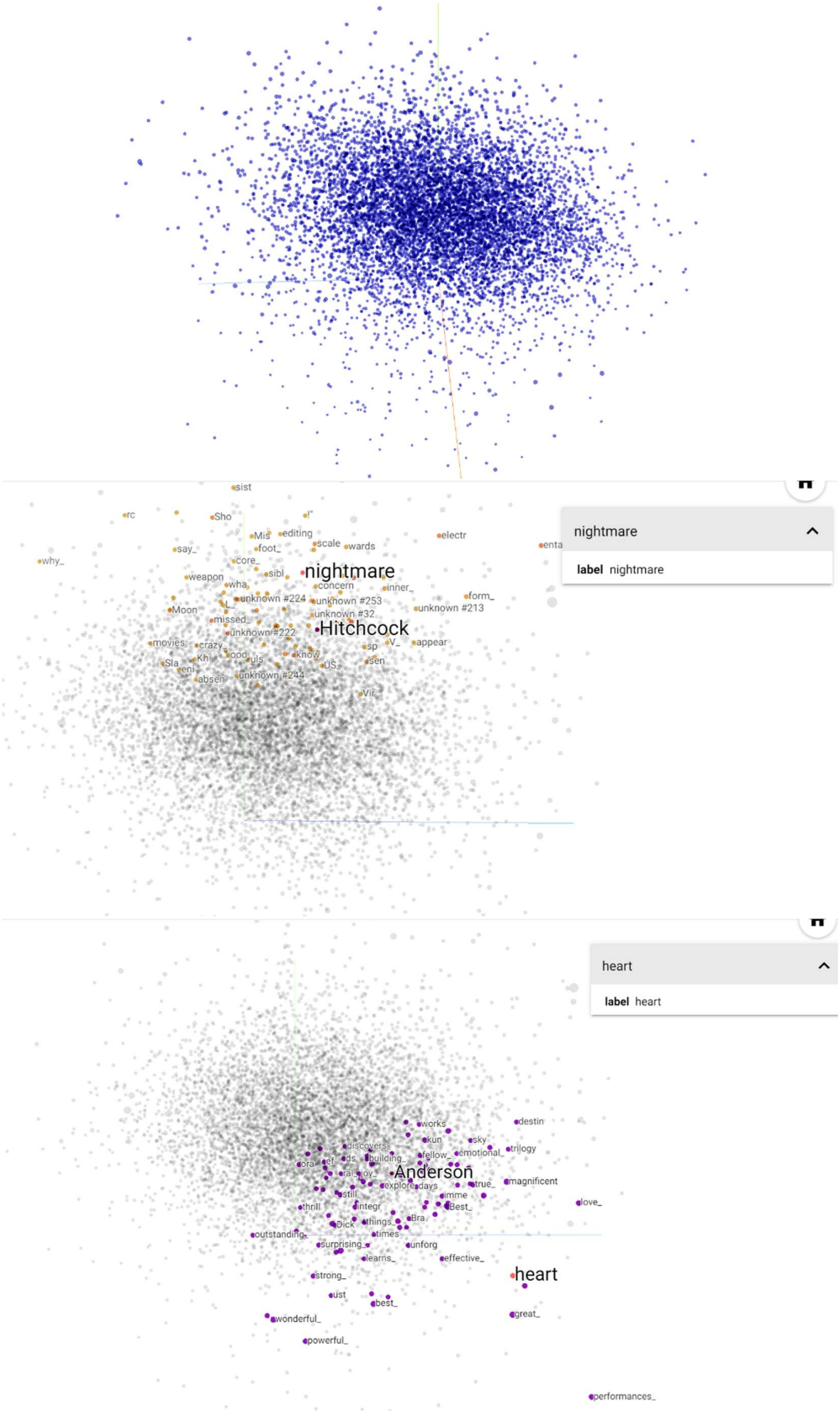
THEORY
The TensorBoard embedding projector is a very powerful tool in data analysis, specifically for interpreting and visualizing low-dimensional embeddings. In order to do so, first, it applies a dimensionality reduction algorithm to the input embeddings, between UMAP, T-SNE, PCA, or a custom one, to reduce their dimension to three and be able to render them in a three-dimensional space. Once the map is generated, this tool can be used, for example, to search for specific keywords associated with the embedding's or highlight similar points in space. Ultimately, its goal is to provide a way to better interpret the embedding's that our machine learning model is generating, to check if the similar ones according to our definition are plotted nearby in the 3D space.

