```
# Expt. 01: Design and implement a neural based network for generating word embedding for words in a document corpus

!pip install torch torchvision torchaudio --quiet
```

```
[notice] A new release of pip is available: 25.0.1 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
# Import Libraries
import re, random, math, collections
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

torch.manual_seed(0)
random.seed(0)
np.random.seed(0)

# demo corpus
corpus = """
Word embeddings are numerical vector representations of words that
capture their
semantic meaning and relationships in a continuous, low-dimensional
space. Unlike traditional
one-hot encoding, which treats words as independent symbols,
embeddings position similar words
closer together based on their contextual usage in large text corpora.
Techniques like Word2Vec,
GloVe, and FastText learn these representations by analyzing co-
occurrence patterns of words within
specific contexts. This allows models to generalize better and
understand linguistic relationships
such as synonyms, analogies, and semantic similarity. Word embeddings
form the foundation for many
natural language processing tasks, including text classification,
sentiment analysis, translation,
and information retrieval.
"""

# Tokenize & build vocabulary
def tokenize(text):
    text = text.lower()
    text = re.sub(r"[^a-z\s']", " ", text)
    tokens = [t for t in text.split() if t.strip()]
    return tokens

tokens = tokenize(corpus)
```

```
min_count = 1  # keep all words for this tiny demo; raise to 2+ for
larger text
freq = collections.Counter(tokens)
vocab = [w for w,c in freq.items() if c >= min_count]
word2id = {w:i for i,w in enumerate(vocab)}
id2word = {i:w for w,i in word2id.items()}

indexed = [word2id[w] for w in tokens if w in word2id]
vocab_size = len(vocab)
vocab_size, list(freq.items())[:10]
```

```
(85,
 [('word', 3),
  ('embeddings', 3),
  ('are', 1),
  ('numerical', 1),
  ('vector', 1),
  ('representations', 2),
  ('of', 2),
  ('words', 4),
  ('that', 1),
  ('capture', 1)])
```

```
# Create skip-gram training pairs (center -> context)
window_size = 2

pairs = []
for i, center in enumerate(indexed):
    left = max(0, i - window_size)
    right = min(len(indexed), i + window_size + 1)
    for j in range(left, right):
        if j == i:
            continue
        pairs.append((center, indexed[j]))

len(pairs), pairs[:10]
```

```
(410,
 [(0, 1),
  (0, 2),
  (1, 0),
  (1, 2),
  (1, 3),
  (2, 0),
  (2, 1),
  (2, 3),
  (2, 4),
  (3, 1)])
```

```
# Negative sampling distribution (unigram^0.75)
word_counts = np.zeros(vocab_size, dtype=np.float64)
for w in indexed: word_counts[w] += 1
unigram = word_counts ** 0.75
unigram = unigram / unigram.sum()

neg_table = np.arange(vocab_size)
# We'll sample negatives using np.random.choice with 'p=unigram'
(simple and fine for small demos)

# Dataset & DataLoader helpers
class SGNSSampler(torch.utils.data.Dataset):
    def __init__(self, pairs):
        self.pairs = pairs
    def __len__(self):
        return len(self.pairs)
    def __getitem__(self, idx):
        return self.pairs[idx]

