AT82.05 Artificial Intelligence: Natural Language Understanding (NLU)

A1: That's What I LIKE!

Name: Arya Shah

StudentID: st125462

In this assignment I will focus on creating a system to find similar context in natural language processing. The system, deployed on a website, should return the top paragraphs with the most similar context to a given query, such as "Harry Potter." This task will involve building upon existing code, understanding and implementing word embedding techniques, and creating a web interface for the system to deliver the results.

Task 1: Preparation and Training

Build upon the code discussed in class. Do not use pre-built solutions from the internet.

1. Read and understand the Word2Vec1 and GloVe2 papers.✅
2. Modify the Word2Vec (with & without negative sampling) and GloVe from the lab lecture (3 points)

- Train using a real-world corpus (suggest to categories news from nltk datset). Ensure to source this dataset from reputable public databases or repositories. It is imperative to give proper credit to the dataset source in your documentation.✅
- Create a function that allows dynamic modification of the window size during training. Use a window size of 2 as default.✅

I make use of the NLTK Brown corpus (as mentioned in the assignment problem statement). Source: https://www.nltk.org/nltk_data/

Utility Functions

The below code helps in facilitating training, logging of results and testing the trainied models by various methods

In [2]:
```
import numpy as np
import torch
from collections import Counter
import nltk
from nltk.corpus import brown
from scipy.stats import spearmanr
from sklearn.metrics import mean_squared_error
```

```python
import time
import logging
import os
import requests
from torch import nn
import torch.nn.functional as F

# Setup logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

def load_news_corpus():
    """Load and preprocess the Brown corpus news category"""
    try:
        nltk.data.find('corpora/brown')
    except LookupError:
        nltk.download('brown')

    # Get news category sentences
    news_sents = brown.sents(categories='news')

    # Lowercase and join sentences
    corpus = [" ".join(sent).lower() for sent in news_sents]
    return corpus

def prepare_vocab(corpus, min_count=5):
    """Create vocabulary from corpus with minimum frequency threshold"""
    # Tokenize
    tokenized = [sent.split() for sent in corpus]
    # Count words
    word_counts = Counter([word for sent in tokenized for word in sent])
    # Filter by minimum count
    vocab = [word for word, count in word_counts.items() if count >= min_count]
    vocab.append('<UNK>')

    # Create mappings
    word2idx = {word: idx for idx, word in enumerate(vocab)}
    idx2word = {idx: word for word, idx in word2idx.items()}

    return tokenized, vocab, word2idx, idx2word

def load_word_analogies():
    """Load semantic and syntactic test sets"""
    semantic_file = "evaluation/capital-common-countries.txt"
    syntactic_file = "evaluation/past-tense.txt"

    semantic_pairs = []
    syntactic_pairs = []

    # Create evaluation directory if it doesn't exist
    os.makedirs("evaluation", exist_ok=True)

    # Create sample semantic analogies (capital-country)
    semantic_analogies = [
        "athens greece berlin germany",
        "athens greece moscow russia",
        "athens greece paris france",
        "berlin germany london england",
        "berlin germany madrid spain",
        "berlin germany paris france",
```

```
        "london england paris france",
        "london england rome italy",
        "madrid spain paris france",
        "madrid spain rome italy",
        "paris france rome italy",
        "rome italy tokyo japan"
    ]

    # Create sample syntactic analogies (verb past tense)
    syntactic_analogies = [
        "dance danced smile smiled",
        "dance danced walk walked",
        "decrease decreased increase increased",
        "describe described destroy destroyed",
        "eat ate speak spoke",
        "fall fell rise rose",
        "feed fed speak spoke",
        "find found lose lost",
        "go went speak spoke",
        "grow grew shrink shrank",
        "lose lost win won",
        "say said speak spoke",
        "sing sang write wrote",
        "sit sat speak spoke",
        "take took give gave"
    ]

    # Write sample files
    with open(semantic_file, 'w') as f:
        f.write('\n'.join(semantic_analogies))

    with open(syntactic_file, 'w') as f:
        f.write('\n'.join(syntactic_analogies))

    # Load and parse files
    def load_analogies(filename):
        pairs = []
        with open(filename, 'r') as f:
            for line in f:
                w1, w2, w3, w4 = line.strip().lower().split()
                pairs.append((w1, w2, w3, w4))
        return pairs

    semantic_pairs = load_analogies(semantic_file)
    syntactic_pairs = load_analogies(syntactic_file)

    return semantic_pairs, syntactic_pairs

def evaluate_analogies(model, word2idx, idx2word, pairs):
    """Evaluate word analogy accuracy"""
    correct = 0
    total = 0

    for w1, w2, w3, w4 in pairs:
        if w1 not in word2idx or w2 not in word2idx or w3 not in word2idx or w4
            continue

        # Get embeddings
        v1 = model.embedding_center(torch.LongTensor([word2idx[w1]])).detach()
        v2 = model.embedding_center(torch.LongTensor([word2idx[w2]])).detach()
```

```
        v3 = model.embedding_center(torch.LongTensor([word2idx[w3]])).detach()

        # v2 - v1 + v3 should be close to v4
        predicted = v2 - v1 + v3

        # Find closest word
        distances = []
        for idx in range(len(word2idx)):
            vec = model.embedding_center(torch.LongTensor([idx])).detach()
            dist = torch.nn.functional.cosine_similarity(predicted, vec)
            distances.append((dist.item(), idx))

        # Sort by similarity
        distances.sort(reverse=True)

        # Get top prediction
        pred_word = idx2word[distances[0][1]]

        if pred_word == w4:
            correct += 1
        total += 1

    return correct / total if total > 0 else 0

def load_similarity_dataset():
    """Load the WordSim-353 dataset for word similarity evaluation"""
    wordsim_path = "evaluation/wordsim353.txt"

    if not os.path.exists(wordsim_path):
        logger.error(f"WordSim-353 file not found at {wordsim_path}")
        return create_fallback_dataset()

    # Load and parse the dataset
    similarities = []
    try:
        with open(wordsim_path, 'r', encoding='utf-8') as f:
            # Read all lines
            lines = f.readlines()

            # Check if there's a header and skip if present
            start_idx = 0
            if lines and any(header in lines[0].lower() for header in ['word1',
                start_idx = 1

            # Parse each line
            for line in lines[start_idx:]:
                try:
                    # Handle both tab and space-separated formats
                    parts = line.strip().split()
                    if len(parts) >= 3:
                        word1, word2, score = parts[0], parts[1], float(parts[-1
                        similarities.append((word1.lower(), word2.lower(), float
                except (ValueError, IndexError) as e:
                    logger.warning(f"Skipping malformed line in similarity datas
                    continue

        if similarities:
            logger.info(f"Successfully loaded {len(similarities)} word pairs fro
            return similarities
        else:
```

```python
            logger.error("No valid similarities found in WordSim-353 file")
            return create_fallback_dataset()

    except Exception as e:
        logger.error(f"Error loading WordSim-353: {e}")
        return create_fallback_dataset()

def create_fallback_dataset():
    """Create a minimal fallback dataset for when WordSim-353 is unavailable"""
    logger.warning("Using fallback similarity dataset")
    return [
        ("car", "automobile", 1.0),
        ("gem", "jewel", 0.96),
        ("journey", "voyage", 0.89),
        ("boy", "lad", 0.83),
        ("coast", "shore", 0.79),
        ("asylum", "madhouse", 0.77),
        ("magician", "wizard", 0.73),
        ("midday", "noon", 0.71),
        ("furnace", "stove", 0.69),
        ("food", "fruit", 0.65),
    ]

def evaluate_similarity(model, word2idx, similarities):
    """Evaluate model performance on word similarity task"""
    model_sims = []
    human_sims = []
    num_pairs = 0

    for w1, w2, score in similarities:
        if w1 not in word2idx or w2 not in word2idx:
            continue

        # Get word vectors
        v1 = model.embedding_center(torch.tensor([word2idx[w1]]))
        v2 = model.embedding_center(torch.tensor([word2idx[w2]]))

        # Calculate cosine similarity
        cos_sim = F.cosine_similarity(v1, v2).item()

        model_sims.append(cos_sim)
        human_sims.append(score)
        num_pairs += 1

    if len(model_sims) > 1:
        # Calculate correlation and MSE
        correlation = spearmanr(model_sims, human_sims)[0]  # Take only the corr
        mse = mean_squared_error(human_sims, model_sims)
        return correlation, mse, num_pairs
    return 0.0, 0.0, 0

class ModelEvaluator:
    """Class to evaluate and compare different word embedding models"""

    def __init__(self):
        self.results = {}
        self.similarities = load_similarity_dataset()
        self.semantic_pairs, self.syntactic_pairs = load_word_analogies()

    def evaluate_model(self, model, word2idx, idx2word, model_name, window_size=
```

```
        """Evaluate a single model and store its results"""
        # Evaluate similarities
        correlation, mse, num_pairs = evaluate_similarity(model, word2idx, self.

        # Evaluate analogies
        semantic_acc = evaluate_analogies(model, word2idx, idx2word, self.semant
        syntactic_acc = evaluate_analogies(model, word2idx, idx2word, self.synta

        self.results[model_name] = {
            'window_size': window_size,
            'training_time': training_time,
            'final_loss': final_loss,
            'correlation': correlation,
            'mse': mse,
            'num_pairs': num_pairs,
            'semantic_acc': semantic_acc,
            'syntactic_acc': syntactic_acc
        }

    def print_training_table(self):
        """Print a table comparing training metrics and accuracy"""
        # Headers
        headers = ['Model', 'Window Size', 'Training Loss', 'Training Time', 'Sy
        col_widths = [max(len(str(h)), 15) for h in headers]

        # Update column widths based on data
        for model_name, metrics in self.results.items():
            col_widths[0] = max(col_widths[0], len(model_name))
            values = [
                metrics.get('window_size', 'N/A'),
                metrics.get('final_loss', 'N/A'),
                metrics.get('training_time', 'N/A'),
                metrics['syntactic_acc'],
                metrics['semantic_acc']
            ]
            for i, value in enumerate(values):
                col_widths[i+1] = max(col_widths[i+1], len(f'{value:.4f}' if isi

        # Print header
        header_line = ' | '.join(h.ljust(w) for h, w in zip(headers, col_widths)
        separator = '-' * len(header_line)
        print('\nTraining and Accuracy Results:')
        print(separator)
        print(header_line)
        print(separator)

        # Print each model's results
        for model_name, metrics in self.results.items():
            row = [
                model_name.ljust(col_widths[0]),
                str(metrics.get('window_size', 'N/A')).ljust(col_widths[1]),
                f"{metrics.get('final_loss', 'N/A'):.4f}".ljust(col_widths[2]) i
                f"{metrics.get('training_time', 'N/A'):.2f}s".ljust(col_widths[3
                f"{metrics['syntactic_acc']:.4f}".ljust(col_widths[4]),
                f"{metrics['semantic_acc']:.4f}".ljust(col_widths[5])
            ]
            print(' | '.join(row))
        print(separator)

    def print_similarity_table(self):
```

```
        """Print a table comparing similarity metrics against human judgments"""
        # Get unique model types
        model_types = {
            name.split()[0]: [] for name in self.results.keys()
        }

        # Group results by model type
        for model_name, metrics in self.results.items():
            model_type = model_name.split()[0]
            model_types[model_type].append((model_name, metrics))

        # Headers
        headers = ['Metric'] + list(model_types.keys()) + ['Y true']
        col_widths = [max(len(str(h)), 15) for h in headers]

        # Print header
        header_line = ' | '.join(h.ljust(w) for h, w in zip(headers, col_widths))
        separator = '-' * len(header_line)
        print('\nSimilarity Comparison Results:')
        print(separator)
        print(header_line)
        print(separator)

        # Print MSE row
        mse_row = ['MSE'.ljust(col_widths[0])]
        for model_type in model_types:
            # Get best MSE for this model type
            best_mse = min((m['mse'] for _, m in model_types[model_type]), defau
            mse_row.append(f"{best_mse:.4f}".ljust(col_widths[len(mse_row)]) if
        mse_row.append('1.0000'.ljust(col_widths[-1]))  # Y true column
        print(' | '.join(mse_row))
        print(separator)

    def get_results_dict(self):
        """Return the results dictionary for external use"""
        return self.results

def save_model(model, word2idx, idx2word, model_path, model_type=None):
    """Save model and vocabulary mappings

    Args:
        model: The PyTorch model to save
        word2idx: Word to index mapping
        idx2word: Index to word mapping
        model_path: Base path for saving the model
        model_type: Type of model (skipgram, skipgram_neg, glove)
    """
    os.makedirs(os.path.dirname(model_path), exist_ok=True)

    # Add model type to filename if provided
    if model_type:
        path_parts = os.path.splitext(model_path)
        model_path = f"{path_parts[0]}_{model_type}{path_parts[1]}"

    # Save the PyTorch model
    torch.save({
        'model_state_dict': model.state_dict(),
        'word2idx': word2idx,
        'idx2word': idx2word,
        'embedding_dim': model.embedding_center.embedding_dim,
```

```
        'vocab_size': len(word2idx),
        'model_type': model_type
    }, model_path)
    logger.info(f"Model saved to {model_path}")

def load_model(model_class, model_path):
    """Load model and vocabulary mappings"""
    if not os.path.exists(model_path):
        raise FileNotFoundError(f"Model file not found: {model_path}")

    # Load the saved state
    checkpoint = torch.load(model_path)

    # Create model instance
    model = model_class(checkpoint['vocab_size'], checkpoint['embedding_dim'])
    model.load_state_dict(checkpoint['model_state_dict'])
    model.eval()  # Set to evaluation mode

    return model, checkpoint['word2idx'], checkpoint['idx2word']

def find_similar_words(query, model, word2idx, idx2word, topk=10):
    """Find top-k similar words for a query using the trained model"""
    if isinstance(query, str):
        # Single word query
        if query not in word2idx:
            return []
        query_idx = word2idx[query]
        query_vec = model.embedding_center(torch.LongTensor([query_idx])).detach
    else:
        # Multiple word query - average the vectors
        query_words = query.lower().split()
        vectors = []
        for word in query_words:
            if word in word2idx:
                word_idx = word2idx[word]
                vectors.append(model.embedding_center(torch.LongTensor([word_idx
        if not vectors:
            return []
        query_vec = torch.mean(torch.stack(vectors), dim=0)

    # Calculate similarities with all words
    similarities = []
    for idx in range(len(word2idx)):
        vec = model.embedding_center(torch.LongTensor([idx])).detach()
        sim = torch.nn.functional.cosine_similarity(query_vec, vec)
        similarities.append((idx2word[idx], sim.item()))

    # Sort by similarity and return top k
    similarities.sort(key=lambda x: x[1], reverse=True)
    return similarities[:topk]

class Timer:
    def __enter__(self):
        self.start = time.time()
        return self

    def __exit__(self, *args):
        self.end = time.time()
        self.interval = self.end - self.start
```