CODE

```
try:
  # %tensorflow_version only exists in Colab.
  %tensorflow_version 2.x
except Exception:
  pass

%load_ext tensorboard

import os
import tensorflow as tf
import tensorflow_datasets as tfds
from tensorboard.plugins import projector

(train_data, test_data), info = tfds.load(
    "imdb_reviews/subwords8k",
    split=(tfds.Split.TRAIN, tfds.Split.TEST),
    with_info=True,
    as_supervised=True,
)
encoder = info.features["text"].encoder

# Shuffle and pad the data.
train_batches = train_data.shuffle(1000).padded_batch(
    10, padded_shapes=((None,), ())
)
test_batches = test_data.shuffle(1000).padded_batch(
    10, padded_shapes=((None,), ())
)
train_batch, train_labels = next(iter(train_batches))

# Create an embedding layer.
embedding_dim = 16
embedding = tf.keras.layers.Embedding(encoder.vocab_size, embedding_dim)
# Configure the embedding layer as part of a keras model.
model = tf.keras.Sequential(
    [
        embedding, # The embedding layer should be the first layer in a model.
```

```
        tf.keras.layers.GlobalAveragePooling1D(),
        tf.keras.layers.Dense(16, activation="relu"),
        tf.keras.layers.Dense(1),
    ]
)

# Compile model.
model.compile(
    optimizer="adam",
    loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
    metrics=["accuracy"],
)

# Train model for one epoch.
history = model.fit(
    train_batches, epochs=1, validation_data=test_batches, validation_steps=20
)
# Set up a logs directory, so Tensorboard knows where to look for files.
log_dir='/logs/imdb-example/'
if not os.path.exists(log_dir):
    os.makedirs(log_dir)

# Save Labels separately on a line-by-line manner.
with open(os.path.join(log_dir, 'metadata.tsv'), "w") as f:
  for subwords in encoder.subwords:
    f.write("{}\n".format(subwords))
  # Fill in the rest of the labels with "unknown".
  for unknown in range(1, encoder.vocab_size - len(encoder.subwords)):
    f.write("unknown #{}\n".format(unknown))


# Save the weights we want to analyze as a variable. Note that the first
# value represents any unknown word, which is not in the metadata, here
# we will remove this value.
weights = tf.Variable(model.layers[0].get_weights()[0][1:])
# Create a checkpoint from embedding, the filename and key are the
# name of the tensor.
checkpoint = tf.train.Checkpoint(embedding=weights)
checkpoint.save(os.path.join(log_dir, "embedding.ckpt"))

# Set up config.
config = projector.ProjectorConfig()
embedding = config.embeddings.add()
# The name of the tensor will be suffixed by `/.ATTRIBUTES/VARIABLE_VALUE`.
embedding.tensor_name = "embedding/.ATTRIBUTES/VARIABLE_VALUE"
embedding.metadata_path = 'metadata.tsv'
projector.visualize_embeddings(log_dir, config)
# Now run tensorboard against on log data we just saved.
%tensorboard --logdir /logs/imdb-example/
```

OUTPUT



RESULT

The implementation was successful.

Experiment 9

AIM

Implement GloVe using tensorflow gradient descent.

SOFTWARE USED

Google Colaboratory

DATASET/CORPUS USED

```
corpus = [
    "the cat in the hat",
    "the quick brown fox",
    "the lazy dog",
]
sample_words = ["the", "cat", "in", "hat", "quick", "brown", "fox", "lazy", "dog"]
```

THEORY

The experiment involves implementing the Global Vectors for Word Representation (GloVe) algorithm using TensorFlow and gradient descent. GloVe is an unsupervised learning technique for generating word embeddings, representing words in a continuous vector space based on their co-occurrence statistics in a corpus. TensorFlow, an open-source machine learning library, is utilized to define and train the GloVe model. The model is constructed with embedding layers for target and context words, and gradient descent is employed to optimize the model parameters, minimizing the difference between predicted and actual co-occurrence probabilities. The resulting word embeddings capture semantic relationships between words, and their interpretation can provide insights into the underlying linguistic structure of the corpus. This experiment provides a practical understanding of the GloVe algorithm, TensorFlow implementation, and the role of gradient descent in training word embeddings for natural language processing tasks.

