Importing necessary libraries

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import re
from collections import Counter
import nltk

nltk.download("brown")

from nltk.corpus import brown
```

```
[nltk_data] Downloading package brown to
[nltk_data]     /Users/anushkaojha/nltk_data...
[nltk_data]   Package brown is already up-to-date!
```

In [2]:
```
#Setting up device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")
```

```
Using device: cpu
```

In [3]:
```
# Ensuring Reproducibility
np.random.seed(42)
```

In [4]:
```
#Loading the "news" category from Brown corpus
corpus = brown.sents(categories="news")
```

In [5]:
```
#Taking a small subset for faster training.
corpus = corpus[:500]
```

In [6]:
```
#Flattening the corpus for vocabulary building
flatten = lambda l: [item for sublist in l for item in sublist]
news_flatten = flatten(corpus)

print("Number of unique tokens:", len(news_flatten))
```

```
Number of unique tokens: 11711
```

In [7]:
```
#Building vocabulary and adding <UNK>
counts = Counter(news_flatten)
vocab = sorted(counts.keys(), key=lambda w: (-counts[w], w))
vocab.append("<UNK>")
```

In [8]:
```
#Mapping the word and index
word2index = {w: i for i, w in enumerate(vocab)}
index2word = {i: w for w, i in word2index.items()}

vocab_size = len(vocab)
print("Vocab size (including <UNK>):", vocab_size)
```

```
Vocab size (including <UNK>): 2947
```

Creating function to generate random training data

In [9]:
```
def random_batch(batch_size, corpus, window_size=2):

    skip_grams = []
    unk = word2index["<UNK>"]


    # Loop through each sentence in the corpus
    for sent in corpus:

        # We start from index = window_size and stop at len(sent) - windo
        # so that each target word has a full context window on both side
        for i in range(window_size, len(sent) - window_size):

            # Center (target) word
            target = word2index[sent[i]]

            # Collect context words within the window
            context = []
            for w in range(1, window_size + 1):
                # Left context
                context.append(word2index[sent[i - w]])
                # Right context
                context.append(word2index[sent[i + w]])

            # Create skip-gram pairs (target, context)
            for ctx_word in context:
                skip_grams.append([target, ctx_word])

    # Randomly sample skip-gram pairs to form a mini-batch
    random_inputs = []
    random_labels = []

    random_index = np.random.choice(
        range(len(skip_grams)), batch_size, replace=False
    )

    for idx in random_index:
        random_inputs.append([skip_grams[idx][0]])  # center word
        random_labels.append([skip_grams[idx][1]])  # context word

    return np.array(random_inputs), np.array(random_labels)
```

In [10]:
```
batch_size = 2
input_batch, target_batch = random_batch(batch_size, corpus, window_size=
print("Input batch:", input_batch)
print("Target batch:", target_batch)
```

```
Input batch: [[   3]
 [1763]]
Target batch: [[1]
 [7]]
```

Skipgram without negative sampling

In [11]:
```
class Skipgram(nn.Module):

    def __init__(self, vocab_size, emb_size):
        super(Skipgram, self).__init__()
        self.embedding_center  = nn.Embedding(vocab_size, emb_size)  # v
        self.embedding_outside = nn.Embedding(vocab_size, emb_size)  # u

    def forward(self, center, outside, all_vocab):
        """
        center:  [batch_size, 1]
        outside: [batch_size, 1]
        all_vocab: [batch_size, vocab_size] (same vocab list repeated acr
        """
        # Get embeddings
        v = self.embedding_center(center)          # [B, 1, D]
        u_o = self.embedding_outside(outside)      # [B, 1, D]
        u_all = self.embedding_outside(all_vocab)  # [B, V, D]

        # Numerator score: u_o dot v
        # [B,1,D] @ [B,D,1] -> [B,1,1] -> squeeze -> [B,1]
        numerator = torch.exp(u_o.bmm(v.transpose(1, 2)).squeeze(2))  # [

        # Denominator: sum_{w in vocab} exp(u_w dot v)
        # [B,V,D] @ [B,D,1] -> [B,V,1] -> squeeze -> [B,V]
        denom_scores = u_all.bmm(v.transpose(1, 2)).squeeze(2)        # [
        denominator = torch.sum(torch.exp(denom_scores), dim=1, keepdim=T

        # Negative log likelihood (scalar loss)
        loss = -torch.mean(torch.log(numerator / denominator))
        return loss
```

In [12]:
```
# Training setup
import torch.optim as optim
batch_size = 2
embedding_size = 2
window_size = 2
model = Skipgram(vocab_size, embedding_size).to(device)
optimizer = optim.Adam(model.parameters(), lr=0.001)

def prepare_sequence(seq, word2index):
    idxs = [
        word2index[w] if word2index.get(w) is not None else word2index["<
        for w in seq
    ]
    return torch.LongTensor(idxs)
```

In [13]:
```
all_vocabs = prepare_sequence(list(vocab), word2index) \
                .expand(batch_size, len(vocab)) \
                .to(device)

print("all_vocabs shape:", all_vocabs.shape)
```

```
all_vocabs shape: torch.Size([2, 2947])
```

In [14]:
```
def epoch_time(start_time, end_time):
    elapsed_time = end_time - start_time
    elapsed_mins = int(elapsed_time / 60)
    elapsed_secs = int(elapsed_time - (elapsed_mins * 60))
    return elapsed_mins, elapsed_secs
```

Training

In [15]:
```
import time

num_epochs = 5000

for epoch in range(num_epochs):

    start = time.time()

    # Get a random mini-batch of Skip-gram pairs
    # window_size is now configurable (Task 1 requirement)
    input_batch, target_batch = random_batch(
        batch_size, corpus, window_size=window_size
    )

    # Convert numpy arrays to torch tensors
    input_batch  = torch.LongTensor(input_batch).to(device)   # [batch_si
    target_batch = torch.LongTensor(target_batch).to(device)  # [batch_si

    # Zero gradients from previous step
    optimizer.zero_grad()

    # Forward pass: compute Skip-gram loss
    loss = model(input_batch, target_batch, all_vocabs)

    # Backpropagation
    loss.backward()

    # Update model parameters
    optimizer.step()

    end = time.time()

    # Compute elapsed time for this epoch
    epoch_time_ms = (end - start) * 1000

    # Print training progress every 1000 epochs (same as professor)
    if (epoch + 1) % 1000 == 0:
        print(
            f"Epoch: {epoch + 1} | "
            f"cost: {loss.item():.6f} | "
            f"time: {epoch_time_ms:.2f} ms"
        )
```

```
Epoch: 1000 | cost: 8.358099 | time: 12.14 ms
Epoch: 2000 | cost: 7.793187 | time: 11.80 ms
Epoch: 3000 | cost: 7.661410 | time: 62.18 ms
Epoch: 4000 | cost: 6.302114 | time: 11.88 ms
Epoch: 5000 | cost: 7.772816 | time: 64.14 ms
```

In [16]:
```
print(vocab[20:30])
```

```
['is', 'as', 'has', 'it', 'not', 'will', 'at', 'with', 'an', 'his']
```

In [23]:
```
# Cosine Similarity for Skip-gram embeddings
from numpy import dot
from numpy.linalg import norm
def get_embed_skip_gram(word):

    idx = word2index.get(word, word2index["<UNK>"])
    idx_tensor = torch.LongTensor([idx]).to(device)

    v_embed = model.embedding_center(idx_tensor)
    u_embed = model.embedding_outside(idx_tensor)

    word_embed = (v_embed + u_embed) / 2.0
    return word_embed.squeeze(0).detach().cpu().numpy()

def cos_sim(a, b):
    return dot(a, b) / (norm(a) * norm(b))
```

In [24]:
```
election = get_embed_skip_gram("election")
vote = get_embed_skip_gram("vote")
campaign = get_embed_skip_gram("campaign")
```

In [25]:
```
print(f"election vs vote:        {cos_sim(election, vote):.4f}")
print(f"election vs campaign:   {cos_sim(election, campaign):.4f}")
print(f"election vs election:   {cos_sim(election, election):.4f}")
```