Skip-gram

In [7]:

```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import logging
import os
from tqdm import tqdm

# Setup logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    datefmt='%H:%M:%S'
)
logger = logging.getLogger(__name__)

class Skipgram(nn.Module):
    def __init__(self, voc_size, emb_size):
        super(Skipgram, self).__init__()
        self.embedding_center = nn.Embedding(voc_size, emb_size)
        self.embedding_outside = nn.Embedding(voc_size, emb_size)

    def forward(self, center, outside, all_vocabs):
        center_embedding = self.embedding_center(center)
        outside_embedding = self.embedding_outside(outside)
        all_vocabs_embedding = self.embedding_outside(all_vocabs)

        # Calculate loss
        top_term = torch.exp(outside_embedding.bmm(center_embedding.transpose(1,
        lower_term = all_vocabs_embedding.bmm(center_embedding.transpose(1, 2)).
        lower_term_sum = torch.sum(torch.exp(lower_term), 1)

        loss = -torch.mean(torch.log(top_term / lower_term_sum))
        return loss

def create_skipgrams(sentence, window_size):
    skipgrams = []
    for i in range(len(sentence)):
        for w in range(-window_size, window_size + 1):
            context_pos = i + w
            if context_pos < 0 or context_pos >= len(sentence) or context_pos ==
                continue
            skipgrams.append((sentence[i], sentence[context_pos]))
    return skipgrams

def prepare_batch(skipgrams, batch_size, word2idx, vocab_size):
    # Random sample from skipgrams
    indices = np.random.choice(len(skipgrams), batch_size, replace=False)

    centers = [[word2idx.get(skipgrams[i][0], word2idx['<UNK>'])] for i in indic
    outsides = [[word2idx.get(skipgrams[i][1], word2idx['<UNK>'])] for i in indi

    # Convert to tensors
    centers = torch.LongTensor(centers)
    outsides = torch.LongTensor(outsides)
    all_vocabs = torch.arange(vocab_size).expand(batch_size, vocab_size)
```

```python
    return centers, outsides, all_vocabs

def train(corpus, window_size=2, embedding_size=100, batch_size=128, epochs=5):
    logger.info(f"\n{'='*20} Training Configuration {'='*20}")
    logger.info(f"Window Size: {window_size}")
    logger.info(f"Embedding Size: {embedding_size}")
    logger.info(f"Batch Size: {batch_size}")
    logger.info(f"Epochs: {epochs}\n")

    # Prepare data
    logger.info("Preparing training data...")
    tokenized, vocab, word2idx, idx2word = prepare_vocab(corpus)
    logger.info(f"Vocabulary size: {len(vocab)} words")

    # Create skipgrams
    logger.info("Creating skipgrams...")
    all_skipgrams = []
    for sentence in tqdm(tokenized, desc="Processing sentences"):
        all_skipgrams.extend(create_skipgrams(sentence, window_size))
    logger.info(f"Created {len(all_skipgrams)} skipgrams")

    # Initialize model
    model = Skipgram(len(vocab), embedding_size)
    optimizer = optim.Adam(model.parameters())
    logger.info(f"Model parameters: {sum(p.numel() for p in model.parameters()):

    # Load evaluation datasets
    logger.info("Loading evaluation datasets...")
    semantic_pairs, syntactic_pairs = load_word_analogies()
    similarities = load_similarity_dataset()
    logger.info(f"Loaded {len(semantic_pairs)} semantic pairs and {len(syntactic

    # Training metrics
    best_loss = float('inf')
    start_time = time.time()

    logger.info(f"\n{'='*20} Starting Training {'='*20}")

    # Training loop
    for epoch in range(epochs):
        epoch_loss = 0
        batch_count = 0

        # Progress bar for batches
        num_batches = len(all_skipgrams) // batch_size + (1 if len(all_skipgrams
        pbar = tqdm(range(0, len(all_skipgrams), batch_size),
                    desc=f"Epoch {epoch+1}/{epochs}",
                    total=num_batches)

        for i in pbar:
            # Prepare batch
            centers, outsides, all_vocabs = prepare_batch(
                all_skipgrams[i:i+batch_size],
                min(batch_size, len(all_skipgrams) - i),
                word2idx,
                len(vocab)
            )

            # Forward pass
```

```
            loss = model(centers, outsides, all_vocabs)

            # Backward pass
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            # Update metrics
            current_loss = loss.item()
            epoch_loss += current_loss
            batch_count += 1

            # Update progress bar
            pbar.set_postfix({
                'loss': f'{current_loss:.4f}',
                'avg_loss': f'{epoch_loss/batch_count:.4f}'
            })

        # Calculate epoch metrics
        avg_loss = epoch_loss / batch_count

        # Evaluate model
        logger.info(f"\nEvaluating epoch {epoch+1}...")
        with Timer() as eval_timer:
            semantic_acc = evaluate_analogies(model, word2idx, idx2word, semanti
            syntactic_acc = evaluate_analogies(model, word2idx, idx2word, syntac
            similarity_corr, mse, num_pairs = evaluate_similarity(model, word2id

        # Print epoch summary
        logger.info(f"\nEpoch {epoch+1} Summary:")
        logger.info(f"Average Loss: {avg_loss:.4f}")
        logger.info(f"Semantic Accuracy: {semantic_acc:.4f}")
        logger.info(f"Syntactic Accuracy: {syntactic_acc:.4f}")
        logger.info(f"Similarity Correlation: {similarity_corr:.4f}")
        logger.info(f"MSE: {mse:.4f}")
        logger.info(f"Evaluation Time: {eval_timer.interval:.2f}s")

        # Save best model
        if avg_loss < best_loss:
            best_loss = avg_loss
            logger.info("New best model! Saving checkpoint...")
            model_dir = "saved_models"
            os.makedirs(model_dir, exist_ok=True)
            model_path = os.path.join(model_dir, f"w{window_size}_e{embedding_si
            save_model(model, word2idx, idx2word, model_path, model_type="skipgr

    training_time = time.time() - start_time
    logger.info(f"\n{'='*20} Training Complete {'='*20}")
    logger.info(f"Total training time: {training_time:.2f}s")
    logger.info(f"Best loss achieved: {best_loss:.4f}")

    return model, {
        'final_loss': avg_loss,
        'best_loss': best_loss,
        'training_time': training_time,
        'semantic_accuracy': semantic_acc,
        'syntactic_accuracy': syntactic_acc,
        'similarity_correlation': similarity_corr,
        'mse': mse,
        'num_pairs': num_pairs,
```

```python
        'model_path': model_path,
        'word2idx': word2idx,
        'idx2word': idx2word
    }

if __name__ == "__main__":
    # Load corpus
    corpus = load_news_corpus()

    # Initialize evaluator
    evaluator = ModelEvaluator()

    # Train models with different configurations
    configs = [
        {'window_size': 2, 'embedding_size': 100},
        {'window_size': 5, 'embedding_size': 100}
    ]

    for config in configs:
        logger.info(f"\nTraining Skip-gram with config: {config}")
        model, results = train(corpus, **config)

        model_name = f"Skipgram (w={config['window_size']})"
        evaluator.evaluate_model(
            model,
            results['word2idx'],
            results['idx2word'],
            model_name,
            window_size=config['window_size'],
            training_time=results['training_time'],
            final_loss=results['final_loss']
        )

    # Print both tables
    evaluator.print_training_table()
    evaluator.print_similarity_table()
```

```
INFO:__main__:Successfully loaded 203 word pairs from WordSim-353
INFO:__main__:
Training Skip-gram with config: {'window_size': 2, 'embedding_size': 100}
INFO:__main__:
==================== Training Configuration ====================
INFO:__main__:Window Size: 2
INFO:__main__:Embedding Size: 100
INFO:__main__:Batch Size: 128
INFO:__main__:Epochs: 5

INFO:__main__:Preparing training data...
INFO:__main__:Vocabulary size: 2560 words
INFO:__main__:Creating skipgrams...
Processing sentences: 100%|██████████| 4623/4623 [00:00<00:00, 32061.37it/s]
INFO:__main__:Created 374548 skipgrams
INFO:__main__:Model parameters: 512,000
INFO:__main__:Loading evaluation datasets...
INFO:__main__:Successfully loaded 203 word pairs from WordSim-353
INFO:__main__:Loaded 12 semantic pairs and 15 syntactic pairs
INFO:__main__:
==================== Starting Training ====================
Epoch 1/5: 100%|██████████| 2927/2927 [05:19<00:00,  9.17it/s, loss=11.1077, avg_
loss=18.2994]
INFO:__main__:
Evaluating epoch 1...
INFO:__main__:
Epoch 1 Summary:
INFO:__main__:Average Loss: 18.2994
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: -0.0160
INFO:__main__:MSE: 0.2838
INFO:__main__:Evaluation Time: 0.93s
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w2_e100_skipgram.pt
Epoch 2/5: 100%|██████████| 2927/2927 [05:13<00:00,  9.34it/s, loss=7.8520, avg_l
oss=11.2770]
INFO:__main__:
Evaluating epoch 2...
INFO:__main__:
Epoch 2 Summary:
INFO:__main__:Average Loss: 11.2770
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: -0.0385
INFO:__main__:MSE: 0.2732
INFO:__main__:Evaluation Time: 0.98s
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w2_e100_skipgram.pt
Epoch 3/5: 100%|██████████| 2927/2927 [05:30<00:00,  8.86it/s, loss=6.4394, avg_l
oss=8.6451]
INFO:__main__:
Evaluating epoch 3...
INFO:__main__:
Epoch 3 Summary:
INFO:__main__:Average Loss: 8.6451
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: -0.0195
INFO:__main__:MSE: 0.2619
```

```
INFO:__main__:Evaluation Time: 1.11s
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w2_e100_skipgram.pt
Epoch 4/5: 100%|██████████| 2927/2927 [05:32<00:00,  8.80it/s, loss=5.5201, avg_l
oss=7.2730]
INFO:__main__:
Evaluating epoch 4...
INFO:__main__:
Epoch 4 Summary:
INFO:__main__:Average Loss: 7.2730
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: -0.0529
INFO:__main__:MSE: 0.2542
INFO:__main__:Evaluation Time: 1.01s
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w2_e100_skipgram.pt
Epoch 5/5: 100%|██████████| 2927/2927 [05:32<00:00,  8.79it/s, loss=4.8776, avg_l
oss=6.4928]
INFO:__main__:
Evaluating epoch 5...
INFO:__main__:
Epoch 5 Summary:
INFO:__main__:Average Loss: 6.4928
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: -0.0710
INFO:__main__:MSE: 0.2487
INFO:__main__:Evaluation Time: 1.01s
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w2_e100_skipgram.pt
INFO:__main__:
==================== Training Complete ====================
INFO:__main__:Total training time: 1633.94s
INFO:__main__:Best loss achieved: 6.4928
INFO:__main__:
Training Skip-gram with config: {'window_size': 5, 'embedding_size': 100}
INFO:__main__:
==================== Training Configuration ====================
INFO:__main__:Window Size: 5
INFO:__main__:Embedding Size: 100
INFO:__main__:Batch Size: 128
INFO:__main__:Epochs: 5

INFO:__main__:Preparing training data...
INFO:__main__:Vocabulary size: 2560 words
INFO:__main__:Creating skipgrams...
Processing sentences: 100%|██████████| 4623/4623 [00:00<00:00, 19776.78it/s]
INFO:__main__:Created 869448 skipgrams
INFO:__main__:Model parameters: 512,000
INFO:__main__:Loading evaluation datasets...
INFO:__main__:Successfully loaded 203 word pairs from WordSim-353
INFO:__main__:Loaded 12 semantic pairs and 15 syntactic pairs
INFO:__main__:
==================== Starting Training ====================
Epoch 1/5: 100%|██████████| 6793/6793 [12:45<00:00,  8.88it/s, loss=10.6393, avg_
loss=15.1985]
INFO:__main__:
Evaluating epoch 1...
INFO:__main__:
```

```
Epoch 1 Summary:
INFO:__main__:Average Loss: 15.1985
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: -0.0425
INFO:__main__:MSE: 0.2617
INFO:__main__:Evaluation Time: 0.90s
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w5_e100_skipgram.pt
Epoch 2/5: 100%|██████████| 6793/6793 [12:22<00:00,  9.15it/s, loss=7.0392, avg_l
oss=8.5793]
INFO:__main__:
Evaluating epoch 2...
INFO:__main__:
Epoch 2 Summary:
INFO:__main__:Average Loss: 8.5793
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: -0.0057
INFO:__main__:MSE: 0.2487
INFO:__main__:Evaluation Time: 1.23s
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w5_e100_skipgram.pt
Epoch 3/5: 100%|██████████| 6793/6793 [13:09<00:00,  8.60it/s, loss=5.9572, avg_l
oss=6.7026]
INFO:__main__:
Evaluating epoch 3...
INFO:__main__:
Epoch 3 Summary:
INFO:__main__:Average Loss: 6.7026
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: -0.0127
INFO:__main__:MSE: 0.2442
INFO:__main__:Evaluation Time: 0.89s
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w5_e100_skipgram.pt
Epoch 4/5: 100%|██████████| 6793/6793 [12:41<00:00,  8.92it/s, loss=5.4494, avg_l
oss=5.9656]
INFO:__main__:
Evaluating epoch 4...
INFO:__main__:
Epoch 4 Summary:
INFO:__main__:Average Loss: 5.9656
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: -0.0619
INFO:__main__:MSE: 0.2451
INFO:__main__:Evaluation Time: 0.89s
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w5_e100_skipgram.pt
Epoch 5/5: 100%|██████████| 6793/6793 [12:30<00:00,  9.05it/s, loss=5.1846, avg_l
oss=5.6520]
INFO:__main__:
Evaluating epoch 5...
INFO:__main__:
Epoch 5 Summary:
INFO:__main__:Average Loss: 5.6520
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
```

```
INFO:__main__:Similarity Correlation: -0.0731
INFO:__main__:MSE: 0.2474
INFO:__main__:Evaluation Time: 0.89s
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w5_e100_skipgram.pt
INFO:__main__:
==================== Training Complete ====================
INFO:__main__:Total training time: 3814.83s
INFO:__main__:Best loss achieved: 5.6520
Training and Accuracy Results:
-----------------------------------------------------------------------------------
------------------------
Model           | Window Size       | Training Loss     | Training Time     | Syntactic
Acc     | Semantic Acc
-----------------------------------------------------------------------------------
------------------------
Skipgram (w=2)  | 2                 | 6.4928            | 1633.94s          | 0.0000
| 0.0000
Skipgram (w=5)  | 5                 | 5.6520            | 3814.83s          | 0.0000
| 0.0000
-----------------------------------------------------------------------------------
------------------------

Similarity Comparison Results:
---------------------------------------------------
Metric          | Skipgram          | Y true
---------------------------------------------------
MSE             | 0.2474            | 1.0000
---------------------------------------------------
```