CODE

```
import tensorflow as tf
import numpy as np
from keras.models import Model
from keras.layers import Input, Embedding, Dot, Reshape

# Sample corpus
corpus = [
    "the cat in the hat",
    "the quick brown fox",
    "the lazy dog",
    # Add more sentences as needed
]

# Tokenize words
tokenizer = tf.keras.preprocessing.text.Tokenizer()
tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1

# Generate word pairs for context and target words
def generate_word_pairs(corpus, window_size=1):
    word_pairs = []
    for sentence in corpus:
        words = tokenizer.texts_to_sequences([sentence])[0]
        for i, target_word in enumerate(words):
```

```
            for context_word in words[max(0, i - window_size) : i + window_size]:
                if context_word != target_word:
                    word_pairs.append([target_word, context_word])
    return np.array(word_pairs)

# Build the GloVe model
embedding_size = 50  # Choose an appropriate size for your embeddings
context_size = 2  # Context window size

input_target = Input(shape=(1,))
input_context = Input(shape=(1,))

embedding = Embedding(total_words, embedding_size, input_length=1)(input_target)
context_embedding = Embedding(total_words, embedding_size, input_length=1)(input_context)

dot_product = Dot(axes=2)([embedding, context_embedding])
dot_product = Reshape((1,))(dot_product)

# Define the GloVe model
glove_model = Model(inputs=[input_target, input_context], outputs=dot_product)
glove_model.compile(optimizer="adam", loss="mean_squared_error")

# Generate training data
word_pairs = generate_word_pairs(corpus, window_size=context_size)
target = np.array([pair[0] for pair in word_pairs], dtype="int32")
context = np.array([pair[1] for pair in word_pairs], dtype="int32")
labels = np.array([1.0] * len(word_pairs))

# Train the model
glove_model.fit([target, context], labels, epochs=100, batch_size=32)

# Extract word embeddings
word_embeddings = glove_model.get_layer("embedding").get_weights()[0]

# Now, word_embeddings contains the trained GloVe embeddings
word_embeddings

# Sample words for interpretation
sample_words = ["the", "cat", "in", "hat", "quick", "brown", "fox", "lazy", "dog"]

# Create a dictionary to store word embeddings
word_embedding_dict = {}
for word in sample_words:
    word_index = tokenizer.word_index[word]
    word_embedding = word_embeddings[word_index]
    word_embedding_dict[word] = word_embedding

# Print the word embeddings
for word, embedding in word_embedding_dict.items():
    print(f"{word}: {embedding}")
```