```
election vs vote:        0.4850
election vs campaign:   -0.0720
election vs election:    1.0000
```

saving the skipgram model without negative sampling

In [26]:
```
# Save the model
import pickle
torch.save(model.state_dict(), 'model/skipgram_model.pth')
pickle.dump(model, open('model/skipgram.pkl', 'wb'))
```

Word2Vec (Negative Sampling)

In [27]:
```
#Building unigram table
Z = 0.001
```

In [28]:
```
flatten = lambda l: [item for sublist in l for item in sublist]
word_count = Counter(flatten(corpus))
num_total_words = sum(word_count.values())
```

In [29]:
```
num_total_words
```

Out[29]:
```
11711
```

In [30]:
```
unigram_table = []
for w in vocab:
    uw = word_count[w] / max(1, num_total_words)     # unigram prob
    uw_alpha = int((uw ** 0.75) / Z)                 # apply 0.75 smoothi
    if uw_alpha > 0:
        unigram_table.extend([w] * uw_alpha)

print("Unigram table size:", len(unigram_table))
```

```
Unigram table size: 3623
```

In [31]:
```
Counter(unigram_table)
```

```
Out[31]: Counter({'the': 120,
                  ',': 87,
                  '.': 85,
                  'of': 75,
                  'to': 65,
                  'a': 52,
                  'in': 49,
                  'and': 46,
                  '``': 33,
                  'for': 33,
                  'that': 33,
                  "''": 33,
                  'The': 28,
                  'said': 27,
                  'be': 25,
                  'would': 24,
                  'on': 24,
                  'was': 23,
                  'by': 22,
                  'he': 21,
                  'is': 21,
                  'as': 18,
                  'has': 17,
                  'it': 15,
                  'not': 15,
                  'will': 14,
                  'at': 14,
                  'with': 14,
                  'an': 13,
                  'his': 13,
                  'been': 13,
                  'which': 12,
                  'He': 12,
                  'this': 11,
                  '--': 11,
                  'Mr.': 11,
                  'more': 11,
                  'have': 10,
                  'who': 10,
                  'from': 10,
                  'President': 9,
                  'administration': 9,
                  'its': 9,
                  'year': 9,
                  'are': 9,
                  'had': 9,
                  'or': 9,
                  'State': 9,
                  'Texas': 9,
                  'election': 9,
                  'jury': 9,
                  'last': 8,
                  'there': 8,
                  'House': 8,
                  'It': 8,
                  'city': 8,
                  'first': 8,
                  'plan': 8,
                  'state': 8,
                  'than': 8,
                  'were': 8,
                  'bill': 7,
                  'made': 7,
                  'new': 7,
                  'no': 7,
                  'should': 7,
                  'up': 7,
                  'also': 7,
                  'federal': 7,
                  'million': 7,
                  'other': 7,
                  'program': 7,
                  'A': 7,
                  'but': 7,
                  'home': 7,
                  'one': 7,
                  'pay': 7,
                  'schools': 7,
                  'they': 7,
                  '(': 6,
                  ')': 6,
                  'Kennedy': 6,
                  'committee': 6,
                  'medical': 6,
                  'some': 6,
                  'under': 6,
                  'Fulton': 6,
                  'United': 6,
                  'council': 6,
                  'law': 6,
                  'out': 6,
                  'resolution': 6,
                  'them': 6,
                  'County': 6,
                  'Dallas': 6,
                  'I': 6,
                  'Laos': 6,
                  'States': 6,
                  'after': 6,
                  'such': 6,
                  'these': 6,
                  'time': 6,
                  'vote': 6,
                  'But': 5,
                  'Sunday': 5,
                  'being': 5,
                  'care': 5,
                  'cases': 5,
                  'court': 5,
                  'funds': 5,
                  'government': 5,
                  'grants': 5,
                  'increase': 5,
                  'local': 5,
                  'public': 5,
                  'school': 5,
                  'take': 5,
                  'their': 5,
                  'when': 5,
                  'In': 5,
                  'Sen.': 5,
                  'Senate': 5,
                  'This': 5,
                  'aid': 5,
                  'any': 5,
                  'campaign': 5,
                  'charter': 5,
                  'did': 5,
                  'director': 5,
                  'general': 5,
                  'if': 5,
                  'into': 5,
                  'now': 5,
                  'over': 5,
                  'per': 5,
                  'told': 5,
                  'years': 5,
                  '1': 4,
                  'Department': 4,
                  'Hawksley': 4,
                  'Republican': 4,
                  'all': 4,
                  'called': 4,
                  'can': 4,
                  'days': 4,
                  'most': 4,
                  'must': 4,
                  'provide': 4,
                  'tax': 4,
                  'two': 4,
                  'what': 4,
                  '10': 4,
                  'Congress': 4,
                  'Eisenhower': 4,
                  'Republicans': 4,
                  'against': 4,
                  'asked': 4,
                  'before': 4,
                  'bonds': 4,
                  'both': 4,
                  'county': 4,
                  'day': 4,
                  'dollars': 4,
                  'each': 4,
                  'go': 4,
                  'issue': 4,
                  'may': 4,
                  'night': 4,
                  'passed': 4,
                  'persons': 4,
                  'present': 4,
                  'special': 4,
                  ':': 4,
                  'ADC': 4,
                  'Committee': 4,
                  'Council': 4,
                  'J.': 4,
                  'Monday': 4,
                  'One': 4,
                  'Rep.': 4,
                  'There': 4,
                  'Washington': 4,
                  'about': 4,
                  'cent': 4,
                  'chairman': 4,
                  'could': 4,
                  'expected': 4,
                  'further': 4,
                  'governor': 4,
                  'health': 4,
                  'make': 4,
                  'meeting': 4,
                  'people': 4,
                  'policy': 4,
                  'possible': 4,
                  'rule': 4,
                  'since': 4,
                  'social': 4,
                  'work': 4,
                  'yesterday': 4,
                  'Austin': 3,
                  'CD': 3,
                  'City': 3,
                  'Communist': 3,
                  'Dr.': 3,
                  'Karns': 3,
                  'Martinelli': 3,
                  'NATO': 3,
                  'Party': 3,
                  'added': 3,
                  'back': 3,
                  'because': 3,
                  'between': 3,
                  'bills': 3,
                  'case': 3,
                  'costs': 3,
                  'defense': 3,
                  'do': 3,
                  'efforts': 3,
                  'given': 3,
                  'involved': 3,
                  'laws': 3,
                  'need': 3,
                  'only': 3,
                  'primary': 3,
                  'proposal': 3,
                  'proposed': 3,
                  'security': 3,
                  'session': 3,
                  'system': 3,
                  'taken': 3,
                  'those': 3,
                  'through': 3,
                  'town': 3,
                  'ward': 3,
                  "'": 3,
                  '?': 3,
                  'B.': 3,
                  'Clark': 3,
                  'College': 3,
                  'Communists': 3,
                  'Education': 3,
                  'General': 3,
                  'Legislature': 3,
                  'Providence': 3,
                  'Secretary': 3,
                  'Socialist': 3,
                  'Thursday': 3,
                  'Wexler': 3,
                  'White': 3,
                  'another': 3,
                  'attack': 3,
                  'board': 3,
                  'candidate': 3,
                  'civil': 3,
                  'cost': 3,
                  'dental': 3,
                  'even': 3,
                  'full-time': 3,
                  'future': 3,
                  'get': 3,
                  'give': 3,
                  'grand': 3,
                  'half': 3,
                  'legislators': 3,
                  'much': 3,
                  'our': 3,
                  'payroll': 3,
                  'plans': 3,
                  'president': 3,
                  'problems': 3,
                  'property': 3,
                  'race': 3,
                  'received': 3,
                  'report': 3,
                  'research': 3,
                  'sales': 3,
                  'similar': 3,
                  'statements': 3,
                  'states': 3,
                  'study': 3,
                  'trial': 3,
                  'very': 3,
                  'where': 3,
                  'without': 3,
                  '24': 2,
                  'A.': 2,
                  'Atlanta': 2,
                  'Bellows': 2,
                  'Citizens': 2,
                  'Court': 2,
                  'Democratic': 2,
                  'District': 2,
                  'East': 2,
                  'Hartsfield': 2,
                  'Highway': 2,
                  'His': 2,
                  'Hughes': 2,
                  'Jones': 2,
                  'Jr.': 2,
                  'Judge': 2,
                  'Oslo': 2,
                  'Parkhouse': 2,
                  'Pelham': 2,
                  'Reama': 2,
                  'They': 2,
                  'Williams': 2,
                  'act': 2,
                  'age': 2,
                  'aged': 2,
                  'approved': 2,
                  'attorney': 2,
                  'ballot': 2,
                  'become': 2,
                  'better': 2,
                  'candidates': 2,
                  'come': 2,
                  'counties': 2,
                  'deaf': 2,
                  'defendants': 2,
                  'degree': 2,