Trying the similar words test function for skip-gram model. Later part of the code has the testing done for all models in one script itself

In [21]:

```python
import torch
import logging
from pathlib import Path
from tabulate import tabulate

# Setup logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    datefmt='%H:%M:%S'
)
logger = logging.getLogger(__name__)

def display_similar_words(query_words, model, word2idx, idx2word, top_k=10):
    """Display similar words for the Skip-gram model

    Args:
        query_words (list): List of words to find similar words for
        model: The Skip-gram model
        word2idx (dict): Word to index mapping
        idx2word (dict): Index to word mapping
        top_k (int): Number of similar words to display
    """
    for query in query_words:
        print(f"\nSimilar words to '{query}':")
        print("-" * 60)
```

```python
        if query not in word2idx:
            logger.warning(f"Word '{query}' not in vocabulary")
            continue

        try:
            # Use the find_similar_words function from utils.py
            similar_words = find_similar_words(query, model, word2idx, idx2word,

            if not similar_words:
                logger.warning(f"No similar words found for '{query}'")
                continue

            # Prepare table data
            table_data = []
            for i, (word, sim) in enumerate(similar_words, 1):
                if word != query:  # Don't show the query word itself
                    table_data.append([f"{i}", word, f"{sim:.4f}"])

            # Print table
            headers = ["Rank", "Word", "Similarity"]
            print(tabulate(table_data, headers=headers, tablefmt="grid"))
        except Exception as e:
            logger.error(f"Error finding similar words: {str(e)}")
            continue
        print()

def main():
    # Model path - update this to your actual model path
    model_path = "/home/jupyter-st125462/NLP/A1/saved_models/w2_e100_skipgram.pt

    # Load Skip-gram model
    try:
        logger.info(f"Loading model from: {model_path}")
        model, word2idx, idx2word = load_model(Skipgram, model_path)
        model.eval()  # Set to evaluation mode
    except Exception as e:
        logger.error(f"Failed to load model: {str(e)}")
        return

    # Query words to test
    query_words = [
        # Common words
        "king", "computer", "good", "day", "time", "person", "world", "work",
        # Domain-specific
        "data", "algorithm", "network", "science",
        # Technical terms
        "python", "machine", "learning", "artificial"
    ]

    # Display similar words
    display_similar_words(query_words, model, word2idx, idx2word)

if __name__ == "__main__":
    main()
```

```
INFO:__main__:Loading model from: /home/jupyter-st125462/NLP/A1/saved_models/w2_e
100_skipgram.pt
/tmp/ipykernel_762838/3318003762.py:379: FutureWarning: You are using `torch.load
` with `weights_only=False` (the current default value), which uses the default p
ickle module implicitly. It is possible to construct malicious pickle data which
will execute arbitrary code during unpickling (See https://github.com/pytorch/pyt
orch/blob/main/SECURITY.md#untrusted-models for more details). In a future releas
e, the default value for `weights_only` will be flipped to `True`. This limits th
e functions that could be executed during unpickling. Arbitrary objects will no l
onger be allowed to be loaded via this mode unless they are explicitly allowliste
d by the user via `torch.serialization.add_safe_globals`. We recommend you start
setting `weights_only=True` for any use case where you don't have full control of
the loaded file. Please open an issue on GitHub for any issues related to this ex
perimental feature.
  checkpoint = torch.load(model_path)
WARNING:__main__:Word 'king' not in vocabulary
WARNING:__main__:Word 'computer' not in vocabulary
```

```
Similar words to 'king':
------------------------------------------------------------

Similar words to 'computer':
------------------------------------------------------------

Similar words to 'good':
------------------------------------------------------------
+--------+------------+--------------+
|   Rank | Word       |   Similarity |
+========+============+==============+
|      2 | brevard    |       0.3579 |
+--------+------------+--------------+
|      3 | nomination |       0.3514 |
+--------+------------+--------------+
|      4 | setting    |       0.3222 |
+--------+------------+--------------+
|      5 | important  |       0.3195 |
+--------+------------+--------------+
|      6 | c.         |       0.3181 |
+--------+------------+--------------+
|      7 | group      |       0.3123 |
+--------+------------+--------------+
|      8 | increased  |       0.3071 |
+--------+------------+--------------+
|      9 | table      |       0.3063 |
+--------+------------+--------------+
|     10 | clark      |       0.3012 |
+--------+------------+--------------+


Similar words to 'day':
------------------------------------------------------------
+--------+-------------+--------------+
|   Rank | Word        |   Similarity |
+========+=============+==============+
|      2 | christmas   |       0.3743 |
+--------+-------------+--------------+
|      3 | problems    |       0.3686 |
+--------+-------------+--------------+
|      4 | calls       |       0.348  |
+--------+-------------+--------------+
|      5 | address     |       0.3348 |
+--------+-------------+--------------+
|      6 | order       |       0.3309 |
+--------+-------------+--------------+
|      7 | immediate   |       0.3283 |
+--------+-------------+--------------+
|      8 | p.m.        |       0.3258 |
+--------+-------------+--------------+
|      9 | opportunity |       0.3223 |
+--------+-------------+--------------+
|     10 | informed    |       0.3212 |
+--------+-------------+--------------+


Similar words to 'time':
------------------------------------------------------------
+--------+----------+--------------+
|   Rank | Word     |   Similarity |
```

2	not	0.5119
3	victory	0.398
4	will	0.3484
5	about	0.3436
6	large	0.3395
7	homes	0.3289
8	raising	0.3279
9	proposed	0.3273
10	previous	0.3266

Similar words to 'person':

Rank	Word	Similarity
2	1959	0.4341
3	comes	0.3612
4	nine	0.3565
5	produced	0.3518
6	cost	0.3374
7	personnel	0.3316
8	over	0.3164
9	board	0.3083
10	only	0.308

Similar words to 'world':

Rank	Word	Similarity
2	opinion	0.4198
3	marr	0.372
4	he	0.3483
5	long	0.3454
6	entering	0.3378

```
+--------+----------+--------------+
|      7 | scene    |       0.3341 |
+--------+----------+--------------+
|      8 | last     |       0.3122 |
+--------+----------+--------------+
|      9 | very     |       0.3119 |
+--------+----------+--------------+
|     10 | i        |       0.3063 |
+--------+----------+--------------+


Similar words to 'work':
------------------------------------------------------------
WARNING:__main__:Word 'data' not in vocabulary
WARNING:__main__:Word 'algorithm' not in vocabulary
WARNING:__main__:Word 'network' not in vocabulary
WARNING:__main__:Word 'python' not in vocabulary
```

Rank	Word	Similarity
2	income	0.3792
3	for	0.3658
4	earnings	0.356
5	posts	0.3471
6	bob	0.3403
7	post	0.3355
8	camp	0.3307
9	orders	0.3253
10	out	0.3233

Similar words to 'data':

Similar words to 'algorithm':

Similar words to 'network':

Similar words to 'science':

Rank	Word	Similarity
2	works	0.3314
3	congolese	0.32
4	sheriff	0.3009
5	weather	0.2996
6	stressed	0.289
7	right	0.2847
8	driven	0.2812
9	declared	0.2792
10	leader	0.278

Similar words to 'python':

```
Similar words to 'machine':
------------------------------------------------------------
WARNING:__main__:Word 'artificial' not in vocabulary
+--------+-------------+--------------+
|   Rank | Word        |   Similarity |
+========+=============+==============+
|      2 | hope        |       0.3684 |
+--------+-------------+--------------+
|      3 | threat      |       0.3361 |
+--------+-------------+--------------+
|      4 | people      |       0.3242 |
+--------+-------------+--------------+
|      5 | boston      |       0.3204 |
+--------+-------------+--------------+
|      6 | recommended |       0.3133 |
+--------+-------------+--------------+
|      7 | yesterday   |       0.3086 |
+--------+-------------+--------------+
|      8 | under       |       0.3083 |
+--------+-------------+--------------+
|      9 | kitchen     |       0.2988 |
+--------+-------------+--------------+
|     10 | business    |       0.2967 |
+--------+-------------+--------------+


Similar words to 'learning':
------------------------------------------------------------
+--------+-----------+--------------+
|   Rank | Word      |   Similarity |
+========+===========+==============+
|      2 | rose      |       0.3376 |
+--------+-----------+--------------+
|      3 | criminal  |       0.3365 |
+--------+-----------+--------------+
|      4 | wisdom    |       0.3349 |
+--------+-----------+--------------+
|      5 | tshombe   |       0.3134 |
+--------+-----------+--------------+
|      6 | time      |       0.3044 |
+--------+-----------+--------------+
|      7 | going     |       0.3033 |
+--------+-----------+--------------+
|      8 | each      |       0.2848 |
+--------+-----------+--------------+
|      9 | joan      |       0.2836 |
+--------+-----------+--------------+
|     10 | furniture |       0.2828 |
+--------+-----------+--------------+


Similar words to 'artificial':
------------------------------------------------------------
```

Skip-gram (Negative Sampling)

