Word Sense Disambiguation

Team 64

May 8, 2024

Arjun Dosajh, Mihika Sanghi

1 Introduction

Word Sense Disambiguation (WSD) is a long-standing and crucial challenge in Natural Language Processing (NLP), aiming to resolve lexical ambiguity by determining the appropriate meaning or sense of a polysemous word within a given context. The ability to accurately disambiguate word senses is fundamental for various NLP applications, including machine translation, information retrieval, text summarization, and semantic analysis.

Despite significant advancements in the field, WSD remains a complex problem due to the inherent ambiguity of natural language, where words can have multiple senses, and their meanings are highly dependent on the surrounding context. Addressing this challenge requires sophisticated algorithms and models capable of leveraging contextual information, semantic knowledge, and linguistic features to disambiguate words accurately.

This study presents a comprehensive investigation of different approaches to WSD, encompassing both traditional algorithms and state-of-the-art neural network models. By evaluating the performance of these techniques on benchmark datasets, this research aims to provide insights into the strengths, limitations, and trade-offs of various WSD methods, ultimately contributing to the development of more robust and effective solutions for resolving lexical ambiguity in natural language processing tasks.

2 Datasets

- Brown Corpus: A collection of text samples in American English, compiled by Brown University in the 1960s. Contains 1,014,312 words of running text from diverse sources, including news articles, editorials, interviews, fiction, scientific articles, and academic texts.
- SemCor 3.0: A sense-tagged subset of the Brown Corpus, where each word is labeled with its sense according to WordNet 2.0. Contains around 235,000 words and is widely used as a benchmark dataset for word sense disambiguation tasks. Considered challenging due to its size, diverse text genres, and the ambiguity of words in context.

Data Preparation: The data is prepared from the .naf files of the Brown Corpus, containing the complete text and other information within the ¡terms¿ tags. The prepared dataset has the following format:

1. file: The path of the .naf file.
2. context: The context of the target word, including 10 words before and after the target word.
3. context_pos: The same context with part-of-speech tags.
4. target_word: The index of the target word (e.g., w1).
5. gloss: Possible gloss (definition) of the word using the WordNet definition.
6. is_proper_gloss: A boolean value indicating whether the gloss is the proper definition according to the context.

7. wn_index: A unique identifier for the sense of the word in WordNet, represented as a string in the format 'lex_name%ss_type:lex_filenum:lex_id::'. For example, the sense identifier for the word "person" in the context "a person who plays a musical instrument" is 'person%1:03:00::'.

This data preparation process transforms the raw corpus data into a structured format suitable for training and evaluating word sense disambiguation models.

3 Baseline

Word Sense Disambiguation (WSD) is typically approached using two main methods: supervised and knowledge-based. This study explores both approaches by implementing three baseline models: two supervised (K-Nearest Neighbor and Naive Bayes Classifier) and one knowledge-based unsupervised (Lesk Algorithm).

3.1 Supervised WSD Methods

Supervised methods leverage sense-annotated training data, such as SemCor, to learn the mapping between words and their senses. A common baseline for these methods is the Most Frequent Sense (MFS) heuristic, which selects the most frequent sense for each target word in the training data. Some supervised methods, particularly neural architectures, may also utilize the SemEval 2007 dataset for development and evaluation.

1. **K-Nearest Neighbor (KNN)**: In the context of WSD, KNN works by finding the k-nearest labeled instances (neighbors) in a training dataset to a given input word. The sense of the input word is then predicted based on the most frequent sense among these neighbors. The value of k, which determines the number of neighbors considered, can significantly impact the algorithm's performance.

2. **Naive Bayes Classifier**

   Naive Bayes is a probabilistic algorithm based on Bayes' theorem, which calculates the probability of a hypothesis (sense of a word) given the evidence (word's context). The algorithm assumes conditional independence between features (context words) given the class (word sense). In WSD, Naive Bayes trains a probabilistic model on labeled instances, then uses Bayes' theorem to calculate the probability of each sense given the input word's context, selecting the sense with the highest probability.

3.2 Knowledge-based WSD Method

Unlike supervised methods, knowledge-based methods rely on the properties of lexical resources, such as WordNet or BabelNet, to determine the most appropriate sense for a word in context. The first sense given by the underlying sense inventory (e.g., WordNet 3.0) is often used as a baseline for knowledge-based systems.

1. **Lesk Algorithm**

   The Lesk algorithm is a knowledge-based approach that involves looking up the definitions of candidate senses for a given word in a dictionary. It selects the sense whose definition has the most overlap (in terms of shared words) with the word's context. This algorithm takes advantage of the fact that senses of a word are often defined using words also present in other senses, allowing disambiguation based on the overlap between defining words and the context. Lesk blah blah Lesk blah

Table 1: Accuracy of Baseline Algorithms on SemCor 3.0

Algorithm	Dataset (SemCor 3.0)
K-Nearest Neighbor	69.75%
Naive Bayes	22.4%
Simple Lesk	34.65%
Extended Lesk	49.42%

4 BiLSTM

4.1 Dataset Preparation

The dataset is prepared by converting context words into indices. The senses of the words are given different indexing with different vocabularies for them. A dictionary is created to store different senses for the target word (lemma) in the corpus. Words with frequency less than 2 are converted into an "UNK" token, and finally, the sentences are padded to a fixed length. The dataloader provides input indices for the context, target sense, and the target word.