                  'evidence': 2,
                  'former': 2,
                  'help': 2,
                  'hospital': 2,
                  'information': 2,
                  'leadership': 2,
                  'like': 2,
                  'long': 2,
                  'means': 2,
                  'meet': 2,
                  'members': 2,
                  'military': 2,
                  'national': 2,
                  'next': 2,
                  'number': 2,
                  'nursing': 2,
                  'petition': 2,
                  'place': 2,
                  'police': 2,
                  'policies': 2,
                  'political': 2,
                  'precinct': 2,
                  'problem': 2,
                  'recommendations': 2,
                  'recommended': 2,
                  'rural': 2,
                  'scheduled': 2,
                  'several': 2,
                  'signatures': 2,
                  'so': 2,
                  'step': 2,
                  'taxes': 2,
                  'teacher': 2,
                  'then': 2,
                  'three': 2,
                  'toward': 2,
                  'upon': 2,
                  'voters': 2,
                  'water': 2,
                  'we': 2,
                  'weeks': 2,
                  '2': 2,
                  '23d': 2,
                  '30': 2,
                  '4': 2,
                  'Assembly': 2,
                  'Central': 2,
                  'Charles': 2,
                  'Cotten': 2,
                  'Criminal': 2,
                  'Cuba': 2,
                  'Daniel': 2,
                  'Friday': 2,
                  'George': 2,
                  'Georgia': 2,
                  "Georgia's": 2,
                  'Gov.': 2,
                  'Group': 2,
                  'Grover': 2,
                  'Hospital': 2,
                  'Island': 2,
                  'James': 2,
                  'John': 2,
                  'Johnston': 2,
                  'Lao': 2,
                  'M.': 2,
                  'Mitchell': 2,
                  'Notte': 2,
                  'Rhode': 2,
                  'Some': 2,
                  'Vandiver': 2,
                  'action': 2,
                  'address': 2,
                  'ago': 2,
                  'amount': 2,
                  'annual': 2,
                  'appointment': 2,
                  'banks': 2,
                  'believes': 2,
                  'charged': 2,
                  'child': 2,
                  'children': 2,
                  'cities': 2,
                  'citizens': 2,
                  'commission': 2,
                  'community': 2,
                  'concerned': 2,
                  'conference': 2,
                  'courses': 2,
                  'declared': 2,
                  'despite': 2,
                  'does': 2,
                  'education': 2,
                  'elected': 2,
                  'enabling': 2,
                  'enough': 2,
                  'establishment': 2,
                  'ever': 2,
                  'five': 2,
                  'got': 2,
                  'group': 2,
                  'heard': 2,
                  'here': 2,
                  'high': 2,
                  'judges': 2,
                  'large': 2,
                  'later': 2,
                  'left': 2,
                  'legislation': 2,
                  'limited': 2,
                  'listed': 2,
                  'major': 2,
                  'making': 2,
                  'man': 2,
                  'matter': 2,
                  'mean': 2,
                  'message': 2,
                  'might': 2,
                  'millions': 2,
                  'money': 2,
                  'movement': 2,
                  'needs': 2,
                  'never': 2,
                  'obtain': 2,
                  'obtained': 2,
                  'opposition': 2,
                  'order': 2,
                  'paid': 2,
                  'parties': 2,
                  'past': 2,
                  'patient': 2,
                  'person': 2,
                  'petitions': 2,
                  'precincts': 2,
                  'programs': 2,
                  'proposals': 2,
                  'question': 2,
                  'railroad': 2,
                  'receive': 2,
                  'relations': 2,
                  'reported': 2,
                  'residents': 2,
                  'right': 2,
                  'service': 2,
                  'services': 2,
                  'set': 2,
                  'seven': 2,
                  'soon': 2,
                  'source': 2,
                  'studied': 2,
                  'suggested': 2,
                  'term': 2,
                  'track': 2,
                  'use': 2,
                  'voted': 2,
                  'votes': 2,
                  'week': 2,
                  'welfare': 2,
                  'whether': 2,
                  'willing': 2,
                  'within': 2,
                  'workers': 2,
                  'working': 2,
                  'yet': 2,
                  '13': 2,
                  '1910': 2,
                  '20': 2,
                  '3': 2,
                  '65': 2,
                  '70': 2,
                  'After': 2,
                  'American': 2,
                  'Authority': 2,
                  'Barber': 2,
                  'Berlin': 2,
                  'Berry': 2,
                  'Bourcier': 2,
                  'Bush': 2,
                  'C.': 2,
                  'Club': 2,
                  'Cook': 2,
                  'D.': 2,
                  'Davis': 2,
                  'E.': 2,
                  'Executive': 2,
                  'Falls': 2,
                  'Feb.': 2,
                  'Foreign': 2,
                  'Fort': 2,
                  'GOP': 2,
                  'Geneva': 2,
                  'Grant': 2,
                  'H.': 2,
                  'Health': 2,
                  'Houston': 2,
                  'If': 2,
                  'Jan.': 2,
                  'July': 2,
                  'March': 2,
                  'Martin': 2,
                  'Massachusetts': 2,
                  'Miss': 2,
                  'New': 2,
                  'Nixon': 2,
                  'Oklahoma': 2,
                  'Organization': 2,
                  'Other': 2,
                  'P.': 2,
                  'Paris': 2,
                  'Parsons': 2,
                  "President's": 2,
                  'R.': 2,
                  'Ratcliff': 2,
                  'Republicanism': 2,
                  'Roberts': 2,
                  "Rusk's": 2,
                  'Sam': 2,
                  'School': 2,
                  'Souvanna': 2,
                  'Soviet': 2,
                  'Superior': 2,
                  'These': 2,
                  'University': 2,
                  'W.': 2,
                  'Welfare': 2,
                  'When': 2,
                  'While': 2,
                  'William': 2,
                  'accept': 2,
                  'achieve': 2,
                  'actions': 2,
                  'again': 2,
                  'alliance': 2,
                  'allowed': 2,
                  'amendment': 2,
                  'announced': 2,
                  'anonymous': 2,
                  'apparent': 2,
                  'approval': 2,
                  'argued': 2,
                  'ask': 2,
                  'attend': 2,
                  'audience': 2,
                  'authority': 2,
                  'bankers': 2,
                  'billion': 2,
                  'blue': 2,
                  'bond': 2,
                  'brought': 2,
                  'business': 2,
                  'calls': 2,
                  'career': 2,
                  'carry': 2,
                  'caused': 2,
                  'causes': 2,
                  'changes': 2,
                  'charge': 2,
                  'chief': 2,
                  'college': 2,
                  'companies': 2,
                  'constituted': 2,
                  'constitutional': 2,
                  'construction': 2,
                  'controversy': 2,
                  'countries': 2,
                  'critical': 2,
                  'debate': 2,
                  'denied': 2,
                  'designed': 2,
                  "didn't": 2,
                  'discrimination': 2,
                  'doctor': 2,
                  'dollar': 2,
                  'down': 2,
                  'during': 2,
                  'early': 2,
                  'effective': 2,
                  'eight': 2,
                  'eliminating': 2,
                  'employment': 2,
                  'enforcement': 2,
                  'essential': 2,
                  'estimated': 2,
                  'fact': 2,
                  'fair': 2,
                  'family': 2,
                  'felt': 2,
                  'fight': 2,
                  'find': 2,
                  'follow': 2,
                  'force': 2,
                  'forces': 2,
                  'foreign': 2,
                  'four': 2,
                  'free': 2,
                  'fund': 2,
                  'generally': 2,
                  'going': 2,
                  'greater': 2,
                  'gubernatorial': 2,
                  'hearing': 2,
                  'held': 2,
                  'highway': 2,
                  'him': 2,
                  'himself': 2,
                  'hold': 2,
                  'homes': 2,
                  'hours': 2,
                  'however': 2,
                  'immediate': 2,
                  'including': 2,
                  'indicated': 2,
                  'instead': 2,
                  'insurance': 2,
                  'interested': 2,
                  'investigation': 2,
                  'irregularities': 2,
                  'issued': 2,
                  'juvenile': 2,
                  'little': 2,
                  'manner': 2,
                  'many': 2,
                  'matching': 2,
                  "mayor's": 2,
                  'me': 2,
                  'medicine': 2,
                  'mention': 2,
                  'mind': 2,
                  'months': 2,
                  'my': 2,
                  "nation's": 2,
                  'nine': 2,
                  'none': 2,
                  'notice': 2,
                  'nuclear': 2,
                  'offenses': 2,
                  'ones': 2,
                  'opinion': 2,
                  'ordinance': 2,
                  'organization': 2,
                  'own': 2,
                  'party': 2,
                  'personal': 2,