In [32]:
```python
import numpy as np
import torch
```

```python
import torch.nn as nn
import torch.optim as optim
import logging
import random
import os
import time
from collections import Counter
from tqdm import tqdm

# Setup logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    datefmt='%H:%M:%S'
)
logger = logging.getLogger(__name__)

class SkipgramNeg(nn.Module):
    def __init__(self, voc_size, emb_size):
        super(SkipgramNeg, self).__init__()
        self.embedding_center = nn.Embedding(voc_size, emb_size)
        self.embedding_outside = nn.Embedding(voc_size, emb_size)
        self.logsigmoid = nn.LogSigmoid()

    def forward(self, center, outside, negative):
        # Get embeddings
        center_embed = self.embedding_center(center)
        outside_embed = self.embedding_outside(outside)
        neg_embed = self.embedding_outside(negative)

        # Positive score
        pos_score = self.logsigmoid(torch.sum(outside_embed * center_embed, dim=

        # Negative score
        neg_score = self.logsigmoid(-torch.bmm(neg_embed, center_embed.transpose
        neg_score = torch.sum(neg_score, dim=1)

        loss = -(pos_score + neg_score).mean()
        return loss

def create_unigram_table(word_counts, vocab_size, table_size=1e6):
    pow_freq = np.array(list(word_counts.values())) ** 0.75
    power_sum = sum(pow_freq)
    ratio = pow_freq / power_sum
    count = np.round(ratio * table_size)

    table = []
    for idx, x in enumerate(count):
        # Ensure idx is within vocabulary range
        if idx < vocab_size:
            table.extend([idx] * int(x))
    return table

def negative_sampling(targets, unigram_table, k, vocab_size):
    batch_size = targets.shape[0]
    neg_samples = []

    for i in range(batch_size):
        negs = []
        target_idx = targets[i].item()
```

```python
        while len(negs) < k:
            neg = random.choice(unigram_table)
            # Make sure the negative sample is within vocabulary range
            if neg != target_idx and neg < vocab_size:
                negs.append(neg)
        neg_samples.append(negs)

    return torch.LongTensor(neg_samples)

def create_skipgrams(sentence, window_size):
    skipgrams = []
    for i in range(len(sentence)):
        for w in range(-window_size, window_size + 1):
            context_pos = i + w
            if context_pos < 0 or context_pos >= len(sentence) or context_pos ==
                continue
            skipgrams.append((sentence[i], sentence[context_pos]))
    return skipgrams

def prepare_batch(skipgrams, batch_size, word2idx, unigram_table, neg_samples=5)
    # Random sample from skipgrams
    indices = np.random.choice(len(skipgrams), batch_size, replace=False)

    centers = [[word2idx.get(skipgrams[i][0], word2idx['<UNK>'])] for i in indic
    outsides = [[word2idx.get(skipgrams[i][1], word2idx['<UNK>'])] for i in indi

    # Convert to tensors
    centers = torch.LongTensor(centers)
    outsides = torch.LongTensor(outsides)

    # Generate negative samples
    negative = negative_sampling(outsides.squeeze(), unigram_table, neg_samples,

    return centers, outsides, negative

def train(corpus, window_size=2, embedding_size=100, neg_samples=5, batch_size=1
    """Train the Skip-gram model with negative sampling"""
    logger.info(f"\n{'='*20} Training Configuration {'='*20}")
    logger.info(f"Window Size: {window_size}")
    logger.info(f"Embedding Size: {embedding_size}")
    logger.info(f"Negative Samples: {neg_samples}")
    logger.info(f"Batch Size: {batch_size}")
    logger.info(f"Epochs: {epochs}\n")

    # Prepare data
    logger.info("Preparing training data...")
    tokenized, vocab, word2idx, idx2word = prepare_vocab(corpus)
    logger.info(f"Vocabulary size: {len(vocab)} words")

    # Create skipgrams
    logger.info("Creating skipgrams...")
    all_skipgrams = []
    for sentence in tqdm(tokenized, desc="Processing sentences"):
        all_skipgrams.extend(create_skipgrams(sentence, window_size))
    logger.info(f"Created {len(all_skipgrams)} skipgrams")

    # Create unigram table for negative sampling
    logger.info("Creating unigram table...")
    word_counts = Counter([word for sent in tokenized for word in sent])
    unigram_table = create_unigram_table(word_counts, len(vocab))
```

```python
    logger.info(f"Created unigram table with {len(unigram_table)} entries")

    # Load evaluation datasets
    logger.info("Loading evaluation datasets...")
    semantic_pairs, syntactic_pairs = load_word_analogies()
    similarities = load_similarity_dataset()
    logger.info(f"Loaded {len(semantic_pairs)} semantic pairs and {len(syntactic

    # Initialize model
    model = SkipgramNeg(len(vocab), embedding_size)
    optimizer = optim.Adam(model.parameters())
    logger.info(f"Model parameters: {sum(p.numel() for p in model.parameters()):

    # Training metrics
    best_loss = float('inf')
    losses = []
    start_time = time.time()

    logger.info(f"\n{'='*20} Starting Training {'='*20}")

    # Training loop
    for epoch in range(epochs):
        epoch_loss = 0
        batch_count = 0

        # Progress bar for batches
        num_batches = len(all_skipgrams) // batch_size + (1 if len(all_skipgrams
        pbar = tqdm(range(0, len(all_skipgrams), batch_size),
                    desc=f"Epoch {epoch+1}/{epochs}",
                    total=num_batches)

        for i in pbar:
            # Prepare batch
            centers, outsides, negative = prepare_batch(
                all_skipgrams[i:i+batch_size],
                min(batch_size, len(all_skipgrams) - i),
                word2idx,
                unigram_table,
                neg_samples
            )

            # Forward pass
            loss = model(centers, outsides, negative)

            # Backward pass
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            # Update metrics
            current_loss = loss.item()
            epoch_loss += current_loss
            batch_count += 1

            # Update progress bar
            pbar.set_postfix({'loss': f'{current_loss:.4f}'})

        # Calculate average loss for epoch
        avg_loss = epoch_loss / batch_count
        losses.append(avg_loss)
```

```
        # Evaluate model
        logger.info(f"\nEvaluating epoch {epoch+1}...")
        semantic_acc = evaluate_analogies(model, word2idx, idx2word, semantic_pa
        syntactic_acc = evaluate_analogies(model, word2idx, idx2word, syntactic_
        similarity_corr, mse, num_pairs = evaluate_similarity(model, word2idx, s

        # Print epoch summary
        logger.info(f"\nEpoch {epoch+1} Summary:")
        logger.info(f"Average Loss: {avg_loss:.4f}")
        logger.info(f"Semantic Accuracy: {semantic_acc:.4f}")
        logger.info(f"Syntactic Accuracy: {syntactic_acc:.4f}")
        logger.info(f"Similarity Correlation: {similarity_corr:.4f}")
        logger.info(f"MSE: {mse:.4f}")

        # Save best model
        if avg_loss < best_loss:
            best_loss = avg_loss
            logger.info("New best model! Saving checkpoint...")
            model_dir = "saved_models"
            os.makedirs(model_dir, exist_ok=True)
            model_path = os.path.join(model_dir, f"w{window_size}_e{embedding_si
            save_model(model, word2idx, idx2word, model_path, model_type="skipgr

    training_time = time.time() - start_time
    logger.info(f"Training Time: {training_time:.2f}s")

    return model, {
        'word2idx': word2idx,
        'idx2word': idx2word,
        'losses': losses,
        'training_time': training_time,
        'final_loss': losses[-1] if losses else None,
        'best_loss': best_loss,
        'semantic_accuracy': semantic_acc,
        'syntactic_accuracy': syntactic_acc,
        'similarity_correlation': similarity_corr,
        'mse': mse,
        'num_pairs': num_pairs,
        'model_path': model_path
    }

if __name__ == "__main__":
    # Load corpus
    corpus = load_news_corpus()

    # Initialize evaluator
    evaluator = ModelEvaluator()

    # Training configurations
    configs = [
        {
            'window_size': 2,
            'embedding_size': 100,
            'neg_samples': 5,
            'batch_size': 128,
            'epochs': 5
        },
        {
            'window_size': 5,
```

```
            'embedding_size': 100,
            'neg_samples': 10,
            'batch_size': 128,
            'epochs': 5
        }
    ]

    # Train and evaluate models
    for config in configs:
        logger.info(f"\nTraining Skip-gram Negative Sampling with config: {confi
        model, results = train(corpus, **config)

        model_name = f"Skipgram-NEG (w={config['window_size']}, n={config['neg_s
        evaluator.evaluate_model(
            model,
            results['word2idx'],
            results['idx2word'],
            model_name,
            window_size=config['window_size'],
            training_time=results['training_time'],
            final_loss=results['final_loss']
        )

    # Print evaluation results
    logger.info("\nTraining Metrics:")
    evaluator.print_training_table()

    logger.info("\nSimilarity Metrics:")
    evaluator.print_similarity_table()
```

```
INFO:__main__:Successfully loaded 203 word pairs from WordSim-353
INFO:__main__:
Training Skip-gram Negative Sampling with config: {'window_size': 2, 'embedding_s
ize': 100, 'neg_samples': 5, 'batch_size': 128, 'epochs': 5}
INFO:__main__:
==================== Training Configuration ====================
INFO:__main__:Window Size: 2
INFO:__main__:Embedding Size: 100
INFO:__main__:Negative Samples: 5
INFO:__main__:Batch Size: 128
INFO:__main__:Epochs: 5

INFO:__main__:Preparing training data...
INFO:__main__:Vocabulary size: 2560 words
INFO:__main__:Creating skipgrams...
Processing sentences: 100%|██████████| 4623/4623 [00:00<00:00, 31570.27it/s]
INFO:__main__:Created 374548 skipgrams
INFO:__main__:Creating unigram table...
INFO:__main__:Created unigram table with 538072 entries
INFO:__main__:Loading evaluation datasets...
INFO:__main__:Successfully loaded 203 word pairs from WordSim-353
INFO:__main__:Loaded 12 semantic pairs and 15 syntactic pairs
INFO:__main__:Model parameters: 512,000
INFO:__main__:
==================== Starting Training ====================
Epoch 1/5: 100%|██████████| 2927/2927 [00:29<00:00, 97.68it/s, loss=10.9655]
INFO:__main__:
Evaluating epoch 1...
INFO:__main__:
Epoch 1 Summary:
INFO:__main__:Average Loss: 13.9636
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: 0.1466
INFO:__main__:MSE: 0.2538
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w2_e100_skipgram_neg_skipgram_neg.pt
Epoch 2/5: 100%|██████████| 2927/2927 [00:30<00:00, 96.98it/s, loss=9.7618]
INFO:__main__:
Evaluating epoch 2...
INFO:__main__:
Epoch 2 Summary:
INFO:__main__:Average Loss: 7.4460
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: 0.1628
INFO:__main__:MSE: 0.2419
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w2_e100_skipgram_neg_skipgram_neg.pt
Epoch 3/5: 100%|██████████| 2927/2927 [00:30<00:00, 95.96it/s, loss=5.5101]
INFO:__main__:
Evaluating epoch 3...
INFO:__main__:
Epoch 3 Summary:
INFO:__main__:Average Loss: 4.7757
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: 0.1648
INFO:__main__:MSE: 0.2222
INFO:__main__:New best model! Saving checkpoint...
```

```
INFO:__main__:Model saved to saved_models/w2_e100_skipgram_neg_skipgram_neg.pt
Epoch 4/5: 100%|██████████| 2927/2927 [00:30<00:00, 96.97it/s, loss=3.8024]
INFO:__main__:
Evaluating epoch 4...
INFO:__main__:
Epoch 4 Summary:
INFO:__main__:Average Loss: 3.4547
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: 0.2017
INFO:__main__:MSE: 0.2021
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w2_e100_skipgram_neg_skipgram_neg.pt
Epoch 5/5: 100%|██████████| 2927/2927 [00:29<00:00, 97.97it/s, loss=2.5063]
INFO:__main__:
Evaluating epoch 5...
INFO:__main__:
Epoch 5 Summary:
INFO:__main__:Average Loss: 2.7663
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: 0.1797
INFO:__main__:MSE: 0.1879
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w2_e100_skipgram_neg_skipgram_neg.pt
INFO:__main__:Training Time: 155.74s
INFO:__main__:
Training Skip-gram Negative Sampling with config: {'window_size': 5, 'embedding_s
ize': 100, 'neg_samples': 10, 'batch_size': 128, 'epochs': 5}
INFO:__main__:
==================== Training Configuration ====================
INFO:__main__:Window Size: 5
INFO:__main__:Embedding Size: 100
INFO:__main__:Negative Samples: 10
INFO:__main__:Batch Size: 128
INFO:__main__:Epochs: 5

INFO:__main__:Preparing training data...
INFO:__main__:Vocabulary size: 2560 words
INFO:__main__:Creating skipgrams...
Processing sentences: 100%|██████████| 4623/4623 [00:00<00:00, 22106.04it/s]
INFO:__main__:Created 869448 skipgrams
INFO:__main__:Creating unigram table...
INFO:__main__:Created unigram table with 538072 entries
INFO:__main__:Loading evaluation datasets...
INFO:__main__:Successfully loaded 203 word pairs from WordSim-353
INFO:__main__:Loaded 12 semantic pairs and 15 syntactic pairs
INFO:__main__:Model parameters: 512,000
INFO:__main__:
==================== Starting Training ====================
Epoch 1/5: 100%|██████████| 6793/6793 [01:14<00:00, 91.59it/s, loss=10.5298]
INFO:__main__:
Evaluating epoch 1...
INFO:__main__:
Epoch 1 Summary:
INFO:__main__:Average Loss: 16.6846
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: -0.1166
INFO:__main__:MSE: 0.2373
```

```
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w5_e100_skipgram_neg_skipgram_neg.pt
Epoch 2/5: 100%|██████████| 6793/6793 [01:14<00:00, 90.71it/s, loss=3.4957]
INFO:__main__:
Evaluating epoch 2...
INFO:__main__:
Epoch 2 Summary:
INFO:__main__:Average Loss: 5.2603
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: -0.1013
INFO:__main__:MSE: 0.1965
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w5_e100_skipgram_neg_skipgram_neg.pt
Epoch 3/5: 100%|██████████| 6793/6793 [01:13<00:00, 92.56it/s, loss=3.2153]
INFO:__main__:
Evaluating epoch 3...
INFO:__main__:
Epoch 3 Summary:
INFO:__main__:Average Loss: 3.2990
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: -0.0730
INFO:__main__:MSE: 0.1860
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w5_e100_skipgram_neg_skipgram_neg.pt
Epoch 4/5: 100%|██████████| 6793/6793 [01:14<00:00, 90.89it/s, loss=2.2040]
INFO:__main__:
Evaluating epoch 4...
INFO:__main__:
Epoch 4 Summary:
INFO:__main__:Average Loss: 2.7308
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: -0.1128
INFO:__main__:MSE: 0.1935
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w5_e100_skipgram_neg_skipgram_neg.pt
Epoch 5/5: 100%|██████████| 6793/6793 [01:12<00:00, 93.74it/s, loss=1.9465]
INFO:__main__:
Evaluating epoch 5...
INFO:__main__:
Epoch 5 Summary:
INFO:__main__:Average Loss: 2.4614
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: -0.1201
INFO:__main__:MSE: 0.2002
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w5_e100_skipgram_neg_skipgram_neg.pt
INFO:__main__:Training Time: 374.31s
INFO:__main__:
Training Metrics:
INFO:__main__:
Similarity Metrics:
```

```
Training and Accuracy Results:
--------------------------------------------------------------------------------
--------------------------------
Model                    | Window Size     | Training Loss    | Training Time    |
Syntactic Acc    | Semantic Acc
--------------------------------------------------------------------------------
--------------------------------
Skipgram-NEG (w=2, n=5)  | 2               | 2.7663           | 155.74s          |
0.0000           | 0.0000
Skipgram-NEG (w=5, n=10) | 5               | 2.4614           | 374.31s          |
0.0000           | 0.0000
--------------------------------------------------------------------------------
--------------------------------

Similarity Comparison Results:
---------------------------------------------------
Metric           | Skipgram-NEG     | Y true
---------------------------------------------------
MSE              | 0.1879           | 1.0000
---------------------------------------------------
```