batch_size = 256
dataset = SGNSSampler(pairs)
loader = torch.utils.data.DataLoader(dataset, batch_size=batch_size,
shuffle=True, drop_last=False)

# Model
class SGNS(nn.Module):
    def __init__(self, vocab_size, emb_dim):
        super().__init__()
        self.in_embed = nn.Embedding(vocab_size, emb_dim)
        self.out_embed = nn.Embedding(vocab_size, emb_dim)
        nn.init.uniform_(self.in_embed.weight, -0.5/emb_dim,
0.5/emb_dim)
        nn.init.zeros_(self.out_embed.weight)

    def forward(self, centers, pos_contexts, neg_contexts):
        # centers: [B], pos_contexts: [B], neg_contexts: [B, K]
        v = self.in_embed(centers)                  # [B, D]
        u_pos = self.out_embed(pos_contexts)        # [B, D]
        pos_score = (v * u_pos).sum(dim=1)          # [B]

        u_neg = self.out_embed(neg_contexts)        # [B, K, D]
        neg_score = torch.bmm(u_neg, v.unsqueeze(2)).squeeze(2)  # [B,
K]

        loss = -torch.log(torch.sigmoid(pos_score) + 1e-9).mean() \
               -torch.log(torch.sigmoid(-neg_score) + 1e-9).mean()
        return loss

    def get_embeddings(self):
        with torch.no_grad():
```

```
            return self.in_embed.weight.detach().cpu().numpy()

emb_dim = 50
model = SGNS(vocab_size, emb_dim)
optimizer = optim.Adam(model.parameters(), lr=0.01)

# Training loop with negative sampling
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)

negatives = 5
epochs = 8

for epoch in range(1, epochs+1):
    running = 0.0
    for centers, pos in loader:
        centers = centers.to(device)
        pos = pos.to(device)

        # sample negatives for each example in the batch
        neg = np.random.choice(vocab_size, size=(centers.size(0),
negatives), p=unigram)
        neg = torch.tensor(neg, dtype=torch.long, device=device)

        loss = model(centers, pos, neg)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        running += loss.item() * centers.size(0)

    print(f"Epoch {epoch}/{epochs} - loss:
{running/len(dataset):.4f}")
```

```
Epoch 1/8 - loss: 1.3863
Epoch 2/8 - loss: 1.3847
Epoch 3/8 - loss: 1.3787
Epoch 4/8 - loss: 1.3674
Epoch 5/8 - loss: 1.3502
Epoch 6/8 - loss: 1.3282
Epoch 7/8 - loss: 1.2977
Epoch 8/8 - loss: 1.2617
```

```
# find most similar words by cosine similarity
def get_word_vector(word):
    emb = model.get_embeddings()
    if word not in word2id:
        raise ValueError(f"'{word}' not in vocabulary")
    return emb[word2id[word]]

def most_similar(query, topn=5):
    emb = model.get_embeddings()
```

```
    if query not in word2id:
        return []
    qv = emb[word2id[query]]
    norms = np.linalg.norm(emb, axis=1) * np.linalg.norm(qv)
    sims = emb @ qv / (norms + 1e-9)
    best = np.argsort(-sims)[:topn+1]
    result = [(id2word[i], float(sims[i])) for i in best if id2word[i]
!= query][:topn]
    return result

for w in ["language", "embeddings", "model", "context", "neural"]:
    if w in word2id:
        print(w, "->", most_similar(w, topn=5))
```

```
language -> [('processing', 0.8033797144889832), ('for',
0.7841214537620544), ('natural', 0.7175846695899963), ('many',