                  'poll': 2,
                  'polls': 2,
                  'practices': 2,
                  'previous': 2,
                  'principal': 2,
                  'procedures': 2,
                  'raises': 2,
                  'ready': 2,
                  'real': 2,
                  'recent': 2,
                  'recommend': 2,
                  'reduce': 2,
                  'remark': 2,
                  'required': 2,
                  'rescue': 2,
                  'retired': 2,
                  'retirement': 2,
                  'scholarships': 2,
                  'semester': 2,
                  'senator': 2,
                  'serious': 2,
                  'served': 2,
                  'situation': 2,
                  'spent': 2,
                  'spokesmen': 2,
                  'statement': 2,
                  'steps': 2,
                  'still': 2,
                  'superintendent': 2,
                  'support': 2,
                  'teaching': 2,
                  'test': 2,
                  'themselves': 2,
                  'today': 2,
                  'too': 2,
                  'trouble': 2,
                  'trucks': 2,
                  'understanding': 2,
                  'urged': 2,
                  'want': 2,
                  'went': 2,
                  'while': 2,
                  'whom': 2,
                  'worth': 2,
                  '$1,000': 1,
                  '$1,500': 1,
                  '$10': 1,
                  '$37': 1,
                  '&': 1,
                  '18': 1,
                  '1961': 1,
                  '1961-62': 1,
                  '1963': 1,
                  '4-year': 1,
                  '50': 1,
                  '58th': 1,
                  '6': 1,
                  '8': 1,
                  '9': 1,
                  ';': 1,
                  'Acting': 1,
                  'Allen': 1,
                  'Angola': 1,
                  'April': 1,
                  'Army': 1,
                  'Association': 1,
                  'At': 1,
                  "Atlanta's": 1,
                  'Attorney': 1,
                  'Aug.': 1,
                  'Bill': 1,
                  'Blue': 1,
                  'Both': 1,
                  'Caldwell': 1,
                  'Calls': 1,
                  'Chairman': 1,
                  'Chapman': 1,
                  'Chief': 1,
                  'Co.': 1,
                  'Colquitt': 1,
                  'Cox': 1,
                  'Crime': 1,
                  'Delinquency': 1,
                  'Denton': 1,
                  "Department's": 1,
                  'Dumont': 1,
                  'During': 1,
                  'El': 1,
                  'Felix': 1,
                  'Fire': 1,
                  'Frank': 1,
                  'Full': 1,
                  'Griffin': 1,
                  'Halleck': 1,
                  'Hays': 1,
                  'Henry': 1,
                  'Hotel': 1,
                  'How': 1,
                  'Institute': 1,
                  'Ivan': 1,
                  'Joe': 1,
                  'Juvenile': 1,
                  'Kan.': 1,
                  "Kennedy's": 1,
                  'L.': 1,
                  'Leader': 1,
                  'Louis': 1,
                  'Mayor': 1,
                  'Miller': 1,
                  'Nam': 1,
                  'North': 1,
                  'Officials': 1,
                  'On': 1,
                  'Only': 1,
                  'Ordinary': 1,
                  'Paradise': 1,
                  'Paso': 1,
                  'Pathet': 1,
                  'Phouma': 1,
                  'Policies': 1,
                  'Port': 1,
                  'Presidential': 1,
                  'Prince': 1,
                  'Raymond': 1,
                  'Research': 1,
                  'Richard': 1,
                  'Ridge': 1,
                  'Riverside': 1,
                  'Roads': 1,
                  'Robert': 1,
                  'Rural': 1,
                  'SEATO': 1,
                  'Salinger': 1,
                  'San': 1,
                  'Science': 1,
                  'Scott': 1,
                  'Seekonk': 1,
                  'Senator': 1,
                  'Sept.': 1,
                  'September': 1,
                  'Several': 1,
                  'Sherman': 1,
                  'South': 1,
                  'Southeast': 1,
                  'Technology': 1,
                  'Thailand': 1,
                  'That': 1,
                  'Town': 1,
                  'Treaty': 1,
                  'Tuesday': 1,
                  'Two': 1,
                  'U.S.': 1,
                  'Under': 1,
                  'Union': 1,
                  'Viet': 1,
                  'Wayne': 1,
                  'We': 1,
                  'Weatherford': 1,
                  'West': 1,
                  'Worth': 1,
                  'York': 1,
                  'You': 1,
                  'able': 1,
                  'absorbed': 1,
                  'acceptable': 1,
                  'according': 1,
                  'addition': 1,
                  'additional': 1,
                  'adjournment': 1,
                  'adopted': 1,
                  'advantage': 1,
                  'advised': 1,
                  'advisement': 1,
                  'agreed': 1,
                  'airport': 1,
                  'allow': 1,
                  'ally': 1,
                  'almost': 1,
                  'alone': 1,
                  'along': 1,
                  'alternative': 1,
                  'among': 1,
                  'apparently': 1,
                  'approve': 1,
                  'area': 1,
                  'areas': 1,
                  'arms': 1,
                  'arrests': 1,
                  'aside': 1,
                  'assembly': 1,
                  'assistance': 1,
                  'assistant': 1,
                  'attempt': 1,
                  'authorities': 1,
                  'authorizing': 1,
                  'available': 1,
                  'bank': 1,
                  'base': 1,
                  'basic': 1,
                  'basis': 1,
                  'begin': 1,
                  'benefit': 1,
                  'benefits': 1,
                  'betting': 1,
                  'big': 1,
                  'bit': 1,
                  'boos': 1,
                  'boost': 1,
                  'brokers': 1,
                  'budget': 1,
                  'build': 1,
                  'building': 1,
                  'burglary': 1,
                  'cabinet': 1,
                  'call': 1,
                  'came': 1,
                  'cannot': 1,
                  'canvassers': 1,
                  'capital': 1,
                  'carcass': 1,
                  'cease-fire': 1,
                  'certain': 1,
                  'certainly': 1,
                  'certificate': 1,
                  'choice': 1,
                  'clear': 1,
                  'client': 1,
                  'combined': 1,
                  'coming': 1,
                  'commented': 1,
                  'confidence': 1,
                  'conflict': 1,
                  'connection': 1,
                  'consulted': 1,
                  'continue': 1,
                  'contracts': 1,
                  'conventional': 1,
                  'coolest': 1,
                  'cooperation': 1,
                  'correspondents': 1,
                  'costly': 1,
                  'country': 1,
                  'couple': 1,
                  'courts': 1,
                  'cover': 1,
                  'credit': 1,
                  'crisis': 1,
                  'danger': 1,
                  'date': 1,
                  'decisions': 1,
                  'defeated': 1,
                  'delay': 1,
                  'delegation': 1,
                  'demand': 1,
                  'department': 1,
                  'departments': 1,
                  'dependency': 1,
                  'deputies': 1,
                  'detailed': 1,
                  'details': 1,
                  'developed': 1,
                  'development': 1,
                  'diplomatic': 1,
                  'directed': 1,
                  'disagreed': 1,
                  'disclosure': 1,
                  'districts': 1,
                  'divorce': 1,
                  'done': 1,
                  'drafts': 1,
                  'duty': 1,
                  'earlier': 1,
                  'economic': 1,
                  'effect': 1,
                  'effort': 1,
                  'elaborate': 1,
                  'eligible': 1,
                  'employed': 1,
                  'end': 1,
                  'endorse': 1,
                  'enforced': 1,
                  'enter': 1,
                  'enterprise': 1,
                  'escheat': 1,
                  'establish': 1,
                  'event': 1,
                  'except': 1,
                  'exception': 1,
                  'executive': 1,
                  'expects': 1,
                  'expense': 1,
                  'explained': 1,
                  'expressed': 1,
                  'facilities': 1,
                  'factor': 1,
                  'far': 1,
                  'favor': 1,
                  'feel': 1,
                  'field': 1,
                  'finally': 1,
                  'finance': 1,
                  'fine': 1,
                  'fire': 1,
                  'firm': 1,
                  'floor': 1,
                  'forced': 1,
                  'form': 1,
                  'formally': 1,
                  'formula': 1,
                  'found': 1,
                  'gas': 1,
                  'gave': 1,
                  'getting': 1,
                  'global': 1,
                  "governor's": 1,
                  'grant': 1,
                  'great': 1,
                  'groups': 1,
                  'growth': 1,
                  'guilt': 1,
                  'hand': 1,
                  'having': 1,
                  'head': 1,
                  'hear': 1,
                  'hearings': 1,
                  'heart': 1,
                  'higher': 1,
                  'hoped': 1,
                  'horse': 1,
                  'hospitals': 1,
                  'house': 1,
                  'housing': 1,
                  'how': 1,
                  'idea': 1,
                  'illness': 1,
                  'impossible': 1,
                  ...})
```

In [32]:
```python
def prepare_sequence(seq, word2index):
    unk = word2index["<UNK>"]
    idxs = [word2index.get(w, unk) for w in seq]
    return torch.LongTensor(idxs)