GloVe

In [37]:

```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import logging
import os
import time
from collections import defaultdict
from tqdm import tqdm

# Setup logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    datefmt='%H:%M:%S'
)
logger = logging.getLogger(__name__)

class GloVe(nn.Module):
    def __init__(self, voc_size, emb_size):
        super(GloVe, self).__init__()
        self.embedding_center = nn.Embedding(voc_size, emb_size)
        self.embedding_outside = nn.Embedding(voc_size, emb_size)
        self.center_bias = nn.Embedding(voc_size, 1)
        self.outside_bias = nn.Embedding(voc_size, 1)

    def forward(self, center, outside, coocs, weighting):
        center_embed = self.embedding_center(center)
        outside_embed = self.embedding_outside(outside)

        center_bias = self.center_bias(center).squeeze()
        outside_bias = self.outside_bias(outside).squeeze()

        inner_product = torch.sum(center_embed * outside_embed, dim=2).squeeze()
```

```python
        prediction = inner_product + center_bias + outside_bias

        loss = weighting * torch.pow(prediction - torch.log(coocs), 2)
        return torch.mean(loss)

def build_cooccurrence_matrix(tokenized, vocab_size, word2idx, window_size=5):
    """Build word co-occurrence matrix"""
    logger.info("Building co-occurrence matrix...")
    cooccurrence = defaultdict(float)

    for sentence in tqdm(tokenized, desc="Processing sentences"):
        for center_pos, center_word in enumerate(sentence):
            center_idx = word2idx.get(center_word, word2idx['<UNK>'])

            # For each context word in window
            for context_pos in range(
                max(0, center_pos - window_size),
                min(len(sentence), center_pos + window_size + 1)
            ):
                if context_pos != center_pos:
                    context_word = sentence[context_pos]
                    context_idx = word2idx.get(context_word, word2idx['<UNK>'])
                    distance = abs(context_pos - center_pos)
                    cooccurrence[(center_idx, context_idx)] += 1.0 / distance

    logger.info(f"Created co-occurrence matrix with {len(cooccurrence)} non-zerc
    return cooccurrence

def train(corpus, window_size=5, embedding_size=100, x_max=100, alpha=0.75, batc
    """Train the GloVe model"""
    logger.info(f"\n{'='*20} Training Configuration {'='*20}")
    logger.info(f"Window Size: {window_size}")
    logger.info(f"Embedding Size: {embedding_size}")
    logger.info(f"X_max: {x_max}")
    logger.info(f"Alpha: {alpha}")
    logger.info(f"Batch Size: {batch_size}")
    logger.info(f"Epochs: {epochs}\n")

    # Prepare data
    logger.info("Preparing training data...")
    tokenized, vocab, word2idx, idx2word = prepare_vocab(corpus)
    logger.info(f"Vocabulary size: {len(vocab)} words")

    # Build co-occurrence matrix
    cooc_matrix = build_cooccurrence_matrix(tokenized, len(vocab), word2idx, wir

    # Initialize model
    model = GloVe(len(vocab), embedding_size)
    optimizer = optim.Adam(model.parameters())
    logger.info(f"Model parameters: {sum(p.numel() for p in model.parameters()):

    # Load evaluation datasets
    logger.info("Loading evaluation datasets...")
    semantic_pairs, syntactic_pairs = load_word_analogies()
    similarities = load_similarity_dataset()
    logger.info(f"Loaded {len(semantic_pairs)} semantic pairs and {len(syntactic

    # Training metrics
    best_loss = float('inf')
    losses = []
```

```python
    start_time = time.time()

    logger.info(f"\n{'='*20} Starting Training {'='*20}")

    # Training Loop
    for epoch in range(epochs):
        total_loss = 0
        batch_count = 0

        # Create batches from co-occurrence matrix
        training_pairs = []
        with tqdm(total=len(cooc_matrix), desc="Creating training pairs") as pba
            for (i, j), xij in cooc_matrix.items():
                if xij > 0:
                    training_pairs.append((i, j, xij))
                    pbar.update(1)

        # Shuffle training pairs
        np.random.shuffle(training_pairs)

        # Progress bar for batches
        num_batches = len(training_pairs) // batch_size + (1 if len(training_pai
        pbar = tqdm(range(0, len(training_pairs), batch_size),
                   desc=f"Epoch {epoch+1}/{epochs}",
                   total=num_batches)

        for i in pbar:
            # Get batch
            batch = training_pairs[i:i + batch_size]

            # Convert to tensors
            i_batch = torch.LongTensor([x[0] for x in batch]).unsqueeze(1)
            j_batch = torch.LongTensor([x[1] for x in batch]).unsqueeze(1)
            xij_batch = torch.FloatTensor([x[2] for x in batch])

            # Weight function
            weights = torch.pow(xij_batch / x_max, alpha)
            weights[xij_batch > x_max] = 1

            # Forward pass
            loss = model(i_batch, j_batch, xij_batch, weights)

            # Backward pass
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            # Update metrics
            current_loss = loss.item()
            total_loss += current_loss
            batch_count += 1

            # Update progress bar
            pbar.set_postfix({'loss': f'{current_loss:.4f}'})

        # Calculate average loss for epoch
        avg_loss = total_loss / batch_count
        losses.append(avg_loss)

        # Evaluate model
```

```
        logger.info(f"\nEvaluating epoch {epoch+1}...")
        semantic_acc = evaluate_analogies(model, word2idx, idx2word, semantic_pa
        syntactic_acc = evaluate_analogies(model, word2idx, idx2word, syntactic_
        similarity_corr, mse, num_pairs = evaluate_similarity(model, word2idx, s

        # Print epoch summary
        logger.info(f"\nEpoch {epoch+1} Summary:")
        logger.info(f"Average Loss: {avg_loss:.4f}")
        logger.info(f"Semantic Accuracy: {semantic_acc:.4f}")
        logger.info(f"Syntactic Accuracy: {syntactic_acc:.4f}")
        logger.info(f"Similarity Correlation: {similarity_corr:.4f}")
        logger.info(f"MSE: {mse:.4f}")

        # Save best model
        if avg_loss < best_loss:
            best_loss = avg_loss
            logger.info("New best model! Saving checkpoint...")
            model_dir = "saved_models"
            os.makedirs(model_dir, exist_ok=True)
            model_path = os.path.join(model_dir, f"w{window_size}_e{embedding_si
            save_model(model, word2idx, idx2word, model_path, model_type="glove"

    training_time = time.time() - start_time
    logger.info(f"\n{'='*20} Training Complete {'='*20}")
    logger.info(f"Total training time: {training_time:.2f}s")
    logger.info(f"Best loss achieved: {best_loss:.4f}")

    return model, {
        'word2idx': word2idx,
        'idx2word': idx2word,
        'losses': losses,
        'training_time': training_time,
        'final_loss': losses[-1] if losses else None,
        'best_loss': best_loss,
        'semantic_accuracy': semantic_acc,
        'syntactic_accuracy': syntactic_acc,
        'similarity_correlation': similarity_corr,
        'mse': mse,
        'num_pairs': num_pairs,
        'model_path': model_path
    }

if __name__ == "__main__":
    # Load corpus
    corpus = load_news_corpus()

    # Initialize evaluator
    evaluator = ModelEvaluator()

    # Training configurations
    configs = [
        {
            'window_size': 2,
            'embedding_size': 100,
            'x_max': 100,
            'alpha': 0.75,
            'batch_size': 128,
            'epochs': 5
        },
        {
```

```python
            'window_size': 5,
            'embedding_size': 100,
            'x_max': 100,
            'alpha': 0.75,
            'batch_size': 128,
            'epochs': 5
        },
        {
            'window_size': 10,
            'embedding_size': 100,
            'x_max': 100,
            'alpha': 0.75,
            'batch_size': 128,
            'epochs': 5
        }
    ]

    # Train and evaluate models
    for config in configs:
        logger.info(f"\nTraining GloVe with config: {config}")
        model, results = train(corpus, **config)

        model_name = f"GloVe (w={config['window_size']}, α={config['alpha']})"
        evaluator.evaluate_model(
            model,
            results['word2idx'],
            results['idx2word'],
            model_name,
            window_size=config['window_size'],
            training_time=results['training_time'],
            final_loss=results['final_loss']
        )

    # Print evaluation results
    logger.info("\nTraining Metrics:")
    evaluator.print_training_table()

    logger.info("\nSimilarity Metrics:")
    evaluator.print_similarity_table()
```

```
INFO:__main__:Successfully loaded 203 word pairs from WordSim-353
INFO:__main__:
Training GloVe with config: {'window_size': 2, 'embedding_size': 100, 'x_max': 10
0, 'alpha': 0.75, 'batch_size': 128, 'epochs': 5}
INFO:__main__:
==================== Training Configuration ====================
INFO:__main__:Window Size: 2
INFO:__main__:Embedding Size: 100
INFO:__main__:X_max: 100
INFO:__main__:Alpha: 0.75
INFO:__main__:Batch Size: 128
INFO:__main__:Epochs: 5

INFO:__main__:Preparing training data...
INFO:__main__:Vocabulary size: 2560 words
INFO:__main__:Building co-occurrence matrix...
Processing sentences: 100%|██████████| 4623/4623 [00:00<00:00, 12175.00it/s]
INFO:__main__:Created co-occurrence matrix with 113821 non-zero entries
INFO:__main__:Model parameters: 517,120
INFO:__main__:Loading evaluation datasets...
INFO:__main__:Successfully loaded 203 word pairs from WordSim-353
INFO:__main__:Loaded 12 semantic pairs and 15 syntactic pairs
INFO:__main__:
==================== Starting Training ====================
Creating training pairs: 100%|██████████| 113821/113821 [00:00<00:00, 1500276.16i
t/s]
Epoch 1/5: 100%|██████████| 890/890 [00:07<00:00, 126.46it/s, loss=2.0785]
INFO:__main__:
Evaluating epoch 1...
INFO:__main__:
Epoch 1 Summary:
INFO:__main__:Average Loss: 4.2792
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: 0.0586
INFO:__main__:MSE: 0.2687
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w2_e100_glove_glove.pt
Creating training pairs: 100%|██████████| 113821/113821 [00:00<00:00, 1408358.93i
t/s]
Epoch 2/5: 100%|██████████| 890/890 [00:06<00:00, 129.65it/s, loss=1.3420]
INFO:__main__:
Evaluating epoch 2...
INFO:__main__:
Epoch 2 Summary:
INFO:__main__:Average Loss: 3.3109
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: 0.0478
INFO:__main__:MSE: 0.2681
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w2_e100_glove_glove.pt
Creating training pairs: 100%|██████████| 113821/113821 [00:00<00:00, 1538203.37i
t/s]
Epoch 3/5: 100%|██████████| 890/890 [00:07<00:00, 121.07it/s, loss=3.6699]
INFO:__main__:
Evaluating epoch 3...
INFO:__main__:
Epoch 3 Summary:
INFO:__main__:Average Loss: 2.5782
```

```
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: 0.0376
INFO:__main__:MSE: 0.2676
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w2_e100_glove_glove.pt
Creating training pairs: 100%|██████████| 113821/113821 [00:00<00:00, 1449116.61i
t/s]
Epoch 4/5: 100%|██████████| 890/890 [00:06<00:00, 131.31it/s, loss=1.3023]
INFO:__main__:
Evaluating epoch 4...
INFO:__main__:
Epoch 4 Summary:
INFO:__main__:Average Loss: 2.0123
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: 0.0340
INFO:__main__:MSE: 0.2671
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w2_e100_glove_glove.pt
Creating training pairs: 100%|██████████| 113821/113821 [00:00<00:00, 1630085.48i
t/s]
Epoch 5/5: 100%|██████████| 890/890 [00:07<00:00, 124.40it/s, loss=1.2750]
INFO:__main__:
Evaluating epoch 5...
INFO:__main__:
Epoch 5 Summary:
INFO:__main__:Average Loss: 1.5731
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: 0.0380
INFO:__main__:MSE: 0.2666
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w2_e100_glove_glove.pt
INFO:__main__:
==================== Training Complete ====================
INFO:__main__:Total training time: 40.77s
INFO:__main__:Best loss achieved: 1.5731
INFO:__main__:
Training GloVe with config: {'window_size': 5, 'embedding_size': 100, 'x_max': 10
0, 'alpha': 0.75, 'batch_size': 128, 'epochs': 5}
INFO:__main__:
==================== Training Configuration ====================
INFO:__main__:Window Size: 5
INFO:__main__:Embedding Size: 100
INFO:__main__:X_max: 100
INFO:__main__:Alpha: 0.75
INFO:__main__:Batch Size: 128
INFO:__main__:Epochs: 5

INFO:__main__:Preparing training data...
INFO:__main__:Vocabulary size: 2560 words
INFO:__main__:Building co-occurrence matrix...
Processing sentences: 100%|██████████| 4623/4623 [00:00<00:00, 8437.94it/s]
INFO:__main__:Created co-occurrence matrix with 221619 non-zero entries
INFO:__main__:Model parameters: 517,120
INFO:__main__:Loading evaluation datasets...
INFO:__main__:Successfully loaded 203 word pairs from WordSim-353
INFO:__main__:Loaded 12 semantic pairs and 15 syntactic pairs
INFO:__main__:
```

```
==================== Starting Training ====================
Creating training pairs: 100%|██████████| 221619/221619 [00:00<00:00, 1568283.74i
t/s]
Epoch 1/5: 100%|██████████| 1732/1732 [00:13<00:00, 129.07it/s, loss=3.9722]
INFO:__main__:
Evaluating epoch 1...
INFO:__main__:
Epoch 1 Summary:
INFO:__main__:Average Loss: 3.2205
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: 0.1041
INFO:__main__:MSE: 0.2877
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w5_e100_glove_glove.pt
Creating training pairs: 100%|██████████| 221619/221619 [00:00<00:00, 1518348.42i
t/s]
Epoch 2/5: 100%|██████████| 1732/1732 [00:13<00:00, 130.40it/s, loss=2.1104]
INFO:__main__:
Evaluating epoch 2...
INFO:__main__:
Epoch 2 Summary:
INFO:__main__:Average Loss: 2.2967
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: 0.0873
INFO:__main__:MSE: 0.2891
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w5_e100_glove_glove.pt
Creating training pairs: 100%|██████████| 221619/221619 [00:00<00:00, 1342988.58i
t/s]
Epoch 3/5: 100%|██████████| 1732/1732 [00:13<00:00, 125.95it/s, loss=2.4845]
INFO:__main__:
Evaluating epoch 3...
INFO:__main__:
Epoch 3 Summary:
INFO:__main__:Average Loss: 1.6509
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: 0.0643
INFO:__main__:MSE: 0.2907
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w5_e100_glove_glove.pt
Creating training pairs: 100%|██████████| 221619/221619 [00:00<00:00, 1468129.62i
t/s]
Epoch 4/5: 100%|██████████| 1732/1732 [00:13<00:00, 125.34it/s, loss=0.9231]
INFO:__main__:
Evaluating epoch 4...
INFO:__main__:
Epoch 4 Summary:
INFO:__main__:Average Loss: 1.1849
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: 0.0614
INFO:__main__:MSE: 0.2919
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w5_e100_glove_glove.pt
Creating training pairs: 100%|██████████| 221619/221619 [00:00<00:00, 1626527.09i
t/s]
Epoch 5/5: 100%|██████████| 1732/1732 [00:14<00:00, 121.58it/s, loss=0.8132]
```

```
INFO:__main__:
Evaluating epoch 5...
INFO:__main__:
Epoch 5 Summary:
INFO:__main__:Average Loss: 0.8460
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: 0.0458
INFO:__main__:MSE: 0.2929
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w5_e100_glove_glove.pt
INFO:__main__:
==================== Training Complete ====================
INFO:__main__:Total training time: 74.49s
INFO:__main__:Best loss achieved: 0.8460
INFO:__main__:
Training GloVe with config: {'window_size': 10, 'embedding_size': 100, 'x_max': 1
00, 'alpha': 0.75, 'batch_size': 128, 'epochs': 5}
INFO:__main__:
==================== Training Configuration ====================
INFO:__main__:Window Size: 10
INFO:__main__:Embedding Size: 100
INFO:__main__:X_max: 100
INFO:__main__:Alpha: 0.75
INFO:__main__:Batch Size: 128
INFO:__main__:Epochs: 5

INFO:__main__:Preparing training data...
INFO:__main__:Vocabulary size: 2560 words
INFO:__main__:Building co-occurrence matrix...
Processing sentences: 100%|██████████| 4623/4623 [00:00<00:00, 4875.92it/s]
INFO:__main__:Created co-occurrence matrix with 333183 non-zero entries
INFO:__main__:Model parameters: 517,120
INFO:__main__:Loading evaluation datasets...
INFO:__main__:Successfully loaded 203 word pairs from WordSim-353
INFO:__main__:Loaded 12 semantic pairs and 15 syntactic pairs
INFO:__main__:
==================== Starting Training ====================
Creating training pairs: 100%|██████████| 333183/333183 [00:00<00:00, 1629485.01i
t/s]
Epoch 1/5: 100%|██████████| 2603/2603 [00:20<00:00, 130.02it/s, loss=2.6360]
INFO:__main__:
Evaluating epoch 1...
INFO:__main__:
Epoch 1 Summary:
INFO:__main__:Average Loss: 2.4781
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: -0.2737
INFO:__main__:MSE: 0.2778
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w10_e100_glove_glove.pt
Creating training pairs: 100%|██████████| 333183/333183 [00:00<00:00, 1377987.43i
t/s]
Epoch 2/5: 100%|██████████| 2603/2603 [00:20<00:00, 128.88it/s, loss=1.2787]
INFO:__main__:
Evaluating epoch 2...
INFO:__main__:
Epoch 2 Summary:
INFO:__main__:Average Loss: 1.6470
```

```
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: -0.2552
INFO:__main__:MSE: 0.2779
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w10_e100_glove_glove.pt
Creating training pairs: 100%|██████████| 333183/333183 [00:00<00:00, 1488330.38i
t/s]
Epoch 3/5: 100%|██████████| 2603/2603 [00:20<00:00, 127.64it/s, loss=0.7111]
INFO:__main__:
Evaluating epoch 3...
INFO:__main__:
Epoch 3 Summary:
INFO:__main__:Average Loss: 1.0999
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: -0.2539
INFO:__main__:MSE: 0.2782
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w10_e100_glove_glove.pt
Creating training pairs: 100%|██████████| 333183/333183 [00:00<00:00, 1468347.94i
t/s]
Epoch 4/5: 100%|██████████| 2603/2603 [00:20<00:00, 129.15it/s, loss=0.7844]
INFO:__main__:
Evaluating epoch 4...
INFO:__main__:
Epoch 4 Summary:
INFO:__main__:Average Loss: 0.7289
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: -0.2519
INFO:__main__:MSE: 0.2786
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w10_e100_glove_glove.pt
Creating training pairs: 100%|██████████| 333183/333183 [00:00<00:00, 1613433.74i
t/s]
Epoch 5/5: 100%|██████████| 2603/2603 [00:19<00:00, 131.31it/s, loss=0.3477]
INFO:__main__:
Evaluating epoch 5...
INFO:__main__:
Epoch 5 Summary:
INFO:__main__:Average Loss: 0.4783
INFO:__main__:Semantic Accuracy: 0.0000
INFO:__main__:Syntactic Accuracy: 0.0000
INFO:__main__:Similarity Correlation: -0.2549
INFO:__main__:MSE: 0.2792
INFO:__main__:New best model! Saving checkpoint...
INFO:__main__:Model saved to saved_models/w10_e100_glove_glove.pt
INFO:__main__:
==================== Training Complete ====================
INFO:__main__:Total training time: 107.17s
INFO:__main__:Best loss achieved: 0.4783
INFO:__main__:
Training Metrics:
INFO:__main__:
Similarity Metrics:
```

```
Training and Accuracy Results:
--------------------------------------------------------------------------------
----------------------------
Model                 | Window Size      | Training Loss    | Training Time    | Synt
actic Acc      | Semantic Acc
--------------------------------------------------------------------------------
----------------------------
GloVe (w=2, α=0.75)  | 2                | 1.5731           | 40.77s           | 0.00
00             | 0.0000
GloVe (w=5, α=0.75)  | 5                | 0.8460           | 74.49s           | 0.00
00             | 0.0000
GloVe (w=10, α=0.75) | 10               | 0.4783           | 107.17s          | 0.00
00             | 0.0000
--------------------------------------------------------------------------------
----------------------------

Similarity Comparison Results:
---------------------------------------------------
Metric           | GloVe            | Y true
---------------------------------------------------
MSE              | 0.2666           | 1.0000
---------------------------------------------------
```