OUTPUT

```
[3] the: [-0.13559745  0.13986742  0.09990193 -0.02728223 -0.16652799 -0.10410953
  0.10235941 -0.11882525  0.13646148  0.09205011  0.15423088 -0.13670278
  0.11939931  0.10564298 -0.14911763 -0.16938724  0.15442142  0.10704798
  0.16037163 -0.11131185 -0.12112981 -0.11738084  0.14015874 -0.15223126
  0.15391329  0.14183487 -0.11505135  0.1162683  -0.08307496  0.12118861
  0.14915636  0.14954105  0.10461583  0.11644858  0.08718924  0.13985723
 -0.12490056 -0.06064502  0.13727891 -0.11223114 -0.15250792 -0.15971738
  0.10271975 -0.14162458  0.1683922   0.15837696  0.1650399  -0.15990724
  0.12750906 -0.1271096 ]
cat: [-0.14928666  0.11549943  0.13652483  0.1122572  -0.16353552  0.06756092
  0.15149362 -0.10322514  0.11409543  0.14993636 -0.16653559  0.12831257
  0.15037337  0.16811304 -0.12538877 -0.14168267  0.14784168 -0.02596062
  0.10879837  0.16144928 -0.00637459 -0.09580443  0.12715404  0.1295661
  0.09505559  0.07303328 -0.08666128 -0.11936042  0.11307866  0.12823392
 -0.15479355 -0.01673919  0.12028623 -0.15988426  0.10652097  0.09353482
 -0.1325447  -0.13804424  0.03679692 -0.09645607 -0.12888972 -0.15713723
  0.10717132 -0.05092543  0.0903785  -0.126365    0.08868318 -0.11563236
  0.11157608  0.08008594]
in: [-0.05398462  0.10063525  0.09435897  0.13003835 -0.08685909 -0.11975079
  0.10285149 -0.11055315  0.12769684  0.14537671 -0.13148291 -0.13177173
  0.13314189  0.15935098 -0.05783899 -0.12205986  0.16101977  0.05166857
  0.15644976  0.06475307 -0.06328317  0.10754517  0.10821843  0.15821095
  0.1055588   0.16484162 -0.13778499  0.13577685 -0.08469302  0.10543764
  0.0118636   0.03097769 -0.00687477 -0.05477075  0.09929983  0.11274143
 -0.08701902 -0.15425731 -0.1436499  -0.0908851  -0.14456367 -0.10637022
  0.16245557  0.1529198   0.14021994  0.01743805  0.09755721 -0.13406254
  0.13485044  0.03270819]
hat: [-0.10581668  0.06642379  0.14099973  0.1284709  -0.14354283  0.12187631
  0.11381733 -0.12684487  0.11688402  0.08547553 -0.15994108  0.09126251
  0.08730674  0.16720426 -0.06623451 -0.15735058  0.0977344   0.0072201
  0.12852992  0.1485314  -0.0511241  -0.1039098   0.08653402  0.11501122
  0.13110934  0.0850236  -0.11263465 -0.14466435  0.14319384  0.14531551
 -0.09980932  0.01311503  0.10384103 -0.10779047  0.13128261  0.14562358
 -0.15375236 -0.11673122  0.12141839 -0.08093201 -0.15670142 -0.14567378
  0.13730432 -0.07121039  0.14135228 -0.16755448  0.06696989 -0.16736384
  0.08898936  0.03889303]
quick: [ 0.06167478  0.18143122  0.15850656  0.08186644  0.13607688 -0.13321047
  0.10388091 -0.09620838 -0.04573585  0.13581364 -0.12310304 -0.08470585
  0.166814   -0.10072432 -0.14401948 -0.12509018  0.13053872 -0.12981224
  0.08659475  0.0637208  -0.14259185  0.11728773  0.09304056  0.06920902
  0.15111835  0.12219233 -0.14923672  0.02637286  0.08731576  0.08371727
 -0.14463459  0.15800303  0.08970069 -0.1354738   0.15364186  0.15237518
 -0.1160382  -0.05909456 -0.11623694 -0.11899624  0.1196941   0.13400733
  0.12668717  0.15373766  0.12000068 -0.1679968  -0.06258024 -0.15814327
  0.12036532 -0.11381164]
```

```
[3] brown: [ 0.10436802  0.10990199 -0.11780477  0.15640703 -0.1157776  -0.15569198
 -0.1211462  -0.131815   -0.12154003  0.1562483  -0.15114327 -0.14297253
  0.07965443 -0.11998425 -0.10342474 -0.12345421 -0.11272462  0.15126173
  0.08407181  0.15619662 -0.11085679 -0.07015147  0.15286592  0.14220907
  0.14291863  0.16144556 -0.16637716  0.08690059  0.15353201  0.07581947
  0.07474918  0.12488483 -0.09805993 -0.16358756  0.12854078  0.09060557
 -0.11482012 -0.09794067  0.13304305 -0.08098911  0.16412818 -0.02000841
  0.14118275  0.15445597  0.11321583 -0.1115113  -0.15541066 -0.11469325
  0.12451593  0.11238744]
fox: [ 0.14701939  0.09219493  0.13443534  0.1640097   0.11895956 -0.08420773
 -0.00986594 -0.16933559 -0.13446514  0.12009328 -0.12681052 -0.09009045
  0.16882697 -0.11820769 -0.1476168  -0.08507994 -0.16626933  0.07983944
 -0.10574045  0.08760214 -0.11623067 -0.16420974 -0.09908992 -0.09937419
  0.03309499  0.03376168 -0.08982676  0.10104119  0.10559034  0.15835604
  0.16193692  0.0777239   0.13856304 -0.06784928  0.08918408  0.11776643
 -0.13354227  0.13008447  0.08114162 -0.16514295  0.12497151  0.10447165
  0.1506413   0.1601158   0.15311189 -0.15515092 -0.13807982 -0.09152925
  0.14847316 -0.13727178]
lazy: [-0.1578617  -0.0880269   0.03773213  0.11614536 -0.13551867 -0.11661079
 -0.07461704 -0.12336148  0.12762047  0.12027603 -0.14340085 -0.08403795
  0.15616316  0.12490605 -0.06687973 -0.12550215  0.08490773 -0.1449506
  0.15150264  0.07833527 -0.12604691  0.15832184  0.14200738  0.12792186
  0.1336452   0.12157423 -0.17200334  0.06216861  0.04967196  0.13173231
 -0.12085041 -0.14238986 -0.14507969 -0.158827    0.07846771  0.14149582
 -0.12169001 -0.16383608 -0.12587965 -0.09840837 -0.16179223 -0.08984502
  0.11040035  0.12425224  0.08800991  0.09204735  0.1720962  -0.11165774
  0.11889309  0.11723846]
dog: [-0.16628991  0.10368244  0.17689967 -0.12706532 -0.05418001 -0.11017081
  0.12413961 -0.16026768  0.1649861   0.12189689 -0.11956696 -0.1293089
  0.16588734  0.11122655 -0.14193958 -0.09299742  0.04163875 -0.14050482
  0.15331616  0.05758605 -0.06465355  0.13246     0.12628125  0.10325813
  0.15141447  0.16896272  0.09746276  0.04036867  0.11243758  0.17767821
  0.13163665  0.10335749  0.07127485 -0.15444438 -0.14909956  0.05795486
 -0.13591999 -0.16849722 -0.10564224 -0.13677143 -0.08176927 -0.09129749
  0.07925633 -0.11735853  0.0864902  -0.07967173  0.14512126 -0.1143282
  0.09439498  0.07484388]
```