def negative_sampling(targets, unigram_table, k):
    if targets.dim() == 2:
        targets_1d = targets.squeeze(1)
    else:
        targets_1d = targets

    batch_size = targets_1d.size(0)
    neg_samples = []

    for i in range(batch_size):
        target_index = targets_1d[i].item()
        nsample = []
        while len(nsample) < k:
            neg_word = random.choice(unigram_table)  # sampled token (str
            neg_idx = word2index[neg_word]
            if neg_idx == target_index:
                continue
            nsample.append(neg_word)

        neg_samples.append(prepare_sequence(nsample, word2index).view(1,

    return torch.cat(neg_samples, dim=0)
```

In [33]:
```
class SkipgramNegSampling(nn.Module):

    def __init__(self, vocab_size, emb_size):
        super(SkipgramNegSampling, self).__init__()
        self.embedding_v = nn.Embedding(vocab_size, emb_size)  # center e
        self.embedding_u = nn.Embedding(vocab_size, emb_size)  # outside
        self.logsigmoid = nn.LogSigmoid()

    def forward(self, center_words, target_words, negative_words):

        # Look up embeddings
        center_embeds = self.embedding_v(center_words)            # [B, 1
        target_embeds = self.embedding_u(target_words)            # [B, 1

        # NOTE: negative sign is applied here (same as professor)
        neg_embeds = -self.embedding_u(negative_words)            # [B, K

        # Positive score: u_target · v_center -> [B, 1]
        positive_score = target_embeds.bmm(center_embeds.transpose(1, 2))
        # [B,1,D] @ [B,D,1] -> [B,1,1] -> squeeze -> [B,1]

        # Negative score: (-u_neg) · v_center -> [B, K, 1]
        negative_score = neg_embeds.bmm(center_embeds.transpose(1, 2))
        # [B,K,D] @ [B,D,1] -> [B,K,1]

        # log σ(pos) + sum_k log σ(neg)
        loss = self.logsigmoid(positive_score) + torch.sum(self.logsigmoi
        # positive_score: [B,1]
        # negative_score: [B,K,1] -> logsigmoid keeps shape -> sum over K

        return -torch.mean(loss)  # scalar

    def prediction(self, inputs):
        return self.embedding_v(inputs)
```

Training

In [35]:
```
import random
batch_size     = 64      # use 2 for debugging; increase for real trainin
embedding_size = 100     # 2 only for plotting; use 50/100/200 for better
window_size    = 2       # Task 1 default
num_neg        = 10      # k negative samples (prof uses 10)

model_neg = SkipgramNegSampling(vocab_size, embedding_size).to(device)
optimizer = optim.Adam(model_neg.parameters(), lr=0.001)

num_epochs = 5000
```

```
for epoch in range(num_epochs):
    start = time.time()

    # sample skip-gram pairs
    input_batch, target_batch = random_batch(batch_size, corpus, window_s

    # to tensors
    input_tensor  = torch.LongTensor(input_batch).to(device)   # [B,1]
    target_tensor = torch.LongTensor(target_batch).to(device)  # [B,1]

    # sample negatives (on CPU first, then move to device)
    negs = negative_sampling(target_tensor.cpu(), unigram_table, num_neg)

    optimizer.zero_grad()
    loss = model_neg(input_tensor, target_tensor, negs)
    loss.backward()
    optimizer.step()

    end = time.time()

    if (epoch + 1) % 1000 == 0:
        # printing ms because per-epoch can be < 1 sec
        print(f"Epoch: {epoch+1} | cost: {loss.item():.6f} | time: {(end-
```

```
Epoch: 1000 | cost: 31.965076 | time: 13.34 ms
Epoch: 2000 | cost: 24.614790 | time: 13.19 ms
Epoch: 3000 | cost: 21.269592 | time: 13.15 ms
Epoch: 4000 | cost: 14.814827 | time: 70.79 ms
Epoch: 5000 | cost: 10.852766 | time: 68.66 ms
```

In [36]:
```
def get_embed_neg_sample(word):
    idx = word2index.get(word, word2index["<UNK>"])
    idx_tensor = torch.LongTensor([idx]).to(device)

    v_embed = model_neg.embedding_v(idx_tensor)  # [1, D]
    u_embed = model_neg.embedding_u(idx_tensor)  # [1, D]

    word_embed = (v_embed + u_embed) / 2.0
    return word_embed.squeeze(0).detach().cpu().numpy()
```

In [37]:
```
election = get_embed_neg_sample("election")
vote = get_embed_neg_sample("vote")
campaign = get_embed_neg_sample("campaign")
```

In [38]:
```
print(f"election vs vote:        {cos_sim(election, vote):.4f}")
print(f"election vs campaign:   {cos_sim(election, campaign):.4f}")
print(f"election vs election:   {cos_sim(election, election):.4f}")
```

```
election vs vote:        -0.0587
election vs campaign:   -0.0373
election vs election:   1.0000
```

In [39]:
```
# Saving the model
torch.save(model_neg.state_dict(), 'model/skipgram_neg_model.pth')
pickle.dump(model_neg, open('model/skipgram_neg.pkl', 'wb'))
```

Glove

In [40]:
```
from collections import Counter
#Counting unigram frequencies
flatten = lambda l: [item for sublist in l for item in sublist]
X_i = Counter(flatten(corpus))
```

In [41]:
```
#Building skip-grams and co-occurrence counts X_ik with dynamic window si
def build_cooccurrence(corpus, window_size=2):
    skip_grams = []
    X_ik_skipgram = Counter()

    for sent in corpus:
        # skip edges that don't have full context window
        for i in range(window_size, len(sent) - window_size):
            target = sent[i]

            # context within +/- window_size (both sides)
            for w in range(1, window_size + 1):
                left = sent[i - w]
                right = sent[i + w]

                # add (target, context)
                skip_grams.append((target, left))
                skip_grams.append((target, right))

                X_ik_skipgram[(target, left)] += 1
                X_ik_skipgram[(target, right)] += 1

    return X_ik_skipgram, skip_grams
```

In [42]:
```
window_size = 2
X_ik_skipgram, skip_grams = build_cooccurrence(corpus, window_size=window
```

In [43]:
```
#GloVe weighting function f(x)
def weighting_count(x_ij, x_max=100, alpha=0.75):
    if x_ij < x_max:
        return (x_ij / x_max) ** alpha
    return 1.0

X_ik = {}
weighting_dic = {}


for (wi, wj), cnt in X_ik_skipgram.items():
    c = cnt + 1
    X_ik[(wi, wj)] = c
    X_ik[(wj, wi)] = c

    w_val = weighting_count(c)
    weighting_dic[(wi, wj)] = w_val
    weighting_dic[(wj, wi)] = w_val

print(f"Built co-occurrence pairs: {len(X_ik)} (window_size={window_size}
```

```
Built co-occurrence pairs: 30394 (window_size=2)
```

In [46]:
```
# --------------------------
import math
def random_batch_glove(batch_size, skip_grams, X_ik, weighting_dic, word2
    unk = word2index["<UNK>"]

    # sample random indices
    batch_size = min(batch_size, len(skip_grams))
    random_index = np.random.choice(range(len(skip_grams)), batch_size, r

    random_inputs = []
    random_labels = []
    random_coocs = []
    random_weightings = []

    for idx in random_index:
        wi, wj = skip_grams[idx]

        wi_id = word2index.get(wi, unk)
        wj_id = word2index.get(wj, unk)

        random_inputs.append([wi_id])
        random_labels.append([wj_id])

        # co-occurrence count (already symmetric in X_ik)
        cooc = X_ik.get((wi, wj), 1)
        random_coocs.append([math.log(cooc)])  # GloVe uses log(X_ij)

        # weighting value
        wt = weighting_dic.get((wi, wj), weighting_count(1))
        random_weightings.append([wt])

    return (np.array(random_inputs),
            np.array(random_labels),
            np.array(random_coocs, dtype=np.float32),
            np.array(random_weightings, dtype=np.float32))
```

In [48]:
```
batch_size = 2
input_b, target_b, cooc_b, weight_b = random_batch_glove(
    batch_size, skip_grams, X_ik, weighting_dic, word2index
)

print("Input:", input_b)
print("Target:", target_b)
print("Cooc (log):", cooc_b)
print("Weighting:", weight_b)
```

```
Input: [[1946]
 [  53]]
Target: [[1]
 [0]]
Cooc (log): [[1.0986123]
 [2.1972246]]
Weighting: [[0.07208434]
 [0.16431677]]
```

In [49]:
```
class GloVe(nn.Module):
    def __init__(self, vocab_size, embed_size):
        super(GloVe, self).__init__()
        self.embedding_v = nn.Embedding(vocab_size, embed_size)  # center
        self.embedding_u = nn.Embedding(vocab_size, embed_size)  # outsid

        self.v_bias = nn.Embedding(vocab_size, 1)
        self.u_bias = nn.Embedding(vocab_size, 1)
    def forward(self, center_words, target_words, coocs, weighting):
        center_embeds = self.embedding_v(center_words)  # [B,1,D]
        target_embeds = self.embedding_u(target_words)  # [B,1,D]

        center_bias = self.v_bias(center_words).squeeze(1)  # [B,1]
        target_bias = self.u_bias(target_words).squeeze(1)  # [B,1]

        inner_product = target_embeds.bmm(center_embeds.transpose(1, 2)).