0.6530065536499023), ('foundation', 0.6163556575775146)]
embeddings -> [('position', 0.807449996471405), ('symbols',
0.6795068383216858), ('form', 0.6321839690208435), ('are',
0.6256264448165894), ('similar', 0.597645103931427)]
```

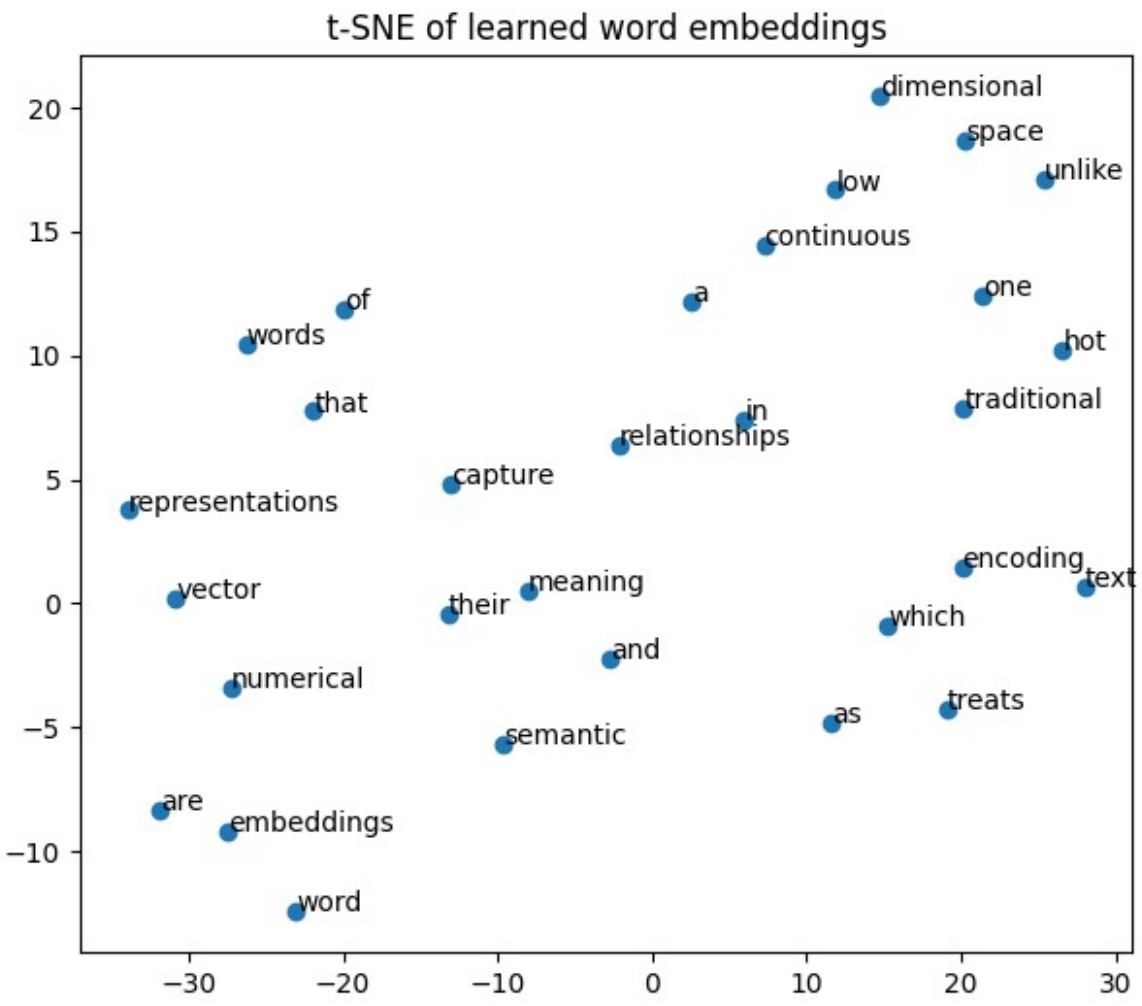
```
# Visualize a small subset with t-SNE
subset_words = [w for w,_ in freq.most_common(30)]
subset_ids = [word2id[w] for w in subset_words if w in word2id]
emb = model.get_embeddings()[subset_ids]

tsne = TSNE(n_components=2, init="random", learning_rate="auto",
perplexity=min(10, len(subset_ids)-1), random_state=0)
xy = tsne.fit_transform(emb)

plt.figure(figsize=(7,6))
plt.scatter(xy[:,0], xy[:,1])
for i,w in enumerate([id2word[i] for i in subset_ids]):
    plt.annotate(w, (xy[i,0], xy[i,1]))
plt.title("t-SNE of learned word embeddings")
plt.show()
```

```
C:\Users\ppjai\AppData\Local\Programs\Python\Python313\Lib\site-
packages\joblib\externals\loky\backend\context.py:110: UserWarning:
Could not find the number of physical cores for the following reason:
[WinError 2] The system cannot find the file specified
Returning the number of logical cores instead. You can silence this
warning by setting LOKY_MAX_CPU_COUNT to the number of cores you want
to use.
  warnings.warn(
  File "C:\Users\ppjai\AppData\Local\Programs\Python\Python313\Lib\
site-packages\joblib\externals\loky\backend\context.py", line 199, in
_count_physical_cores
    cpu_info = subprocess.run(
        "wmic CPU Get NumberOfCores /Format:csv".split(),
```

```
        capture_output=True, text=True)
  File "C:\Users\ppjai\AppData\Local\Programs\Python\Python313\Lib\
subprocess.py", line 556, in run
    with Popen(*popenargs, **kwargs) as process:
         ~~~~~^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\ppjai\AppData\Local\Programs\Python\Python313\Lib\
subprocess.py", line 1038, in __init__
    self._execute_child(args, executable, preexec_fn, close_fds,
    ~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
                        pass_fds, cwd, env,
                        ^^^^^^^^^^^^^^^^^^^
    ...<5 lines>...
                        gid, gids, uid, umask,
                        ^^^^^^^^^^^^^^^^^^^^^^
                        start_new_session, process_group)
                        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\ppjai\AppData\Local\Programs\Python\Python313\Lib\
subprocess.py", line 1550, in _execute_child
    hp, ht, pid, tid = _winapi.CreateProcess(executable, args,
                       ~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^
                             # no special security
                             ^^^^^^^^^^^^^^^^^^^^^
    ...<4 lines>...
                             cwd,
                             ^^^^
                             startupinfo)
                             ^^^^^^^^^^^^
```


t-SNE of learned word embeddings
dimensional
space
unlike
low
continuous
a
one
of
words
hot
that
in
traditional
relationships
capture
representations
encoding
text
vector
meaning
their
which
and
numerical
treats
as
semantic
are
embeddings
word
20
15
10
5
0
−5
−10
−30
−20
−10
0
10
20
30