RESULT

The implementation was successful.

Experiment 10

AIM

To perform point wise Mutual Information.

SOFTWARE USED

Google Colaboratory

DATASET/CORPUS USED

```
corpus = ['this is a foo bar bar black sheep foo bar bar black sheep foo bar bar black sheep
shep bar bar black sentence']
```

THEORY

In statistics, probability theory and information theory, pointwise mutual information (PMI), or point mutual information, is a measure of association. It compares the probability of two events occurring together to what this probability would be if the events were independent.

PMI (especially in its positive pointwise mutual information variant) has been described as "one of the most important concepts in NLP", where it "draws on the intuition that the best way to weigh the association between two words is to ask how much more the two words co-occur in a corpus than we would have a priori expected them to appear by chance.

$$PMI(w_1, w_2) = max(\log_2 \frac{p(w_1,w_2)}{p(w_1)p(w_2)}, 0)$$

CODE

```
from collections import Counter, OrderedDict
import numpy as np
import pandas as pd

def calculate_pmi(word1_count, word2_count, cooccur_count, total_count):
    p_word1 = word1_count / total_count
    p_word2 = word2_count / total_count
    p_cooccur = cooccur_count / total_count

    # Check for non-positive values to avoid math domain error
    if p_word1 * p_word2 == 0 or p_cooccur == 0:
        return 0  # Cap negative PMI values to 0
    else:
        pmi = max(0, np.log2(p_cooccur / (p_word1 * p_word2)))  # Use numpy for log2
        return pmi

# Given corpus
corpus = ['this is a foo bar bar black sheep foo bar bar black sheep foo bar bar black sheep
shep bar bar black sentence']

# Tokenize the corpus
tokenized_corpus = corpus[0].split()

print(f"Tokens : {len(tokenized_corpus)}\n")

# Calculate word counts
word_counts = Counter(tokenized_corpus)

# Convert word_counts to DataFrame
word_counts_df = pd.DataFrame.from_dict(word_counts, orient='index', columns=['Count'])
```

```
word_counts_df.index.name = 'Word'

# Print word counts DataFrame
print("Word Counts:")
print(word_counts_df.T)
print()

# Create a co-occurrence matrix with headers
unique_words = list(OrderedDict.fromkeys(tokenized_corpus))  # Preserve order of appearance
cooccur_matrix = np.zeros((len(unique_words), len(unique_words)), dtype=int)

# Populate the co-occurrence matrix
for i in range(len(tokenized_corpus) - 1):
    row_index = unique_words.index(tokenized_corpus[i])
    col_index = unique_words.index(tokenized_corpus[i + 1])
    cooccur_matrix[row_index][col_index] += 1

# Convert co-occurrence matrix to DataFrame
cooccur_df = pd.DataFrame(cooccur_matrix, index=unique_words, columns=unique_words)

# Print the co-occurrence matrix DataFrame
print("Co-occurrence Matrix:")
print(cooccur_df)
print()

# Calculate PMI for all possible word pairs
word_counts = Counter(tokenized_corpus)
cooccur_counts = Counter(zip(tokenized_corpus, tokenized_corpus[1:]))

word_pairs = [(word1, word2) for word1 in word_counts.keys() for word2 in word_counts.keys()
if word1 != word2]

for word1, word2 in word_pairs:
    cooccur_pair = (word1, word2)
    pmi_value = calculate_pmi(word_counts[word1], word_counts[word2],
cooccur_counts[cooccur_pair], len(tokenized_corpus))
    if(pmi_value != 0):
      print(f'PMI between {word1} and {word2}: {pmi_value}')
```