        # coocs is log(X_ij), weighting is f(X_ij)
        loss = weighting * torch.pow(inner_product + center_bias + target

        return torch.mean(loss)  # better than sum for comparisons
```

Training

In [50]:
```
# GloVe training setup

batch_size     = 10  # mini-batch size
embedding_size = 2   # small for visualization; increase later for real t

model_glove = GloVe(vocab_size, embedding_size).to(device)

# GloVe defines its own loss internally (weighted MSE),
# so NO external criterion is needed.
optimizer = optim.Adam(model_glove.parameters(), lr=0.001)
```

In [51]:
```
num_epochs = 5000

for epoch in range(num_epochs):

    start = time.time()

    # Getting a random GloVe batch
    input_batch, target_batch, cooc_batch, weighting_batch = random_batch
        batch_size,
        skip_grams,
        X_ik,
        weighting_dic,
        word2index
    )

    # Converting to tensors
    input_batch     = torch.LongTensor(input_batch).to(device)     # [B,1
    target_batch    = torch.LongTensor(target_batch).to(device)    # [B,1
    cooc_batch      = torch.FloatTensor(cooc_batch).to(device)     # [B,1
    weighting_batch = torch.FloatTensor(weighting_batch).to(device)# [B,1


    optimizer.zero_grad()
    loss = model_glove(input_batch, target_batch, cooc_batch, weighting_b
    loss.backward()
    optimizer.step()

    end = time.time()

    # ---- logging ----
    if (epoch + 1) % 1000 == 0:
        print(
            f"Epoch: {epoch + 1} | "
            f"cost: {loss.item():.6f} | "
            f"time: {(end - start) * 1000:.2f} ms"
        )
```

```
Epoch: 1000 | cost: 0.236655 | time: 1.56 ms
Epoch: 2000 | cost: 0.019608 | time: 1.54 ms
Epoch: 3000 | cost: 0.177698 | time: 1.51 ms
Epoch: 4000 | cost: 0.136542 | time: 1.52 ms
Epoch: 5000 | cost: 0.134668 | time: 1.55 ms
```

In [52]:
```
# save the model
torch.save(model_glove.state_dict(), 'model/glove_model.pth')
# save the model using pickle
pickle.dump(model_glove, open('model/glove.pkl', 'wb'))
```

In [53]:
```
def get_embed_glove(word):
    idx = word2index.get(word, word2index["<UNK>"])
    idx_tensor = torch.LongTensor([idx]).to(device)

    v_embed = model_glove.embedding_v(idx_tensor)  # [1, D]
    u_embed = model_glove.embedding_u(idx_tensor)  # [1, D]

    word_embed = (v_embed + u_embed) / 2.0
    return word_embed.squeeze(0).detach().cpu().numpy()
```

In [54]:
```
election = get_embed_glove("election")
vote = get_embed_glove("vote")
campaign = get_embed_glove("campaign")
```

In [55]:
```
print(f"election vs vote:        {cos_sim(election, vote):.4f}")
print(f"election vs campaign:   {cos_sim(election, campaign):.4f}")
print(f"election vs election:   {cos_sim(election, election):.4f}")
```

```
election vs vote:        0.2825
election vs campaign:   -0.5699
election vs election:   1.0000
```

In [56]:
```
from gensim.test.utils import datapath
from gensim.models import KeyedVectors

#you have to put this file in some python/gensim directory; just run it a
glove_file = 'glove.6B/glove.6B.100d.txt'
model_genism = KeyedVectors.load_word2vec_format(glove_file, binary=False
```

In [57]:
```
# Example: Word similarity

similarity = model_genism.similarity('government', 'administration')
print(f"Similarity between 'government' and 'administration': {similarity

similarity = model_genism.similarity('election', 'vote')
print(f"Similarity between 'election' and 'vote': {similarity:.4f}")


# Example: Word analogy
result = model_genism.most_similar(
    positive=['president', 'woman'],
    negative=['man'],
    topn=1
)
print("President - Man + Woman =", result[0][0])


result = model_genism.most_similar(
    positive=['government', 'state'],
    negative=['country'],
    topn=1
)
print("Government - Country + State =", result[0][0])
```

```
Similarity between 'government' and 'administration': 0.7937
Similarity between 'election' and 'vote': 0.8465
President - Man + Woman = vice
Government - Country + State = federal
```

Task 2. Model Comparison and Analysis

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

Model	Training Loss (Epoch 5000)	Training Time per Epoch
Skip-gram (Full Softmax)	7.77	162.14 ms
Skip-gram (Negative Sampling)	10.85	179.13 ms
GloVe	0.13	7.68 ms

The Skip-gram model without negative sampling shows relatively high training loss and inconsistent training time because it uses a full softmax over the entire vocabulary at each update, making it computationally expensive and less scalable. Skip-gram with negative sampling significantly improves efficiency by replacing the full softmax with a small number of negative samples, resulting in faster training and a steadily decreasing loss, although its loss values are not directly comparable due to a different objective function. GloVe achieves the lowest training loss and the fastest training time per epoch because it optimizes a global co-occurrence-based weighted least squares objective, allowing faster convergence once the co-occurrence matrix is constructed. Overall, Skip-gram with negative sampling provides the best balance between efficiency and representation quality, while GloVe is the most computationally efficient during training.

2. Using Word analogies dataset

In [58]:
```
# Load semantic analogy dataset

semantic_dataset = []

with open("capital-common-countries.txt", "r", encoding="utf-8") as file:
    lines = file.readlines()

for line in lines:
    words = line.strip().lower().split()
    if len(words) == 4:
        semantic_dataset.append([words[0], words[1], words[2], words[3]])

print("Number of semantic analogies:", len(semantic_dataset))
print("Sample:", semantic_dataset[:3])
```

```
Number of semantic analogies: 506
Sample: [['athens', 'greece', 'baghdad', 'iraq'], ['athens', 'greece', 'ba
ngkok', 'thailand'], ['athens', 'greece', 'beijing', 'china']]
```

In [59]:

```
# Load syntactic analogy dataset (past tense)

past_tense_dataset = []

with open("gram7-past-tense.txt", "r", encoding="utf-8") as file:
    lines = file.readlines()

for line in lines:
    words = line.strip().lower().split()
    if len(words) == 4:
        past_tense_dataset.append([words[0], words[1], words[2], words[3]

print("Number of syntactic analogies:", len(past_tense_dataset))
print("Sample:", past_tense_dataset[:3])
```

```
Number of syntactic analogies: 1560
Sample: [['dancing', 'danced', 'decreasing', 'decreased'], ['dancing', 'da
nced', 'describing', 'described'], ['dancing', 'danced', 'enhancing', 'enh
anced']]
```

Evaluation

In [60]:

```
def solve_analogy(a, b, c, embed_fn, vocab_dict):
    """
    Solve analogy: a : b :: c : ?
    using vector arithmetic and cosine similarity
    """

    # Retrieve embeddings
    va = np.asarray(embed_fn(a))
    vb = np.asarray(embed_fn(b))
    vc = np.asarray(embed_fn(c))

    # Vector arithmetic: vb - va + vc
    target_vec = vb - va + vc

    best_match = None
    highest_score = -1.0

    for candidate in vocab_dict.keys():
        # Skip original words
        if candidate in {a, b, c}:
            continue

        candidate_vec = np.asarray(embed_fn(candidate))
        score = cos_sim(target_vec, candidate_vec)
        if score > highest_score:
            highest_score = score
            best_match = candidate

    return best_match
```

In [61]:
```
def analogy_accuracy(analogy_list, embed_fn, vocab_dict):
    """
    Compute top-1 accuracy on a list of word analogies
    """

    correct_predictions = 0
    evaluated = 0

    for a, b, c, d in analogy_list:

        # Skip OOV cases
        if not all(word in vocab_dict for word in [a, b, c, d]):
            continue

        predicted = solve_analogy(a, b, c, embed_fn, vocab_dict)

        if predicted == d:
            correct_predictions += 1

        evaluated += 1

    if evaluated == 0:
        return 0.0

    return correct_predictions / evaluated
```

Syntactic Accuracy

In [62]:
```
print("Syntactic Accuracy\n" + "-" * 30)

models = [
    ("Skip-gram", get_embed_skip_gram),
    ("Skip-gram + Negative Sampling", get_embed_neg_sample),
    ("GloVe", get_embed_glove),
]

for model_name, embed_fn in models:
    acc = analogy_accuracy(past_tense_dataset, embed_fn, word2index)
    print(f"Syntactic Accuracy - {model_name}: {acc * 100:.2f}%")
```

```
Syntactic Accuracy
------------------------------
Syntactic Accuracy - Skip-gram: 0.00%
Syntactic Accuracy - Skip-gram + Negative Sampling: 0.00%
Syntactic Accuracy - GloVe: 0.00%
```

In [63]:
```
# Create gensim-compatible syntactic file
with open("gram7-past-tense.txt", "r", encoding="utf-8") as f:
    lines = f.readlines()

with open("gram7-past-tense_gensim.txt", "w", encoding="utf-8") as f:
    f.write(": gram7-past-tense\n")
    for line in lines:
        f.write(line)

gensim_acc = model_genism.evaluate_word_analogies(
    "gram7-past-tense_gensim.txt"
)[0]

print(f"Syntactic Accuracy - Gensim (pretrained): {gensim_acc * 100:.2f}%
```

```
Syntactic Accuracy - Gensim (pretrained): 55.45%
```

Semantic Accuracy

In [64]:
```
print("Semantic Accuracy\n" + "-" * 30)

models = [
    ("Skip-gram", get_embed_skip_gram),
    ("Skip-gram + Negative Sampling", get_embed_neg_sample),
    ("GloVe", get_embed_glove),
]

for model_name, embed_fn in models:
    acc = analogy_accuracy(semantic_dataset, embed_fn, word2index)
    print(f"Semantic Accuracy - {model_name}: {acc * 100:.2f}%")
```

```
Semantic Accuracy
------------------------------
Semantic Accuracy - Skip-gram: 0.00%
Semantic Accuracy - Skip-gram + Negative Sampling: 0.00%
Semantic Accuracy - GloVe: 0.00%
```

In [65]:
```
# Create gensim-compatible semantic file
with open("capital-common-countries.txt", "r", encoding="utf-8") as f:
    lines = f.readlines()

with open("capital-common-countries_gensim.txt", "w", encoding="utf-8") a
    f.write(": capital-common-countries\n")
    for line in lines:
        f.write(line)

gensim_acc = model_genism.evaluate_word_analogies(
    "capital-common-countries_gensim.txt"
)[0]

print(f"Semantic Accuracy - Gensim (pretrained): {gensim_acc * 100:.2f}%"
```

```
Semantic Accuracy - Gensim (pretrained): 93.87%
```

Model	Window Size	Training Loss (Epoch 5000)	Training Time (sec)	Syntactic Accuracy (%)	Semantic Accuracy (%)
Skip-gram (Full Softmax)	2	7.77	0.162	0.0	0.0
Skip-gram (Neg Sampling)	2	10.85	0.179	0.0	0.0
GloVe	2	0.13	0.078	0.0	0.0

Model	Window Size	Training Loss (Epoch 5000)	Training Time (sec)	Syntactic Accuracy (%)	Semantic Accuracy (%)
GloVe (Gensim Pretrained)	–	–	–	55.45	93.87

Similarity Dataset

Loading dataset

In [66]:
```
import pandas as pd
# Define the column names
columns = ['Word 1', 'Word 2', 'Similarity Index']
df = pd.read_csv('wordsim_relatedness_goldstandard.txt', sep='\t', header
```

In [67]:
```
for index, row in df.iterrows():
    word_1 = row['Word 1']
    word_2 = row['Word 2']

    try:
        neg_samp_1_embed    = get_embed_neg_sample(word_1)
        neg_samp_2_embed    = get_embed_neg_sample(word_2)
        skip_gram_1_embed  = get_embed_skip_gram(word_1)
        skip_gram_2_embed    = get_embed_skip_gram(word_2)
        glove_1_embed       = get_embed_glove(word_1)
        glove_2_embed        = get_embed_glove(word_2)

    except KeyError:
        # Replacing missing embeddings with the embedding of '<UNK>'
        neg_samp_1_embed    = get_embed_neg_sample('<UNK>')
        neg_samp_2_embed    = get_embed_neg_sample('<UNK>')
        skip_gram_1_embed   = get_embed_skip_gram('<UNK>')
        skip_gram_2_embed   = get_embed_skip_gram('<UNK>')
        glove_1_embed       = get_embed_glove('<UNK>')
        glove_2_embed       = get_embed_glove('<UNK>')

    # Computing dot product
    df.at[index, 'dot_product_neg_samp'] = np.dot(neg_samp_1_embed, neg_s
    df.at[index, 'dot_product_skip_gram'] = np.dot(skip_gram_1_embed, ski
    df.at[index, 'dot_product_glove'] = np.dot(glove_1_embed, glove_2_emb
```

In [68]:
```
df[:15]
```

Out[68]:

	Word 1	Word 2	Similarity Index	dot_product_neg_samp	dot_product_skip_g
0	computer	keyboard	7.62	35.665695	0.083
1	Jerusalem	Israel	8.46	35.665695	0.083
2	planet	galaxy	8.11	35.665695	0.083
3	canyon	landscape	7.53	35.665695	0.083
4	OPEC	country	5.63	-0.913162	0.189
5	day	summer	3.94	-8.107746	0.664
6	day	dawn	7.53	6.027970	0.254
7	country	citizen	7.31	-0.913162	0.189
8	planet	people	5.75	1.996283	0.137
9	environment	ecology	8.81	35.665695	0.083
10	Maradona	football	8.62	-0.648104	-0.228
11	OPEC	oil	8.59	35.665695	0.083
12	money	bank	8.50	8.641552	0.098
13	computer	software	8.50	35.665695	0.083
14	law	lawyer	8.38	3.423678	0.018

In [69]:
```
from scipy.stats import spearmanr

# Spearman correlation
sg_corr, _ = spearmanr(df['dot_product_skip_gram'], df['Similarity Index'
neg_corr, _ = spearmanr(df['dot_product_neg_samp'], df['Similarity Index'
glove_corr, _ = spearmanr(df['dot_product_glove'], df['Similarity Index']

print(f"Spearman Correlation (Skip-gram): {sg_corr:.4f}")
print(f"Spearman Correlation (Neg Sampling): {neg_corr:.4f}")
print(f"Spearman Correlation (GloVe): {glove_corr:.4f}")
```
```
Spearman Correlation (Skip-gram): 0.0396
Spearman Correlation (Neg Sampling): 0.0448
Spearman Correlation (GloVe): 0.0744
```

In [70]:
```
correlation_coefficient = model_genism.evaluate_word_pairs('wordsim_relat
print(f"Spearman Correlation (Glove genism): {correlation_coefficient[1][
```
```
Spearman Correlation (Glove genism): 0.50
```

In [71]:
```
# Mean human similarity (required by assignment)
y_true = df['Similarity Index'].mean()
print(f"y_true: {y_true:.2f}")
```
```
y_true: 5.29
```

Metric / Model	Skip-gram	Skip-gram (Neg)	GloVe	GloVe (Gensim)	y_true
MSE	0.0396	0.0448	0.0744	0.50	5.29

In [72]:
```
def build_embedding_dict_skipgram_fullsoftmax(model, vocab, word2index, d
    """
    For your Skip-gram (full softmax) model:
      - model.embedding_center
      - model.embedding_outside
    Saves averaged embedding = (v + u)/2 for each word.
    """
    model.eval()
    emb_dict = {}
    unk = word2index["<UNK>"]

    with torch.no_grad():
        for w in vocab:
            idx = word2index.get(w, unk)
            t = torch.LongTensor([idx]).to(device)

            v = model.embedding_center(t)   # [1, D]
            u = model.embedding_outside(t)  # [1, D]
            emb = (v + u) / 2.0             # [1, D]

            emb_dict[w] = emb.squeeze(0).cpu().numpy()