GloVe Gensim

In [5]:

```
import numpy as np
import torch
import torch.nn as nn
import logging
import os
import time
from tqdm import tqdm

# Setup logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    datefmt='%H:%M:%S'
)
logger = logging.getLogger(__name__)

class PretrainedGloVe(nn.Module):
    """Direct wrapper for pretrained GloVe embeddings"""

    def __init__(self, embeddings, word2idx):
        super().__init__()
        self.word2idx = word2idx
        self.idx2word = {i: word for word, i in word2idx.items()}
        self.embedding_size = embeddings.shape[1]

        # Create embedding layer from pretrained vectors
        self.embedding = nn.Embedding.from_pretrained(torch.FloatTensor(embeddin

    def embedding_center(self, indices):
        """Match the interface of our other models"""
        return self.embedding(indices)

    def forward(self, x):
```

```
        return self.embedding(x)

def load_pretrained_glove(path, dim=100):
    """Load pretrained GloVe embeddings directly

    Args:
        path: Path to GloVe embeddings file
        dim: Embedding dimension

    Returns:
        PretrainedGloVe: Model with pretrained embeddings
    """
    logger.info(f"\n{'='*20} Loading Configuration {'='*20}")
    logger.info(f"Model Path: {path}")
    logger.info(f"Embedding Dimension: {dim}\n")

    # Load GloVe vectors
    logger.info("Loading pretrained embeddings...")
    word2idx = {}
    vectors = []

    # First pass: collect words and create word2idx
    with open(path, 'r', encoding='utf-8') as f:
        for i, line in enumerate(tqdm(f, desc="Building vocabulary")):
            tokens = line.rstrip().split(' ')
            word = tokens[0]
            word2idx[word] = i

    # Initialize embedding matrix
    embeddings = np.zeros((len(word2idx), dim))

    # Second pass: fill embedding matrix
    with open(path, 'r', encoding='utf-8') as f:
        for line in tqdm(f, desc="Loading embeddings"):
            tokens = line.rstrip().split(' ')
            word = tokens[0]
            vector = np.array([float(x) for x in tokens[1:]], dtype=np.float32)
            embeddings[word2idx[word]] = vector

    logger.info(f"Loaded {len(word2idx):,} words with dimension {dim}")

    # Create model
    model = PretrainedGloVe(embeddings, word2idx)

    # Load evaluation datasets
    logger.info("\nLoading evaluation datasets...")
    semantic_pairs, syntactic_pairs = load_word_analogies()
    similarities = load_similarity_dataset()
    logger.info(f"Loaded {len(semantic_pairs)} semantic pairs and {len(syntactic

    # Evaluate model
    logger.info("\n" + "="*50)
    logger.info("Starting Model Evaluation")
    logger.info("="*50)

    # Semantic analogies evaluation
    logger.info("\nEvaluating semantic analogies...")
    semantic_acc = evaluate_analogies(model, word2idx, model.idx2word, semantic_
    logger.info(f"Number of semantic pairs evaluated: {len(semantic_pairs)}")
    logger.info(f"Semantic accuracy: {semantic_acc:.4f}")
```

```
    # Syntactic analogies evaluation
    logger.info("\nEvaluating syntactic analogies...")
    syntactic_acc = evaluate_analogies(model, word2idx, model.idx2word, syntacti
    logger.info(f"Number of syntactic pairs evaluated: {len(syntactic_pairs)}")
    logger.info(f"Syntactic accuracy: {syntactic_acc:.4f}")

    # Word similarity evaluation
    logger.info("\nEvaluating word similarities...")
    similarity_corr, mse, num_pairs = evaluate_similarity(model, word2idx, simil
    logger.info(f"Number of similarity pairs evaluated: {num_pairs}")
    logger.info(f"Spearman correlation: {similarity_corr:.4f}")
    logger.info(f"Mean squared error: {mse:.4f}")

    # Example analogies
    logger.info("\nExample analogies evaluation:")
    example_analogies = [
        ('king', 'man', 'queen', 'woman'),
        ('paris', 'france', 'rome', 'italy'),
        ('good', 'better', 'bad', 'worse'),
        ('small', 'smaller', 'large', 'larger')
    ]

    for a, b, c, d in example_analogies:
        if all(word in word2idx for word in [a, b, c, d]):
            # Get embeddings
            va = model.embedding(torch.tensor(word2idx[a]))
            vb = model.embedding(torch.tensor(word2idx[b]))
            vc = model.embedding(torch.tensor(word2idx[c]))
            vd = model.embedding(torch.tensor(word2idx[d]))

            # Calculate cosine similarity between analogy pairs
            cos = nn.CosineSimilarity(dim=0)
            similarity = cos(vb - va, vd - vc)
            logger.info(f"Analogy {a}:{b} :: {c}:{d} - Similarity: {similarity:.

    # Example similarities
    logger.info("\nExample word similarities:")
    example_pairs = [
        ('man', 'woman'),
        ('king', 'queen'),
        ('computer', 'machine'),
        ('happy', 'sad')
    ]

    for word1, word2 in example_pairs:
        if word1 in word2idx and word2 in word2idx:
            # Get embeddings
            v1 = model.embedding(torch.tensor(word2idx[word1]))
            v2 = model.embedding(torch.tensor(word2idx[word2]))

            # Calculate cosine similarity
            cos = nn.CosineSimilarity(dim=0)
            similarity = cos(v1, v2)
            logger.info(f"Similarity between '{word1}' and '{word2}': {similarit

    # Print evaluation summary
    logger.info(f"\n{'='*20} Evaluation Summary {'='*20}")
    logger.info(f"Semantic Accuracy: {semantic_acc:.4f} ({len(semantic_pairs)} p
    logger.info(f"Syntactic Accuracy: {syntactic_acc:.4f} ({len(syntactic_pairs)
```

```python
    logger.info(f"Similarity Correlation: {similarity_corr:.4f} ({num_pairs} pai
    logger.info(f"Mean Squared Error: {mse:.4f}")
    logger.info("="*50)

    # Save model in our format
    model_dir = "saved_models"
    os.makedirs(model_dir, exist_ok=True)
    model_path = os.path.join(model_dir, f"glove_pretrained_d{dim}.pt")
    save_model(model, word2idx, model.idx2word, model_path, model_type="glove_pr
    logger.info(f"\nModel saved to {model_path}")

    return model, {
        'word2idx': word2idx,
        'idx2word': model.idx2word,
        'semantic_accuracy': semantic_acc,
        'syntactic_accuracy': syntactic_acc,
        'similarity_correlation': similarity_corr,
        'mse': mse,
        'num_pairs': num_pairs,
        'model_path': model_path,
        'vocab_size': len(word2idx),
        'embedding_size': dim
    }

if __name__ == "__main__":
    # Initialize evaluator
    evaluator = ModelEvaluator()

    # Configurations for pretrained models
    configs = [
        {
            'path': 'glove.6B.100d.txt',
            'dim': 100
        },
        {
            'path': 'glove.6B.300d.txt',
            'dim': 300
        }
    ]

    # Load and evaluate models
    for config in configs:
        logger.info(f"\nLoading GloVe with config: {config}")
        try:
            start_time = time.time()
            model, results = load_pretrained_glove(**config)
            loading_time = time.time() - start_time

            model_name = f"GloVe-Pretrained (d={config['dim']})"
            evaluator.evaluate_model(
                model,
                results['word2idx'],
                results['idx2word'],
                model_name,
                window_size=None,  # N/A for pretrained models
                training_time=loading_time,  # Use loading time instead
                final_loss=None,  # N/A for pretrained models
                semantic_accuracy=results['semantic_accuracy'],
                syntactic_accuracy=results['syntactic_accuracy'],
                similarity_correlation=results['similarity_correlation'],
```

```
                mse=results['mse']
            )

            logger.info(f"\nModel Statistics:")
            logger.info(f"Vocabulary Size: {results['vocab_size']:,}")
            logger.info(f"Embedding Size: {results['embedding_size']}")
            logger.info(f"Loading Time: {loading_time:.2f}s")

        except FileNotFoundError:
            logger.error(f"Pretrained embeddings not found at {config['path']}")
            logger.error("Please download the embeddings from https://nlp.stanfo
            logger.error("and extract them to the 'pretrained' directory")
            continue
        except Exception as e:
            logger.error(f"Error loading model: {str(e)}")
            continue

    # Print evaluation results
    logger.info("\nTraining Metrics:")
    evaluator.print_training_table()

    logger.info("\nSimilarity Metrics:")
    evaluator.print_similarity_table()
```

```
09:40:14 - INFO - Successfully loaded 203 word pairs from WordSim-353
09:40:14 - INFO -
Loading GloVe with config: {'path': 'glove.6B.100d.txt', 'dim': 100}
09:40:14 - INFO -
==================== Loading Configuration ====================
09:40:14 - INFO - Model Path: glove.6B.100d.txt
09:40:14 - INFO - Embedding Dimension: 100

09:40:14 - INFO - Loading pretrained embeddings...
Building vocabulary: 400000it [00:02, 165074.48it/s]
Loading embeddings: 400000it [00:10, 38129.74it/s]
09:40:27 - INFO - Loaded 400,000 words with dimension 100
09:40:27 - INFO -
Loading evaluation datasets...
09:40:27 - INFO - Successfully loaded 203 word pairs from WordSim-353
09:40:27 - INFO - Loaded 12 semantic pairs and 15 syntactic pairs
09:40:27 - INFO -
==================================================
09:40:27 - INFO - Starting Model Evaluation
09:40:27 - INFO - ==================================================
09:40:27 - INFO -
Evaluating semantic analogies...
09:45:48 - INFO - Number of semantic pairs evaluated: 12
09:45:48 - INFO - Semantic accuracy: 0.9167
09:45:48 - INFO -
Evaluating syntactic analogies...
09:52:32 - INFO - Number of syntactic pairs evaluated: 15
09:52:32 - INFO - Syntactic accuracy: 0.5333
09:52:32 - INFO -
Evaluating word similarities...
09:52:32 - INFO - Number of similarity pairs evaluated: 203
09:52:32 - INFO - Spearman correlation: 0.6035
09:52:32 - INFO - Mean squared error: 0.0502
09:52:32 - INFO -
Example analogies evaluation:
09:52:32 - INFO - Analogy king:man :: queen:woman - Similarity: 0.7581
09:52:32 - INFO - Analogy paris:france :: rome:italy - Similarity: 0.7056
09:52:32 - INFO - Analogy good:better :: bad:worse - Similarity: 0.5263
09:52:32 - INFO - Analogy small:smaller :: large:larger - Similarity: 0.6943
09:52:32 - INFO -
Example word similarities:
09:52:32 - INFO - Similarity between 'man' and 'woman': 0.8323
09:52:32 - INFO - Similarity between 'king' and 'queen': 0.7508
09:52:32 - INFO - Similarity between 'computer' and 'machine': 0.5942
09:52:32 - INFO - Similarity between 'happy' and 'sad': 0.6801
09:52:32 - INFO -
==================== Evaluation Summary ====================
09:52:32 - INFO - Semantic Accuracy: 0.9167 (12 pairs)
09:52:32 - INFO - Syntactic Accuracy: 0.5333 (15 pairs)
09:52:32 - INFO - Similarity Correlation: 0.6035 (203 pairs)
09:52:32 - INFO - Mean Squared Error: 0.0502
09:52:32 - INFO - ==================================================
09:52:32 - ERROR - Error loading model: 'function' object has no attribute 'embed
ding_dim'
09:52:32 - INFO -
Loading GloVe with config: {'path': 'glove.6B.300d.txt', 'dim': 300}
09:52:32 - INFO -
==================== Loading Configuration ====================
09:52:32 - INFO - Model Path: glove.6B.300d.txt
09:52:32 - INFO - Embedding Dimension: 300
```

```
09:52:32 - INFO - Loading pretrained embeddings...
Building vocabulary: 400000it [00:06, 64095.88it/s]
Loading embeddings: 400000it [00:31, 12855.92it/s]
09:53:09 - INFO - Loaded 400,000 words with dimension 300
09:53:09 - INFO -
Loading evaluation datasets...
09:53:09 - INFO - Successfully loaded 203 word pairs from WordSim-353
09:53:09 - INFO - Loaded 12 semantic pairs and 15 syntactic pairs
09:53:09 - INFO -
==================================================
09:53:09 - INFO - Starting Model Evaluation
09:53:09 - INFO - ==================================================
09:53:09 - INFO -
Evaluating semantic analogies...
09:58:42 - INFO - Number of semantic pairs evaluated: 12
09:58:42 - INFO - Semantic accuracy: 0.7500
09:58:42 - INFO -
Evaluating syntactic analogies...
10:05:35 - INFO - Number of syntactic pairs evaluated: 15
10:05:35 - INFO - Syntactic accuracy: 0.4000
10:05:35 - INFO -
Evaluating word similarities...
10:05:35 - INFO - Number of similarity pairs evaluated: 203
10:05:35 - INFO - Spearman correlation: 0.6638
10:05:35 - INFO - Mean squared error: 0.0750
10:05:35 - INFO -
Example analogies evaluation:
10:05:35 - INFO - Analogy king:man :: queen:woman - Similarity: 0.6814
10:05:35 - INFO - Analogy paris:france :: rome:italy - Similarity: 0.6228
10:05:35 - INFO - Analogy good:better :: bad:worse - Similarity: 0.5290
10:05:35 - INFO - Analogy small:smaller :: large:larger - Similarity: 0.6370
10:05:35 - INFO -
Example word similarities:
10:05:35 - INFO - Similarity between 'man' and 'woman': 0.6999
10:05:35 - INFO - Similarity between 'king' and 'queen': 0.6336
10:05:35 - INFO - Similarity between 'computer' and 'machine': 0.4563
10:05:35 - INFO - Similarity between 'happy' and 'sad': 0.5653
10:05:35 - INFO -
==================== Evaluation Summary ====================
10:05:35 - INFO - Semantic Accuracy: 0.7500 (12 pairs)
10:05:35 - INFO - Syntactic Accuracy: 0.4000 (15 pairs)
10:05:35 - INFO - Similarity Correlation: 0.6638 (203 pairs)
10:05:35 - INFO - Mean Squared Error: 0.0750
10:05:35 - INFO - ==================================================
10:05:35 - ERROR - Error loading model: 'function' object has no attribute 'embed
ding_dim'
10:05:35 - INFO -
Training Metrics:
10:05:35 - INFO -
Similarity Metrics:
```

```
Training and Accuracy Results:
---------------------------------------------------------------------------------------------------
Model            | Window Size     | Training Loss    | Training Time    | Syntactic Acc    | Semantic Acc
---------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------

Similarity Comparison Results:
---------------------------------
Metric           | Y true
---------------------------------
MSE              | 1.0000
---------------------------------
```

