NLP Assignment-1

Bala Gayatri Sruthi Manikonda

October 2024

1 Problem Statement-1: Open Information Extraction from Sentences

The goal of this task is to design a supervised model to perform Open Information Extraction. This model can extract tuples from a sentence, in the form (subject, relation, object, time, location). A single sentence might be able to give rise to more than one tuple depending on its structure and complexity. In this report, we present a supervised approach to OpenIE, using Python, and will discuss the dataset utilized, evaluation and error analysis of the model. We evaluated the model using the CaRB metric, which is a benchmark designed for OpenIE.

1.1 Methodology

This OpenIE task is a type of sequence labeling problem. Each word in a sentence is tagged with one of the six labels:

- ARG1: Subject of the sentence
- ARG2: Object of the sentence
- REL: Relation between ARG1 and ARG2
- TIME: Time of the event
- LOC: Location of the event
- NONE: Words that are not part of the extraction

We utilized a pre-trained BERT model to assign each word in the sentence with one of these labels.

1.2 Model Architecture

For this sequence labeling task, we utilized BERT (Bidirectional Encoder Representations from Transformers; which is a transformer-based architecture). BERT is able to understand contextual information and handle complex sentence structure. The architecture has:

- BERT Encoder: This processes the sentence and generates the contextual embeddings for each for each token.
- Classification Layer: This gives the probability distribution over the six labels for each token.

The code is designed to return multiple tuples wherever possible for the given sentences. We do this using the generate-predictions function, in the loop where the tokens are grouped into the different labels. Multiple tuples are able to be generated because:

- Each time the labels ARG1, REL and ARG2 are found in the token sequence, a tuple is created and added to the list of extractions. Then, the tuple is reset which allows for further extractions from the same sentence.
- Since a new tuple is created and added to the extractions list whenever a subject-relation-object triplet is found, more than one of these tuples can be stored for a single sentence.
- The code also attempts to capture the optional TIME and LOC labels in the tuple. Even if these are found later in the sentence, the main extraction (subject-relation-object) stays independent.
- If a sentence has more than one subject-relation-object triplet, they will all be returned as separate extractions

1.3 Dataset and Preprocessing

The Training Data consisted of sentences that were paired with their respective sequence labels, where each word is annotated with one of the six mentioned labels (ARG1, ARG2, REL, TIME, LOC, NONE).

For data pre-processing, we tokenized the sentences using BERT tokenizer. For tokens that were split into multiple subwords, only the first subword was assigned the label, while the remaining were ignored. We also did padding and truncation to make sure that all the input sentences had a fixed length, to make it appropriate for batch processing in BERT.

1.4 Training and Setup

The model was fine-tuned on the pre-processed data with:

- AdamW Optimizer (learning rate: 5e-5)
- Batch Size = 8
- Epochs = 3

The training was done using the PyTorch framework, on a GPU. For token classification we used BertForTokenClassification model. The Training loop worked on backpropagation and gradient updates based on the cross-entropy loss.

After training was completed, both the model and tokenizer were saved using the hugging face library's function, save-pretrained().

1.5 Results

With our own test, train and validation split we are able to get the following results:

Metric	Value
Test Precision	0.4896
Recall	0.3697
F1 score	0.4042

Epoch	Loss
Epoch 1	0.8602
Epoch 2	0.7081
Epoch 3	0.6468

We used the CaRB metric to evaluate our results. We trained the model on approximately 1000 sentences from the original dataset (about 5000 rows of the dataset provided). We gathered the following observations with respect to the datasets provided.

Value	Incorrect Extractions
53.55%	Correct relation phrase, incorrect arguments
29.13%	Over-specified relation phrase
17.32%	Other, including POS/chunking errors

Table 1: Majority of the incorrect extractions returned are due to errors in Correct relation phrase, incorrect arguments

The highest percentage of incorrect extraction was when the model identified the correct relation but failed to capture the correct arguments. This could be since the argument boundary detection can be difficult. This was seen to occur predominantly with complex and specific structures. The model is often selecting the wrong subject for object. Over specified relation phrase came in a close second where over specification is occurring. The model captures surrounding context as well. POS errors occur when the model is misinterpreting the sentence structure.

The highest percentage of incorrect extraction was when the model identifies a relation but fails to extract the arguments. This is a problem when the arguments show ambiguity or complexity in sentence structure. Also the model can identify a more specific relation because it overfits to a more specific relation

Model	Missed Extractions
67.15%	Could not identify correct arguments
17.22%	Identified a more specific relation
15.63%	POS/chunking error

Table 2: Majority of the incorrect extractions returned are due to errors in Could not identify correct arguments

than needed. Some relations are also missed completely due to POS tagging or tokenisation errors.

2 Problem Statement 2: Word2Vec Algorithms

This report presents the implementation of Skip Gram (with softmax and negative sampling) and Continuous Bag of Words (CBOW) models from scratch using the WikiText-20-raw dataset. The models were trained using sliding windows of size 2, 3, and 5, with an embedding size of 25. To overcome memory constraints during pair creation, batch processing was implemented using pickle files. Training was conducted in batches of 1,000 rows, with each epoch taking around 6 minutes. Results were evaluated using Mean Reciprocal Rank (MRR) on a portion of the dataset.

2.1 Word2Vec

Word embeddings play a crucial role in natural language processing (NLP) by capturing semantic relationships between words. Word2Vec, a popular method, includes two architectures: Skip Gram and Continuous Bag of Words (CBOW). In this project, we implement these models using the WikiText-20-raw dataset to understand their performance.