4.2 Model Architecture

The `biLSTMModel` class is a PyTorch module that inherits from the `nn.Module` class. The constructor of the class takes several arguments: `input_size` (the size of the input vocabulary), `hidden_size` (the number of hidden units in the LSTM layer), `sense_vocab` (a list of possible sense labels), `embedding_dim` (the dimensionality of word embeddings), and `dataset` (the dataset object containing the training and test data). The constructor initializes several attributes of the class, including the input size, hidden size, sense vocabulary, embedding dimension, embedding layer, LSTM layer, and linear layer. It also initializes the sense-to-index and index-to-target dictionaries.

The `forward` method is defined, which takes two arguments: `x` (the input sequence of word indices) and `target_word` (the indices of the target words in the input sequence). The input sequence is first passed through the embedding layer to get the word embeddings. The dimensions of the embedding tensor are then permuted to match the input format expected by the LSTM layer. The embeddings are then passed through the LSTM layer to obtain the output tensor.

The LSTM layer is defined as a bidirectional LSTM, which means that it processes the input sequence in both forward and backward directions. The final hidden state of the LSTM layer is extracted by concatenating the last hidden states of the forward and backward LSTM layers. The concatenated hidden state is passed through the linear layer to obtain the output tensor. A loop is then used to apply the softmax function to the sense labels that start with the lemma of the target word. This is done to restrict the output probabilities to only those senses that are relevant to the target word. The final output tensor is returned.

Table 2: Accuracy of BiLSTM

Dataset	Accuracy
SemCor 3.0	66.33%

5 BERT Fine-Tuned Model for Word Sense Disambiguation

Our approach adapts the BERT model for Word Sense Disambiguation (WSD) by fine-tuning it on a sequence-pair binary classification task. The training data consists of context-gloss pairs, where each pair includes a sentence with a target word to be disambiguated (context) and a candidate sense definition of the target word (gloss) from a lexical database like WordNet.

During fine-tuning, each context-gloss pair is classified as either positive or negative, depending on whether the sense definition corresponds to the correct sense of the target word in the given context. Instead of treating each context-gloss pair as an independent training instance, we group related pairs

as a single training instance. This formulation allows us to model WSD as a ranking/selection problem, where the most probable sense is ranked first. By processing all candidate senses together, the model can learn better discriminative features between positive and negative context-gloss pairs.

5.1 Dataset

The dataset used for training consists of annotated sentences from SemCor 3.0 (part of the Senseval-3 corpus) and sense definitions from WordNet 3.0. SemCor is a large corpus of English text annotated with WordNet senses, containing over 226k sense-tagged word occurrences across various text genres, including newswire, fiction, and non-fiction.

For testing, datasets from the SemEval-2007, SemEval-2013, and SemEval-2015 shared tasks on Word Sense Disambiguation are used. These datasets contain sense-annotated instances of target words from various domains, such as news, science, and technology, allowing for a fair evaluation of the WSD model's performance on unseen data.

5.2 Dataset Preparation

A CSV file is created containing a dataset for word sense disambiguation. The code reads an XML file containing sentences and word instances, along with a gold key file indicating the correct sense for each instance. The resulting dataset has columns for 'id', 'sentence', 'sense_keys', 'glosses', and 'target_words'.

5.3 Feature Generation

For each record (instance), the sentence and each gloss are tokenized using the BERT tokenizer. The sentence and gloss are then combined, and the sequence is truncated if necessary to fit within the maximum length. Input features are created by concatenating the tokens for the sentence and gloss, adding special tokens ('[CLS]' and '[SEP]') to mark the beginning and end, and adding segment IDs to indicate which tokens belong to the sentence and which belong to the gloss. A label ID is also created based on whether the gloss is related to the target word in the sentence.

5.4 Model Architecture

The BERT for WSD model architecture consists of the following layers:

- **BertModel**: The base BERT model with pre-trained weights, consisting of 12 transformer layers with attention mechanisms.
- **Dropout**: A dropout layer for regularization to prevent overfitting.
- **Linear**: A fully connected layer with one output neuron used for ranking the glosses to select the correct sense of the target word.

During the forward pass, the input batch is passed through the BertModel, and the output from the second-to-last transformer layer (pooled output) is extracted. This output is then passed through the Dropout layer and the Linear layer to produce a single output value, which is used to rank the glosses and select the correct sense during training.

Table 3: Accuracy of BERT-Based Model

Dataset	Accuracy
SemEval-2013	74.63%
SemEval-2015	78.49%
Combined	74.13%

6 Conclusion

Among the non-neural models, we employed the Lesk algorithm, which is an unsupervised approach, and its accuracy was not satisfactory. On the other hand, the supervised K-Nearest Neighbor (KNN) algorithm achieved a slightly higher accuracy compared to the Naive Bayes Classifier (also supervised). However, the BERT-based Word Sense Disambiguation (BERT-WSD) model outperformed all the other models, demonstrating good performance.

The transformer architecture employed in BERT is based on the concept of attention, which allows the model to selectively focus on the relevant parts of the input while ignoring irrelevant information. The attention mechanism in BERT is used to compute the importance of each word in the input sentence, based on its relationships with other words in the sentence.

BERT is a bidirectional model, meaning that it can take into account the context of a word both before and after it in a sentence. This is achieved through a process called masked language modeling, where a certain percentage of the input tokens are randomly replaced with a special token `[MASK]`, and the model is trained to predict the original token from the surrounding context.

In addition to masked language modeling, BERT also utilizes a process called next sentence prediction, where it is trained to predict whether a pair of sentences are consecutive or not. This helps the model capture the relationships between sentences and improve its understanding of the context in which words are used.

Overall, the transformer-based architecture used in BERT, along with its self-supervised pretraining, attention mechanism, and bidirectional modeling, allows it to capture more complex relationships between words and their contexts, making it a highly effective model for tasks like word sense disambiguation.