Name : Suryansh Srivastava

ID : 124997

Task 2

Report at the end of the notebook

```
import numpy as np
import pickle

import sys
import os
sys.path.append(os.path.abspath('../'))

# importing classes for the respective models
from models.skipgram import Skipgram
from models.skipgram_negSampling import SkipgramNeg
from models.glove import Glove

# Load pickle files
def load(filepath):
    with open(filepath, 'rb') as f:
        return pickle.load(f)

# models name and path

models = {
    "Skipgram": "./models/skipgram_model.pkl",
    "SkipgramNEG": "./models/skipgram_negSampling_model.pkl",
    "Glove": "./models/glove_model.pkl",
    "GloveGensim": "./models/glove_gensim_model.pkl"
}

# index2word name and path for the respective models
index2word={
    "Skipgram": "./models/skipgram_index2word.pkl",
    "SkipgramNEG": "./models/skipgram_negSampling_index2word.pkl",
    "Glove": "./models/glove_index2word.pkl",
}

# word2index name and path for the respective models
word2index={
    "Skipgram": "./models/skipgram_word2index.pkl",
    "SkipgramNEG": "./models/skipgram_negSampling_word2index.pkl",
    "Glove": "./models/glove_word2index.pkl",
}

# Load models
loaded_models = {name: load(path) for name, path in models.items()}

# Load index2word
loaded_index2word = {name: load(path) for name, path in
index2word.items()}

# load word2index
loaded_word2index = {name: load(path) for name, path in
word2index.items()}

# Word analogy dataset url
wordAnalogy_url = "https://www.fit.vutbr.cz/~imikolov/rnnlm/word-
test.v1.txt"

import requests

# syntactic
def fetch_data_syntactic_analogy(url):
    response = requests.get(url)
    response.raise_for_status()
    lines = response.text.strip().split('\n')

    # Extract specific section
    section_start = ': gram7-past-tense'
    section_end = ': gram8-plural'
    extract_lines = []
    in_section = False

    for line in lines:
        if line.startswith(section_start):
            in_section = True
            continue
        elif line.startswith(section_end):
            break

        if in_section:
            extract_lines.append(line)

    return [line.split() for line in extract_lines if line]

# semantic
def fetch_data_semantic_analogy(url):
    response = requests.get(url)
    response.raise_for_status()
    lines = response.text.strip().split('\n')

    # Extract specific section
    section_start = ': capital-common-countries'
    section_end = ': currency'
    extract_lines = []
    in_section = False

    for line in lines:
        if line.startswith(section_start):
            in_section = True
            continue
        elif line.startswith(section_end):
            break

        if in_section:
            extract_lines.append(line)

    return [line.split() for line in extract_lines if line]

syntactic_analogy_data = fetch_data_syntactic_analogy(wordAnalogy_url)
semantic_analogy_data = fetch_data_semantic_analogy(wordAnalogy_url)

import torch

def predict_analogy(model_name,word_a, word_b, word_c,
embeddings=None, word_to_idx=None, idx_to_word=None):
    if(model_name == "GloveGensim"):
        result =
loaded_models['GloveGensim'].most_similar(positive=[word_c, word_b],
negative=[word_a])
        return result[0][0]

    try:
        vec_a = embeddings[word_to_idx[word_a]]
        vec_b = embeddings[word_to_idx[word_b]]
        vec_c = embeddings[word_to_idx[word_c]]
        target_vec = vec_b - vec_a + vec_c

        similarities = torch.matmul(embeddings, target_vec) / (
            torch.norm(embeddings, dim=1) * torch.norm(target_vec) +
1e-8
        )
        best_match_idx = torch.argmax(similarities).item()
        return idx_to_word[best_match_idx]
    except KeyError as e:
        return None  # Return None if any word is not in the
vocabulary


def semantic_accuracy(model_name,analogy_data, embeddings=None,
word_to_idx=None, idx_to_word=None):
    correct = 0
    total = 0
    for question in analogy_data:
        if len(question) != 4:
            continue
        word_a, word_b, word_c, word_d = question
        predicted_word=None
        if(model_name == "GloveGensim"):
            try:
                predicted_word = predict_analogy(model_name,word_a,
word_b, word_c)
            except:
                predicted_word = None
        else:
            predicted_word = predict_analogy(model_name,word_a,
word_b, word_c, embeddings, word_to_idx, idx_to_word)

        if predicted_word == word_d:
            correct += 1

        total += 1

    accuracy = correct / total if total > 0 else 0
    return accuracy

def syntactic_accuracy(model_name,analogy_data, embeddings=None,
word_to_idx=None, idx_to_word=None):
    correct = 0
    total = 0

    for question in analogy_data:
        if len(question) != 4:
            continue
        word_a, word_b, word_c, word_d = question
        # Process syntactic relationships directly from the dataset
        if word_a.endswith("ing") or word_a.endswith("ed"):
            predicted_word=None
            if(model_name == "GloveGensim"):
                try:
                    predicted_word =
predict_analogy(model_name,word_a, word_b, word_c)
                except:
                    predicted_word = None
            else:
                predicted_word = predict_analogy(model_name,word_a,
word_b, word_c, embeddings, word_to_idx, idx_to_word)

            if predicted_word == word_d:
                correct += 1

            total += 1
    syntactic_accuracy = correct / total if total > 0 else 0
    return syntactic_accuracy

for model_name, model in loaded_models.items():
    syntactic_acc = None
    semantic_acc = None
    if(model_name == "GloveGensim"):
        syntactic_acc =
syntactic_accuracy(model_name,syntactic_analogy_data)
        semantic_acc =
semantic_accuracy(model_name,semantic_analogy_data)
    else:

        if(model_name == "Glove"):
            center_embeddings = model.center_embedding.weight.data
            outside_embeddings = model.outside_embedding.weight.data
        else:
            center_embeddings = model.embedding_center.weight.data
            outside_embeddings = model.embedding_outside.weight.data

        word_to_idx = loaded_word2index[model_name]
        idx_to_word = loaded_index2word[model_name]

        syntactic_acc =
syntactic_accuracy(model_name,syntactic_analogy_data,
center_embeddings, word_to_idx, idx_to_word)
        semantic_acc =
semantic_accuracy(model_name,semantic_analogy_data, center_embeddings,
word_to_idx, idx_to_word)

    print(f"{model_name} Model")
    print(f"Syntactic Accuracy: {syntactic_acc * 100:.2f}%")
    print(f"Semantic Accuracy: {semantic_acc * 100:.2f}%")
    print("\n")
```

```
Skipgram Model
Syntactic Accuracy: 0.00%
Semantic Accuracy: 0.00%


SkipgramNEG Model
Syntactic Accuracy: 0.00%
Semantic Accuracy: 0.00%


Glove Model
Syntactic Accuracy: 0.00%
Semantic Accuracy: 0.00%
```

```
GloveGensim Model
Syntactic Accuracy: 0.32%
Semantic Accuracy: 0.00%
```