OUTPUT

```
Tokens : 23

Word Counts:
Word   this  is  a  foo  bar  black  sheep  shep  sentence
Count     1   1  1    3    8      4      3     1         1

Co-occurrence Matrix:
          this  is  a  foo  bar  black  sheep  shep  sentence
this         0   1  0    0    0      0      0     0         0
is           0   0  1    0    0      0      0     0         0
a            0   0  0    1    0      0      0     0         0
foo          0   0  0    0    3      0      0     0         0
bar          0   0  0    0    4      4      0     0         0
black        0   0  0    0    0      0      3     0         1
sheep        0   0  0    2    0      0      0     1         0
shep         0   0  0    0    1      0      0     0         0
sentence     0   0  0    0    0      0      0     0         0
```

```
PMI between this and is: 4.523561956057013
PMI between is and a: 4.523561956057013
PMI between a and foo: 2.9385994553358565
PMI between foo and bar: 1.5235619560570128
PMI between bar and black: 1.5235619560570128
PMI between black and sheep: 2.523561956057013
PMI between black and sentence: 2.523561956057013
PMI between sheep and foo: 2.3536369546147005
PMI between sheep and shep: 2.9385994553358565
PMI between shep and bar: 1.5235619560570128
```

RESULT

The implementation was successful.

Experiment 11

AIM

Implement Recursive Neural Tensor Network using tensorflow.

SOFTWARE USED

Google Colaboratory

THEORY

The Recursive Neural Tensor Network (RNTN) is a neural network architecture designed for structured data, particularly hierarchical structures like parse trees or sequences. It extends recursive neural networks by incorporating a tensor layer, allowing the model to capture intricate interactions between pairs of words. RNTN starts with word vectors, computes tensor-based representations to capture higher-order relationships, and recursively combines these representations to form higher-level structures. The resulting vectors are processed through fully connected layers, introducing non-linearities. In this experiment, the RNTN is implemented using TensorFlow, with key components defined through the Keras API. The objective is to provide hands-on experience in leveraging tensor-based operations to enhance neural network architectures for tasks involving hierarchical relationships in structured data.