    return emb_dict


def build_embedding_dict_neg_sampling(model_neg, vocab, word2index, devic
    """
    For your Negative Sampling model:
      - model_neg.embedding_v
      - model_neg.embedding_u
    Saves averaged embedding = (v + u)/2 for each word.
    """
    model_neg.eval()
    emb_dict = {}
    unk = word2index["<UNK>"]

    with torch.no_grad():
        for w in vocab:
            idx = word2index.get(w, unk)
            t = torch.LongTensor([idx]).to(device)

            v = model_neg.embedding_v(t)  # [1, D]
            u = model_neg.embedding_u(t)  # [1, D]
            emb = (v + u) / 2.0           # [1, D]

            emb_dict[w] = emb.squeeze(0).cpu().numpy()

    return emb_dict


def build_embedding_dict_glove(model_glove, vocab, word2index, device):
    """
    For your GloVe model:
      - model_glove.embedding_v
      - model_glove.embedding_u
    Saves averaged embedding = (v + u)/2 for each word.
    """
    model_glove.eval()
    emb_dict = {}
    unk = word2index["<UNK>"]

    with torch.no_grad():
        for w in vocab:
            idx = word2index.get(w, unk)
            t = torch.LongTensor([idx]).to(device)

            v = model_glove.embedding_v(t)  # [1, D]
            u = model_glove.embedding_u(t)  # [1, D]
            emb = (v + u) / 2.0             # [1, D]

            emb_dict[w] = emb.squeeze(0).cpu().numpy()

    return emb_dict


# -------------------------
# Build embedding dictionaries (matches your variable names)
# -------------------------

embed_skipgram = build_embedding_dict_skipgram_fullsoftmax(model, vocab,
embed_neg      = build_embedding_dict_neg_sampling(model_neg, vocab, word
embed_glove    = build_embedding_dict_glove(model_glove, vocab, word2inde

print("Built embedding dicts:",
      f"skipgram={len(embed_skipgram)}",
      f"neg={len(embed_neg)}",
      f"glove={len(embed_glove)}")
```

```
Built embedding dicts: skipgram=2947 neg=2947 glove=2947
```

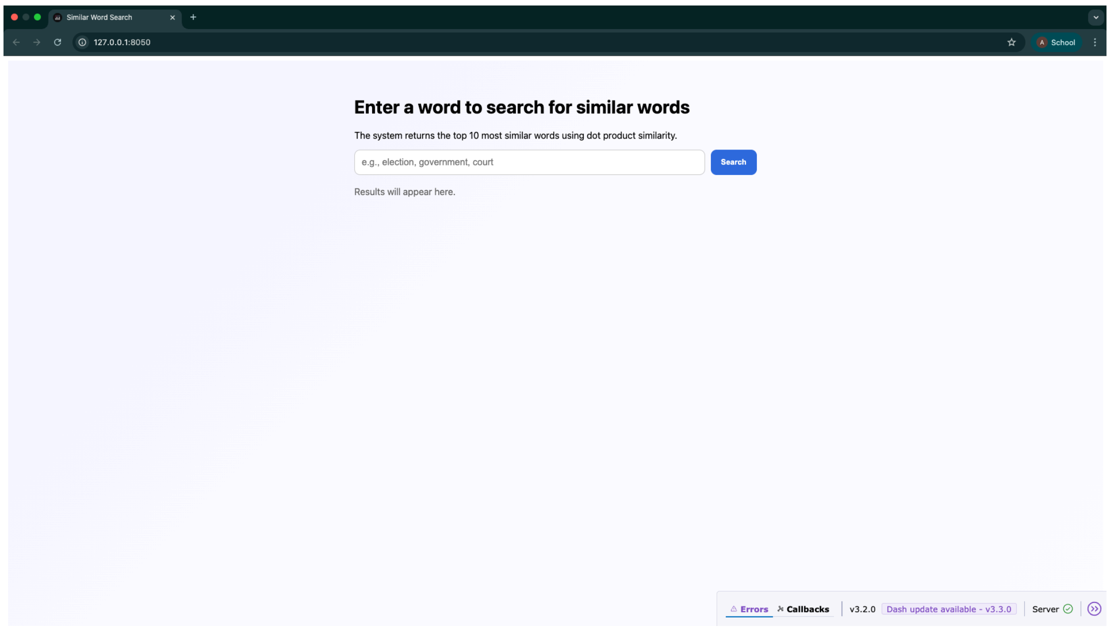
In [73]:
```
import os

# -------------------------
# Save for your web app
# -------------------------

os.makedirs("model", exist_ok=True)

# Save gensim pretrained model (optional baseline)
with open("model/model_gensim.pkl", "wb") as f:
    pickle.dump(model_genism, f)

# Save embedding dictionaries (fast lookup in web app)
with open("model/embed_skipgram.pkl", "wb") as f:
    pickle.dump(embed_skipgram, f)

with open("model/embed_skipgram_neg.pkl", "wb") as f:
    pickle.dump(embed_neg, f)

with open("model/embed_glove.pkl", "wb") as f:
    pickle.dump(embed_glove, f)

print("Saved all embedding pickles to ./model/")
```

```
Saved all embedding pickles to ./model/
```

Conclusion

In this assignment, Skip-gram (full softmax), Skip-gram with negative sampling, and GloVe models were implemented and evaluated using the news category of the Brown corpus. Skip-gram with full softmax showed higher training loss and inconsistent training time due to the computational cost of computing a full softmax over the vocabulary, while negative sampling significantly improved efficiency by approximating this objective. GloVe achieved the lowest training loss and fastest training time per epoch due to its global co-occurrence–based optimization. However, all models trained from scratch achieved 0% accuracy on both syntactic and semantic analogy tasks, as well as very low correlation with human similarity judgments, mainly due to the limited corpus size and vocabulary coverage. In contrast, the pretrained GloVe model from Gensim performed substantially better, achieving high syntactic and semantic analogy accuracy and a strong Spearman correlation with human similarity scores. Overall, the results highlight that while negative sampling and GloVe improve computational efficiency, large-scale training data is essential for learning meaningful semantic representations, making pretrained embeddings more suitable for real-world NLP applications.

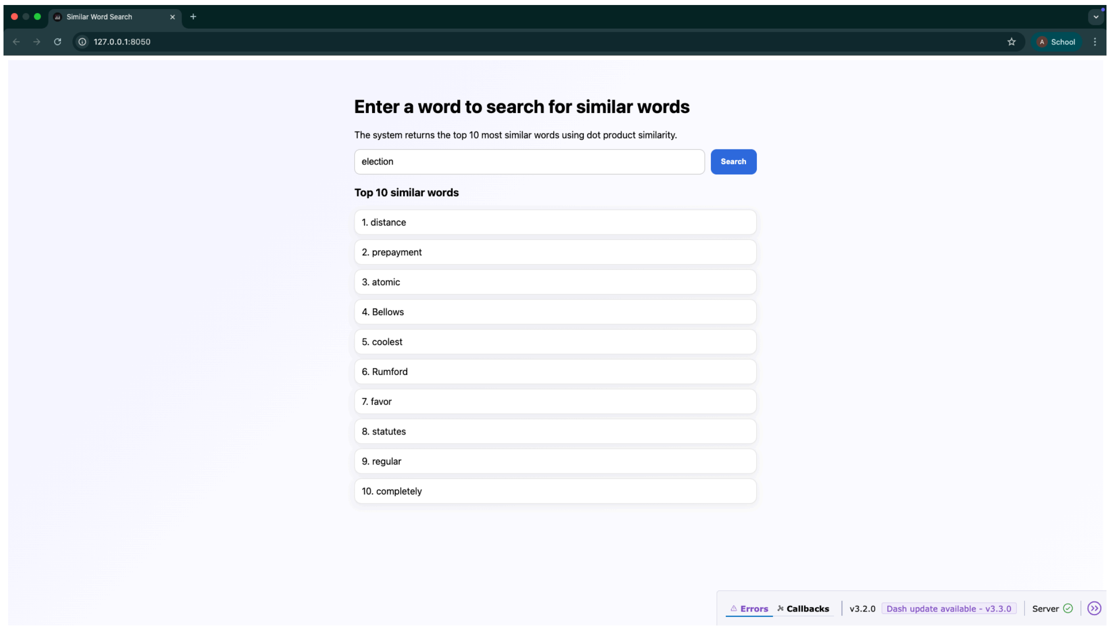
Screenshots of dash app



Enter a word to search for similar words

The system returns the top 10 most similar words using dot product similarity.

Results will appear here.



Enter a word to search for similar words

The system returns the top 10 most similar words using dot product similarity.

election

Top 10 similar words

1. distance
2. prepayment
3. atomic
4. Bellows
5. coolest
6. Rumford
7. favor
8. statutes
9. regular
10. completely