💡 Even though the table values were not printed, the results can be retrieved from the training logs(the results have indeed being logged, so nothing to look here or panic!)

Task 2: Model Comparison and Analysis

1. Compare Skip-gram, Skip-gram negative sampling, GloVe models on training loss, training time. (1 points)✅
2. Use Word analogies dataset to calucalte between syntactic and semantic accuracy, similar to the methods in the Word2Vec and GloVe paper. (1 points)✅

- Note : using only capital-common-countries for semantic and past-tense for syntactic.
- Note : Do not be surprised if you achieve 0% accuracy in these experiments, as this may be due to the limitations of our corpus. If you are curious, you can try the same experiments with a pre-trained GloVe model from the Gensim library for a comparison.

Here's the comparison table: (I even tried experimenting with larger window size since I was getting 0 for Window Size = 2)

Model	Window Size	Training Loss	Training Time	Syntactic Accuracy	Semantic Accuracy
Skipgram	2	6.4928	1633.94s	0	0
Skipgram	5	5.6520	3814.83s	0	0
Skipgram (NEG)	2	2.7663	155.74s	0	0
Skipgram (NEG)	5	2.4614	374.31s	0	0
Glove	2	1.5731	40.77s	0	0
Glove	5	0.8460	74.49s	0	0
Glove	10	0.4783	107.17s	0	0

Model	Window Size	Training Loss	Training Time	Syntactic Accuracy	Semantic Accuracy
Glove (Gensim) 6B 100 Dim	2	-	-	0.5333	0.9167
Glove (Gensim) 6B 300 Dim	2	-	-	0.4000	0.7500

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. (1 points) ✅

Here's the comparison table:

Model	Skipgram	NEG	GloVe	GloVe (gensim) 100 Dim	GloVe (gensim) 300 Dim	Y_True
MSE	0.2474	0.1879	0.2666	0.0750	0.0502	1.0000

Key Observations & Analysis

In terms of **Training Efficiency**:

- GloVe demonstrated superior training efficiency with the fastest training times (40-107s) compared to Skip-gram (1633-3814s) and Skip-gram with Negative Sampling (155-374s)
- GloVe achieved the lowest training losses (0.4783-1.5731) across all window sizes, showing better convergence than both Skip-gram variants

In terms of **Window Size Impact**

- Larger window sizes generally improved model performance:
  - GloVe's loss decreased from 1.5731 (window=2) to 0.4783 (window=10)
  - Skip-gram's loss decreased from 6.4928 (window=2) to 5.6520 (window=5)
  - Skip-gram with Negative Sampling's loss decreased from 2.7663 (window=2) to 2.4614 (window=5)

In terms of **Accuracy Metrics**

- Custom-trained models showed poor performance on semantic and syntactic accuracy (all 0%)
- Pre-trained GloVe models performed significantly better:
  - 100D model: 91.67% semantic accuracy, 53.33% syntactic accuracy
  - 300D model: 75% semantic accuracy, 40% syntactic accuracy

In terms of **Mean Squared Error Analysis**

- Skip-gram with Negative Sampling achieved the best MSE (0.1879) among custom-trained models

- Pre-trained GloVe models significantly outperformed custom models:
  - 100D: 0.0502 MSE
  - 300D: 0.0750 MSE

Key Findings

1. GloVe's algorithm demonstrates superior computational efficiency while achieving better convergence
2. Skip-gram with Negative Sampling shows better performance than basic Skip-gram, suggesting the effectiveness of negative sampling in improving training
3. The significant performance gap between custom-trained and pre-trained models highlights the importance of large-scale training data and proper hyperparameter tuning
4. The 100D pre-trained GloVe model surprisingly outperformed the 300D model, suggesting that higher dimensionality doesn't always guarantee better performance

Similar 10 Words

In [38]:

```
import torch
import logging
from pathlib import Path
from tabulate import tabulate

# Setup logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    datefmt='%H:%M:%S'
)
logger = logging.getLogger(__name__)

def load_models():
    """Load all available models"""
    models = {}
    base_path = "/home/jupyter-st125462/NLP/A1/saved_models"

    # Model configurations
    model_configs = {
        'skipgram': {
            'class': Skipgram,
            'path': f"{base_path}/w2_e100_skipgram.pt",
            'name': "Skip-gram"
        },
        'skipgram_neg': {
            'class': SkipgramNeg,
            'path': f"{base_path}/w2_e100_skipgram_neg_skipgram_neg.pt",
            'name': "Skip-gram (Neg)"
        },
        'glove': {
            'class': GloVe,
```

```
            'path': f"{base_path}/w2_e100_glove_glove.pt",
            'name': "GloVe"
        }
    }

    # Try loading each model
    for model_type, config in model_configs.items():
        try:
            logger.info(f"Loading {config['name']} from: {config['path']}")
            model, word2idx, idx2word = load_model(config['class'], config['path
            model.eval()
            models[model_type] = {
                'model': model,
                'word2idx': word2idx,
                'idx2word': idx2word,
                'name': config['name']
            }
            logger.info(f"Successfully loaded {config['name']}")
        except Exception as e:
            logger.warning(f"Could not load {config['name']}: {str(e)}")

    return models

def display_similar_words_comparison(query_words, models, top_k=10):
    """Display similar words comparison across all models"""
    for query in query_words:
        print(f"\nSimilar words to '{query}':")
        print("=" * 80)

        # Collect results from all models
        all_results = []
        headers = ["Rank"]

        # Add model names to headers
        for model_info in models.values():
            headers.append(f"{model_info['name']}")
            headers.append("Sim")

        # Get similar words from each model
        max_rows = 0
        model_results = {}

        for model_type, model_info in models.items():
            if query not in model_info['word2idx']:
                logger.warning(f"Word '{query}' not in vocabulary for {model_inf
                continue

            try:
                similar = find_similar_words(
                    query,
                    model_info['model'],
                    model_info['word2idx'],
                    model_info['idx2word'],
                    top_k
                )
                model_results[model_type] = [
                    (word, sim) for word, sim in similar if word != query
                ]
                max_rows = max(max_rows, len(model_results[model_type]))
            except Exception as e:
```

```
                logger.error(f"Error finding similar words for {model_info['name
                continue

        # Create table rows
        table_data = []
        for i in range(max_rows):
            row = [f"{i+1}"]
            for model_type, model_info in models.items():
                if model_type in model_results and i < len(model_results[model_t
                    word, sim = model_results[model_type][i]
                    row.extend([word, f"{sim:.4f}"])
                else:
                    row.extend(["", ""])
            table_data.append(row)

        if table_data:
            print(tabulate(table_data, headers=headers, tablefmt="grid"))
        else:
            print(f"No results found for '{query}'")
        print()

def main():
    # Load all available models
    models = load_models()

    if not models:
        logger.error("No models could be loaded. Please check model paths.")
        return

    # Query words to test
    query_words = [
        # Common words
        "good", "day", "time", "person", "world", "work",
        "news", "sad", "lion", "donkey",
        "man", "woman", "learning", "language"
    ]

    # Display similar words comparison
    display_similar_words_comparison(query_words, models)

if __name__ == "__main__":
    main()
```

```
INFO:__main__:Loading Skip-gram from: /home/jupyter-st125462/NLP/A1/saved_models/
w2_e100_skipgram.pt
/tmp/ipykernel_762838/3318003762.py:379: FutureWarning: You are using `torch.load
` with `weights_only=False` (the current default value), which uses the default p
ickle module implicitly. It is possible to construct malicious pickle data which 
will execute arbitrary code during unpickling (See https://github.com/pytorch/pyt
orch/blob/main/SECURITY.md#untrusted-models for more details). In a future releas
e, the default value for `weights_only` will be flipped to `True`. This limits th
e functions that could be executed during unpickling. Arbitrary objects will no l
onger be allowed to be loaded via this mode unless they are explicitly allowliste
d by the user via `torch.serialization.add_safe_globals`. We recommend you start 
setting `weights_only=True` for any use case where you don't have full control of
the loaded file. Please open an issue on GitHub for any issues related to this ex
perimental feature.
  checkpoint = torch.load(model_path)
INFO:__main__:Successfully loaded Skip-gram
INFO:__main__:Loading Skip-gram (Neg) from: /home/jupyter-st125462/NLP/A1/saved_m
odels/w2_e100_skipgram_neg_skipgram_neg.pt
INFO:__main__:Successfully loaded Skip-gram (Neg)
INFO:__main__:Loading GloVe from: /home/jupyter-st125462/NLP/A1/saved_models/w2_e
100_glove_glove.pt
INFO:__main__:Successfully loaded GloVe
WARNING:__main__:Word 'king' not in vocabulary for Skip-gram
WARNING:__main__:Word 'king' not in vocabulary for Skip-gram (Neg)
WARNING:__main__:Word 'king' not in vocabulary for GloVe
WARNING:__main__:Word 'computer' not in vocabulary for Skip-gram
WARNING:__main__:Word 'computer' not in vocabulary for Skip-gram (Neg)
WARNING:__main__:Word 'computer' not in vocabulary for GloVe
```

```
Similar words to 'king':
================================================================================
No results found for 'king'


Similar words to 'computer':
================================================================================
No results found for 'computer'


Similar words to 'good':
================================================================================
+--------+-------------+--------+------------------+--------+--------------+----
----+
|   Rank | Skip-gram   |    Sim | Skip-gram (Neg)  |    Sim | GloVe        |
Sim |
+========+=============+========+==================+========+==============+====
====+
|      1 | brevard     | 0.3579 | size             | 0.4519 | pitchers     | 0.3
564 |
+--------+-------------+--------+------------------+--------+--------------+----
----+
|      2 | nomination  | 0.3514 | give             | 0.4355 | four         | 0.3
358 |
+--------+-------------+--------+------------------+--------+--------------+----
----+
|      3 | setting     | 0.3222 | property         | 0.4291 | manufacturer | 0.3
077 |
+--------+-------------+--------+------------------+--------+--------------+----
----+
|      4 | important   | 0.3195 | parker           | 0.4269 | sue          | 0.2
992 |
+--------+-------------+--------+------------------+--------+--------------+----
----+
|      5 | c.          | 0.3181 | 1953             | 0.4258 | italian      | 0.2
947 |
+--------+-------------+--------+------------------+--------+--------------+----
----+
|      6 | group       | 0.3123 | women's          | 0.4098 | present      | 0.2
939 |
+--------+-------------+--------+------------------+--------+--------------+----
----+
|      7 | increased   | 0.3071 | the              | 0.393  | eisenhower   | 0.2
796 |
+--------+-------------+--------+------------------+--------+--------------+----
----+
|      8 | table       | 0.3063 | arms             | 0.3891 | hot          | 0.2
754 |
+--------+-------------+--------+------------------+--------+--------------+----
----+
|      9 | clark       | 0.3012 | 15               | 0.3887 | corp.        | 0.2
728 |
+--------+-------------+--------+------------------+--------+--------------+----
----+


Similar words to 'day':
================================================================================
+--------+-------------+--------+------------------+--------+------------+-------
-+
```