CODE

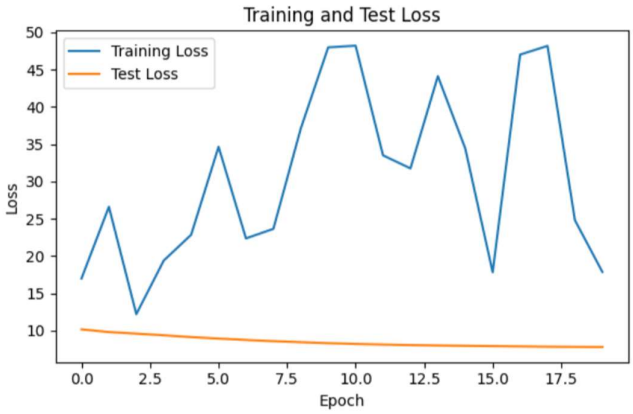
```
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

import tensorflow as tf
import pandas as pd
import matplotlib.pyplot as plt
from keras.models import Model
from keras.layers import Input, Dense, Concatenate, Flatten, Dropout
from keras.initializers import RandomNormal

# Define Recursive Neural Tensor Network (RNTN) model with increased complexity
def build_rntn(vocab_size, embedding_size):
    input_layer = Input(shape=(vocab_size, embedding_size))

    # Word vectors
    word_vectors = Flatten()(input_layer)

    # Tensor layer
    tensor_layer = tf.linalg.matmul(tf.expand_dims(word_vectors, axis=-1),
tf.expand_dims(word_vectors, axis=1))
    tensor_layer = Flatten()(tensor_layer)

    # Concatenate word vectors and tensor layer
    combined_layer = Concatenate()([word_vectors, tensor_layer])

    # Fully connected layers with increased complexity
    fc1 = Dense(50, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
stddev=0.1))(combined_layer)
    fc1 = Dropout(0.5)(fc1)  # Adding dropout for regularization
    fc2 = Dense(25, activation='relu', kernel_initializer=RandomNormal(mean=0.0,
stddev=0.1))(fc1)
```

```
    fc2 = Dropout(0.5)(fc2)
    output_layer = Dense(1, activation='sigmoid', kernel_initializer=RandomNormal(mean=0.0,
stddev=0.1))(fc2)

    model = Model(inputs=input_layer, outputs=output_layer)
    return model

# Example usage with increased batch size for dummy data
vocab_size = 100  # Increased vocabulary size
embedding_size = 20  # Increased embedding size

rntn_model = build_rntn(vocab_size, embedding_size)
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001, clipvalue=1.0)  # Adjusted learning
rate
rntn_model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])

# Generate dummy training data with increased batch size
train_batch_size = 64
train_input_data = tf.random.normal((train_batch_size, vocab_size, embedding_size))
train_labels = tf.random.uniform((train_batch_size, 1), minval=0, maxval=2, dtype=tf.int32)

# Generate dummy test data with increased batch size
test_batch_size = 32
test_input_data = tf.random.normal((test_batch_size, vocab_size, embedding_size))
test_labels = tf.random.uniform((test_batch_size, 1), minval=0, maxval=2, dtype=tf.int32)

# Train the model with increased epochs
history = rntn_model.fit(train_input_data, train_labels, epochs=20,
batch_size=train_batch_size, validation_data=(test_input_data, test_labels), verbose=1)

# Predictions on test data
predictions = rntn_model.predict(test_input_data)

# Create a table of predicted values against test labels
table = pd.DataFrame({'Test Labels': test_labels.numpy().flatten(), 'Predicted Values':
predictions.flatten()})
print("Table of Predicted Values against Test Labels:")
print(table)

# Plot accuracy and loss curves
plt.figure(figsize=(12, 4))

# Plot Training Accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Test Accuracy')
plt.title('Training and Test Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Plot Training Loss
plt.subplot(1, 2, 2)
```

```
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Test Loss')
plt.title('Training and Test Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

OUTPUT

```
1/1 [==============================] - 2s 2s/step
Table of Predicted Values against Test Labels:
    Test Labels  Predicted Values
0             0      8.301851e-01
1             1      1.262085e-11
2             1      9.999568e-01
3             1      9.166635e-01
4             1      1.000000e+00
5             1      1.000000e+00
6             0      7.581893e-18
7             0      8.254212e-01
8             0      1.000000e+00
9             1      1.210851e-06
10            0      5.184276e-03
11            0      1.613476e-07
12            0      1.000000e+00
13            0      1.000000e+00
14            1      9.998938e-01
15            0      9.995377e-01
16            1      2.832852e-09
17            0      1.563346e-06
18            0      2.299066e-10
19            1      1.000000e+00
20            1      3.473338e-13
21            0      1.000000e+00
22            0      2.585898e-08
23            1      9.998140e-01
24            0      7.930070e-01
25            1      1.228318e-09
26            0      4.922176e-05
27            1      1.661944e-03
28            1      9.723052e-01
29            1      7.061279e-11
30            0      9.999881e-01
31            0      1.479091e-05
```



RESULT

The implementation was successful.