2.2 Methodology

2.2.1 Skip Gram and CBOW

Skip Gram aims to predict context words from a given target word, whereas CBOW predicts a target word from context words. We implemented both models from scratch using NumPy.

In Skip Gram, two variants were explored:

- Softmax-based Skip Gram
- Skip Gram with Negative Sampling

For CBOW, the context words were averaged to predict the target word.

2.2.2 Model Architecture

Both the Skip Gram and CBOW models were implemented with a single hidden layer. The architecture can be described as follows:

- **Input Layer**: The input is a one-hot encoded vector representing either the target word (for Skip Gram) or the context words (for CBOW). For a vocabulary of size V, this input vector is of size V.
- **Hidden Layer**: The hidden layer acts as the embedding layer, which maps the high-dimensional input (size V) to a lower-dimensional vector (embedding size n). The weight matrix $W \in R^{V \times n}$ transforms the one-hot encoded input into a dense vector of size n. This embedding captures the semantic meaning of the words.
- **Output Layer**: - For **Skip Gram**, the output layer is responsible for predicting the context words from the target word. It does this by multiplying the hidden layer's output with a second weight matrix $W' \in R^{d \times V}$, followed by a softmax or negative sampling to produce a probability distribution over the vocabulary. - For **CBOW**, the model predicts the target word from the averaged context word embeddings. The context word vectors are summed or averaged before being passed to the output layer. The same weight matrix W' is used to compute probabilities over the vocabulary using softmax or negative sampling.
- **Loss Function**: - For **Skip Gram**, cross-entropy loss was used with softmax for full vocabulary prediction, and negative sampling was used to improve training speed. - For **CBOW**, the loss is calculated similarly, with the model predicting the target word from the context embeddings.

The forward pass for both models follows the same steps: transforming the input one-hot vector into an embedding, applying a linear transformation, and computing the output probabilities. The hidden layer's weights serve as the learned word embeddings.

2.3 Dataset and Preprocessing

The WikiText-20-raw dataset was used. A subset of 5,000 training rows and 500 testing rows were extracted from the total 37,000 and 4,000 rows, respectively. Sliding windows of sizes 2, 3, and 5 were applied to create word pairs.

2.4 Memory Optimization with Pickle

Due to the large memory overhead in storing word pairs, we used Python's pickle module to save intermediate data to disk, allowing batch processing during training. The training was conducted in batches of 1,000 rows at a time, and each epoch took approximately 6 minutes to complete with an embedding size of 25.

2.5 Experiments and Results

2.5.1 Training Setup

We trained both models with an embedding size of 50 and sliding windows of sizes 2, 3, and 5. Training was conducted on 10,000 rows from the WikiText-20-raw dataset, and results were evaluated on 1,000 testing rows.

2.6 Results

The performance of each model was evaluated using the Mean Reciprocal Rank (MRR). Table 3 shows the MRR values for Skip Gram (with softmax and negative sampling) and CBOW for different window sizes.

Model	Window Size 2	Window Size 3	Window Size 5
Skip Gram (Softmax)	0.133483219	0.134727898	0.137111444
Skip Gram (Negative Sampling)	0.133925360	0.13527525	0.136612144
CBOW	0.133322492	0.13303910	0.132717317

Table 3: Mean Reciprocal Rank (MRR) Results for Skip Gram and CBOW

2.7 Discussion

The results indicate several important trends regarding model performance across window sizes and between the two Skip Gram variants (softmax and negative sampling).

2.7.1 Window Size Impact on Skip Gram

For both Skip Gram models (softmax and negative sampling), the MRR increases as the window size increases. This can be attributed to the nature of how context words are predicted in Skip Gram. With larger windows, the model is exposed to a broader context, allowing it to better capture semantic relationships across a wider range of words. This helps the model generalize more effectively, leading to improved MRR scores as the window size increases. In essence, the model learns more robust representations of target words when it has access to more context, which is reflected in the higher MRR values.

2.7.2 Window Size Impact on CBOW

In contrast, for the CBOW model, the MRR decreases as the window size increases. This can be explained by the way CBOW predicts the target word from the context words. With larger window sizes, the averaged context embeddings tend to lose specificity, as the model focuses on a broader range of context words. As the window expands, the context words become more diffuse, making it harder for the model to predict the exact target word accurately.

Consequently, the model's performance declines with increasing window size, reflected by the decreasing MRR.

2.7.3 Skip Gram with Negative Sampling vs. Skip Gram with Softmax

Interestingly, Skip Gram with negative sampling consistently outperformed Skip Gram with softmax. The main reason for this is the efficiency and regularization provided by negative sampling. Softmax attempts to compute the full probability distribution over the entire vocabulary, which can become computationally expensive and prone to overfitting, especially for large vocabularies. On the other hand, negative sampling only updates a small number of negative examples (randomly chosen incorrect words), making the training faster and more efficient. This also acts as a regularization technique, helping the model to generalize better and focus on learning meaningful word embeddings. As a result, Skip Gram with negative sampling achieves higher MRR values compared to the full softmax-based approach.

2.8 Conclusion

This report detailed the implementation of Skip Gram (with softmax and negative sampling) and CBOW models using the WikiText-20-raw dataset. By leveraging batch processing with pickle, we managed memory efficiently and achieved meaningful results using the MRR metric. Each epoch took around 6 minutes to train in batches of 1,000 rows at a time. Future work could involve experimenting with larger datasets, more sophisticated sampling techniques, and integrating pre-trained embeddings like GloVe or BERT.