Rank	Skip-gram	Sim	Skip-gram (Neg)	Sim	GloVe	Sim
1	christmas	0.3743	reason	0.404	hawksley	0.4003
2	problems	0.3686	hill	0.3953	although	0.3108
3	calls	0.348	harry	0.3843	want	0.3097
4	address	0.3348	15	0.3777	reading	0.2942
5	order	0.3309	city's	0.3717	schedule	0.2757
6	immediate	0.3283	younger	0.3668	efforts	0.2709
7	p.m.	0.3258	an	0.3653	treatment	0.2697
8	opportunity	0.3223	johnston	0.3611	learned	0.2649
9	informed	0.3212	additional	0.3603	april	0.2623

Similar words to 'time':

==

Rank	Skip-gram	Sim	Skip-gram (Neg)	Sim	GloVe	Sim
1	not	0.5119	audience	0.4409	higher	0.3364
2	victory	0.398	great	0.4259	competition	0.2999
3	will	0.3484	responsibility	0.4087	offer	0.288

4	about	0.3436	<UNK>	0.404	treatment	0.2824
5	large	0.3395	mr.	0.4013	lake	0.2786
6	homes	0.3289	doesn't	0.3965	la	0.2726
7	raising	0.3279	that	0.3945	neutral	0.2718
8	proposed	0.3273	all	0.3937	stopped	0.2713
9	previous	0.3266	thought	0.3856	added	0.2696

Similar words to 'person':
==

Rank	Skip-gram	Sim	Skip-gram (Neg)	Sim	GloVe	Sim
1	1959	0.4341	did	0.4501	individuals	0.3723
2	comes	0.3612	baltimore	0.441	supreme	0.3623
3	nine	0.3565	indeed	0.3679	mills	0.3576
4	produced	0.3518	trouble	0.3671	library	0.332
5	cost	0.3374	pirates	0.3642	1949	0.3107
6	personnel	0.3316	belgians	0.3469	first	0.2889

```
|       7 | over         | 0.3164 | reply              | 0.3351 | mayor       | 0.28
43 |
+--------+-------------+--------+------------------+--------+-----------+-----
---+
|       8 | board        | 0.3083 | agreed             | 0.329  | half        | 0.28
01 |
+--------+-------------+--------+------------------+--------+-----------+-----
---+
|       9 | only         | 0.308  | a.m.               | 0.3223 | paso        | 0.27
85 |
+--------+-------------+--------+------------------+--------+-----------+-----
---+


Similar words to 'world':
================================================================================
+--------+-------------+--------+--------------------+--------+---------+--------+
|   Rank | Skip-gram   |    Sim | Skip-gram (Neg)    |    Sim | GloVe   |    Sim |
+========+=============+========+====================+========+=========+========+
|      1 | opinion     | 0.4198 | stage              | 0.4094 | ap      | 0.3047 |
+--------+-------------+--------+--------------------+--------+---------+--------+
|      2 | marr        | 0.372  | they're            | 0.4077 | mary    | 0.3027 |
+--------+-------------+--------+--------------------+--------+---------+--------+
|      3 | he          | 0.3483 | final              | 0.4073 | rayburn | 0.2853 |
+--------+-------------+--------+--------------------+--------+---------+--------+
|      4 | long        | 0.3454 | military           | 0.3936 | holmes  | 0.2788 |
+--------+-------------+--------+--------------------+--------+---------+--------+
|      5 | entering    | 0.3378 | car                | 0.3927 | walter  | 0.2762 |
+--------+-------------+--------+--------------------+--------+---------+--------+
|      6 | scene       | 0.3341 | order              | 0.3844 | coast   | 0.2678 |
+--------+-------------+--------+--------------------+--------+---------+--------+
|      7 | last        | 0.3122 | better             | 0.3832 | shea    | 0.2648 |
+--------+-------------+--------+--------------------+--------+---------+--------+
|      8 | very        | 0.3119 | giants             | 0.3798 | vice    | 0.2632 |
+--------+-------------+--------+--------------------+--------+---------+--------+
|      9 | i           | 0.3063 | all                | 0.3774 | back    | 0.253  |
+--------+-------------+--------+--------------------+--------+---------+--------+


Similar words to 'work':
================================================================================
```

```
WARNING:__main__:Word 'data' not in vocabulary for Skip-gram
WARNING:__main__:Word 'data' not in vocabulary for Skip-gram (Neg)
WARNING:__main__:Word 'data' not in vocabulary for GloVe
WARNING:__main__:Word 'algorithm' not in vocabulary for Skip-gram
WARNING:__main__:Word 'algorithm' not in vocabulary for Skip-gram (Neg)
WARNING:__main__:Word 'algorithm' not in vocabulary for GloVe
WARNING:__main__:Word 'network' not in vocabulary for Skip-gram
WARNING:__main__:Word 'network' not in vocabulary for Skip-gram (Neg)
WARNING:__main__:Word 'network' not in vocabulary for GloVe
```

Rank	Skip-gram	Sim	Skip-gram (Neg)	Sim	GloVe	Sim
1	income	0.3792	,	0.4035	parker	0.3405
2	for	0.3658	states	0.4029	ramsey	0.2976
3	earnings	0.356	soviet	0.3916	witnesses	0.2955
4	posts	0.3471	taken	0.3847	corn	0.2861
5	bob	0.3403	grady	0.3801	senators	0.2822
6	post	0.3355	music	0.3787	serve	0.2753
7	camp	0.3307	england	0.3783	massachusetts	0.2701
8	orders	0.3253	change	0.3755	budget	0.268
9	out	0.3233	fact	0.3721	points	0.2663

```
Similar words to 'data':
================================================================================
No results found for 'data'


Similar words to 'algorithm':
================================================================================
No results found for 'algorithm'


Similar words to 'network':
================================================================================
No results found for 'network'
```

```
Similar words to 'science':
================================================================================
WARNING:__main__:Word 'python' not in vocabulary for Skip-gram
WARNING:__main__:Word 'python' not in vocabulary for Skip-gram (Neg)
WARNING:__main__:Word 'python' not in vocabulary for GloVe
```

Rank	Skip-gram	Sim	Skip-gram (Neg)	Sim	GloVe	Sim
1	works	0.3314	urged	0.4003	doesn't	0.3474
2	congolese	0.32	benington	0.3727	eliminate	0.3293
3	sheriff	0.3009	how	0.3685	recovery	0.2973
4	weather	0.2996	lao	0.3521	none	0.2962
5	stressed	0.289	do	0.348	gubernatorial	0.2871
6	right	0.2847	proposed	0.3459	ben	0.2855
7	driven	0.2812	rules	0.3355	women	0.2854
8	declared	0.2792	gordon	0.3289	wanted	0.2667
9	leader	0.278	frank	0.324	must	0.2643

Similar words to 'python':
==
No results found for 'python'

Similar words to 'machine':
==

Rank	Skip-gram	Sim	Skip-gram (Neg)	Sim	GloVe	Sim
1	hope	0.3684	contributions	0.3766	below	0.3292

```
+
|      2 | threat      | 0.3361 | while            | 0.3745 | problem  | 0.3264
|
+--------+------------+--------+------------------+--------+---------+--------
+
|      3 | people      | 0.3242 | attend           | 0.3589 | remains  | 0.3135
|
+--------+------------+--------+------------------+--------+---------+--------
+
|      4 | boston      | 0.3204 | de               | 0.3551 | stock    | 0.3078
|
+--------+------------+--------+------------------+--------+---------+--------
+
|      5 | recommended | 0.3133 | christ           | 0.3494 | change   | 0.2955
|
+--------+------------+--------+------------------+--------+---------+--------
+
|      6 | yesterday   | 0.3086 | states           | 0.3268 | sports   | 0.294
|
+--------+------------+--------+------------------+--------+---------+--------
+
|      7 | under       | 0.3083 | entertainment    | 0.3264 | champion | 0.2919
|
+--------+------------+--------+------------------+--------+---------+--------
+
|      8 | kitchen     | 0.2988 | areas            | 0.3184 | senators | 0.2806
|
+--------+------------+--------+------------------+--------+---------+--------
+
|      9 | business    | 0.2967 | district         | 0.3179 | shares   | 0.2799
|
+--------+------------+--------+------------------+--------+---------+--------
+


Similar words to 'learning':
================================================================================
WARNING:__main__:Word 'artificial' not in vocabulary for Skip-gram
WARNING:__main__:Word 'artificial' not in vocabulary for Skip-gram (Neg)
WARNING:__main__:Word 'artificial' not in vocabulary for GloVe
```

Rank	Skip-gram	Sim	Skip-gram (Neg)	Sim	GloVe	Sim
1	rose	0.3376	harris	0.3955	found	0.4245
2	criminal	0.3365	been	0.3683	mills	0.3497
3	wisdom	0.3349	pitching	0.3565	ruling	0.3168
4	tshombe	0.3134	harvey	0.3411	b.	0.3119
5	time	0.3044	ap	0.3361	why	0.3105
6	going	0.3033	executive	0.335	officer	0.3087
7	each	0.2848	struck	0.3273	advance	0.2924
8	joan	0.2836	gin	0.3244	john	0.291
9	furniture	0.2828	senate	0.3219	adopted	0.2843

```
Similar words to 'artificial':
================================================================================
No results found for 'artificial'
```

Fun Analysis of Top 10 Words predicted by various models 🤩

I. Common Words Performance

- For common words like "good", "day", "time", and "person", all models found related words but with varying degrees of semantic relevance Skip-gram with Negative Sampling (NEG) generally produced higher similarity scores (0.40-0.45) compared to basic Skip-gram (0.30-0.35) and GloVe (0.25-0.35)
- For the word "time", Skip-gram captured temporal relationships ("previous", "last") while Skip-gram NEG focused more on contextual usage ("audience", "responsibility") GloVe showed better performance in capturing related concepts, like "individuals" for "person" and "competition" for "time"
- Technical terms like "data", "algorithm", "network", "python", and "artificial" were not in the vocabulary, indicating limitations of the training corpus

This proves that training data was likely news-focused (which is the Brown corpus in ouyr case) rather than technical or scientific text.

II. Model-Specific Insights

1. Skip-gram Model

- Tends to find grammatically similar words
- Shows lower similarity scores overall (mostly 0.30-0.35)
- Often captures syntactic relationships better than semantic ones

2. Skip-gram with Negative Sampling

- Produces higher similarity scores (0.35-0.45)
- Shows better performance in capturing contextual relationships
- More computationally efficient while maintaining good quality of word relationships

3. GloVe Model

- More balanced between syntactic and semantic relationships
- Generally produces moderate similarity scores (0.25-0.35)
- Shows better performance in capturing domain-specific relationships

III. Limitations and Observations

- The absence of technical terms suggests a domain-specific bias in the training corpus
- All models struggle with rare words or domain-specific terminology
- The similarity scores vary significantly across models, indicating different approaches to capturing word relationships
- The vocabulary size appears limited, which affects the models' ability to represent a broad range of concepts

Thank You :)

In [ ]:

```

```