MSE

```
# dataset
with open('./dataset/wordsim_similarity_goldstandard.txt', 'r') as f:
    content = f.readlines()

def get_embed(model_name,word):
    word2index = loaded_word2index[model_name]
    try:
        index = word2index[word]
    except:
        index = word2index['<UNK>']

    word = torch.LongTensor([word2index[word]])
    embed_c=None
    embed_o=None
    if(model_name == "Glove"):
        embed_c = loaded_models[model_name].center_embedding(word)
        embed_o = loaded_models[model_name].outside_embedding(word)
    else:
        embed_c = loaded_models[model_name].embedding_center(word)
        embed_o = loaded_models[model_name].embedding_outside(word)
    embed   = (embed_c + embed_o) / 2

    return embed[0][0].item(), embed[0][1].item()

def get_dot_product(A, B):
    dot_product = np.dot(A, B)
    return dot_product

model_names = ["Skipgram","SkipgramNEG","Glove","GloveGensim"]
for model_name in model_names:
    similarity = []
    for line in lines:
        word1 = line[0]
        word2 = line[1]
        score = float(line[2])

        try:
            if(model_name == "GloveGensim"):

similarity.append(get_dot_product(model.get_vector(word1),
model.get_vector(word2)))
```

```
            else:

similarity.append(get_dot_product(get_embed(model_name,word1),
get_embed(model_name,word2)))
        except:
            similarity.append(0.0)

    print(f"{model_name} Model")
    # find the Spearman Correlation
    spearman_correlation = spearmanr(similarity, [line[2] for line in
lines])
    print(spearman_correlation)

    # find the MSE
    squared_error = [(similarity[i] - float(lines[i][2]))**2 for i in
range(len(similarity))]
    mse = sum(squared_error) / len(similarity)
    print(mse)
    print("\n")
```

```
Skipgram Model
SignificanceResult(statistic=-0.08945283790183964,
pvalue=0.20437601213430073)
32.53141376806432


SkipgramNEG Model
SignificanceResult(statistic=-0.09749908646575546,
pvalue=0.16639849187449327)
33.057867528332956


Glove Model
SignificanceResult(statistic=-0.053119810535583206,
pvalue=0.4516309214048949)
33.43038299665055


GloveGensim Model
SignificanceResult(statistic=-0.1061521629817607,
pvalue=0.13171888783314104)
28.534438082743094
```

Report

1.Compare Skip-gram, Skip-gram negative sampling, GloVe models on training loss, training time.

2. Use Word analogies dataset to calucalte between syntactic and semantic accuracy, similar to the methods in the Word2Vec and GloVe paper.

- From the four notebooks i.e. 01 - Word2Vec (Skipgram).ipynb, 02 - Word2Vec (Neg Sampling).ipynb, 03 - GloVe from Scratch.ipynb and 04 - GloVe (Gensim).ipynb which are used to train their respective models, we observe the training loss and training time for each model

Model	Window Size	Training Loss	Training time	Syntactic Accuracy	Semantic Accuracy
Skipgram	5	13.209658	28m 36s	0%	0%
Skipgram (NEG)	5	3.428915	28m 46s	0%	0%
Glove	5	0.431736	0m 0s	0%	0%
Glove (Gensim)	5	-	-	0.32%	0%

3. Use the similarity dataset4 to find the correlation between your models' dot product and the provided similarity metrics. (from scipy.stats import spearmanr) Assess if your embeddings correlate with human judgment.

Model	Skipgram	NEG	GloVe	GloVe (gensim)	Y_True
MSE	32.5314	33.0578	33.4303	28.5344	1.0000