Project Report For Team 9

Table of Contents

1. **Introduction and Motivation** 1
2. **Problem Definition** 1
3. **Data** 1
4. **Implementation Approaches** 2
5. **Results** 3
6. **Conclusion** 4
7. **References** 4

1. Introduction and Motivation

Relation extraction is a fundamental NLP task that plays a crucial role in knowledge extraction and information retrieval. It involves identifying and categorizing the semantic relationships that exist between entities (such as people, organizations, locations, and more) mentioned in text documents. This task is essential for various applications, including information retrieval, question answering, knowledge graph construction, and sentiment analysis. Relation Extraction helps in converting unstructured text data into structured knowledge, making it easier for machines to comprehend and reason about the information. By identifying relationships between entities, we can unlock valuable insights, discover hidden connections, and create knowledge graphs that represent structured information.

2. Problem Definition

We have done Relation Extraction using various models in our project. This is Multi-Way Classification of Semantic Relations between Pairs of Nominals. The goal is to develop a system that can automatically identify and classify relationships between entities mentioned in text.
Example:

Sentence: The <e1>burst</e1> has been caused by water hammer <e2>pressure</e2>.
Entities:

Entity 1: burst (denoted as <e1> burst </e1>)
Entity 2: pressure (denoted as <e2>pressure</e2>)

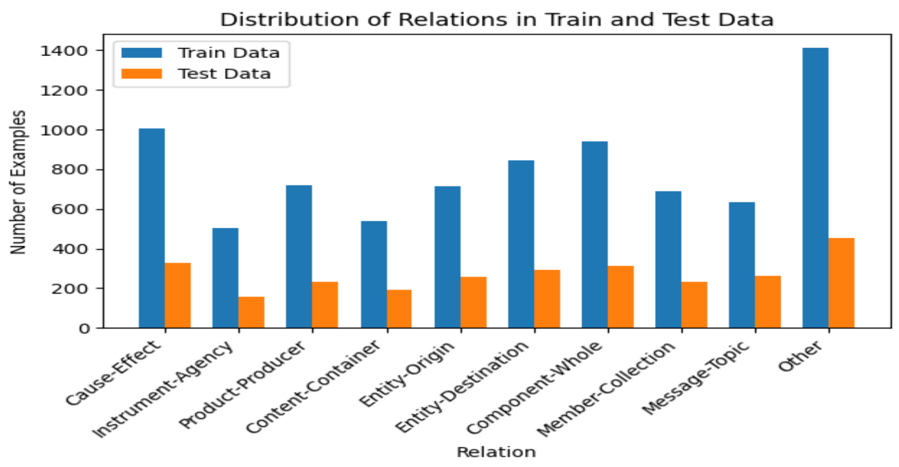
Relation: Cause-Effect (e2, e1)

3. Data

Dataset used for the project is **SemEval-2010 Task 8 Dataset [Download].** Below is the breakup of the number of train and test data instances.

Train data instances	8000
Test Data instances	2717

Classes Distribution: The goal of the project was to classify pairs of nominals (noun phrases) into predefined semantic relation classes. These classes encompassed diverse semantic relationships, such as Cause-Effect, Instrument-Agency, Product-Producer, Content-Container, Entity-Origin, Entity-Destination, Component-Whole, Member-Collection, Message-Topic, and an additional class for relations not falling into the specified categories, labeled as "Other." Below bar-chart represents the number of examples catering to each of the relation in Train & Test data.



4. Implementation Approaches

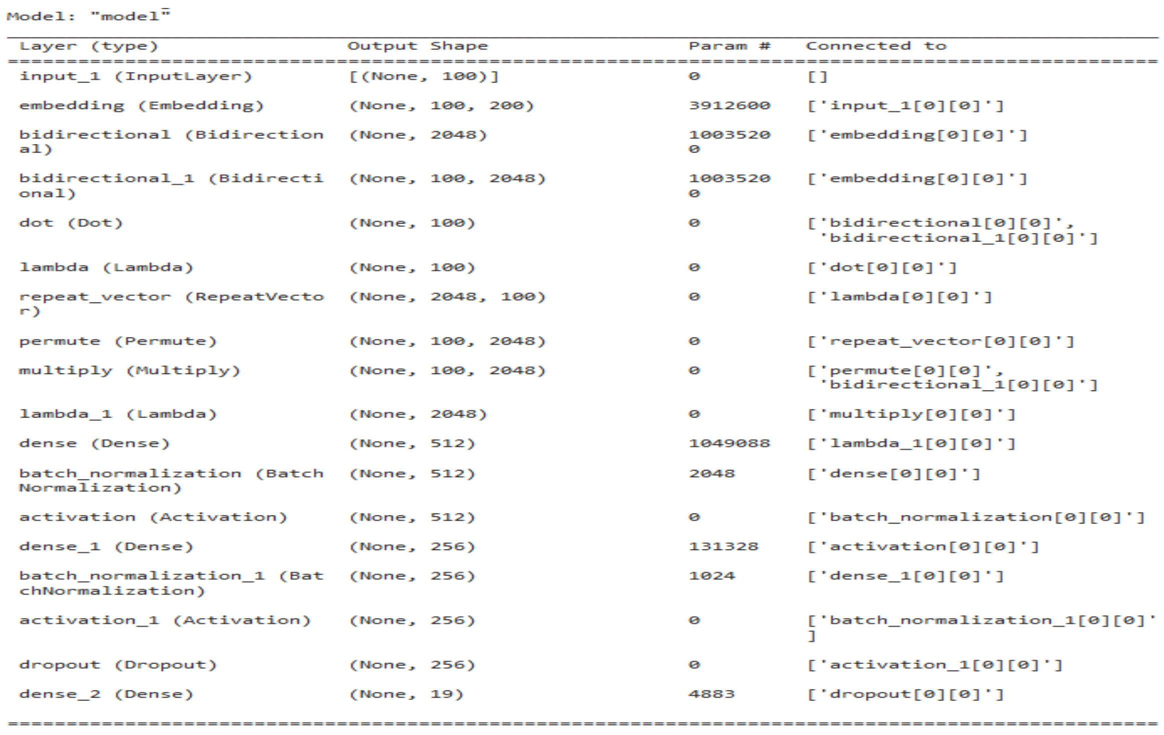
As part of the project, we explored the below approaches for relation identification & classification.

1. Attention based Bi-LSTM
2. CNN
3. BERT

Below is a brief description of each of the above implementations.

4.1 Bi-directional LSTM with attention mechanism

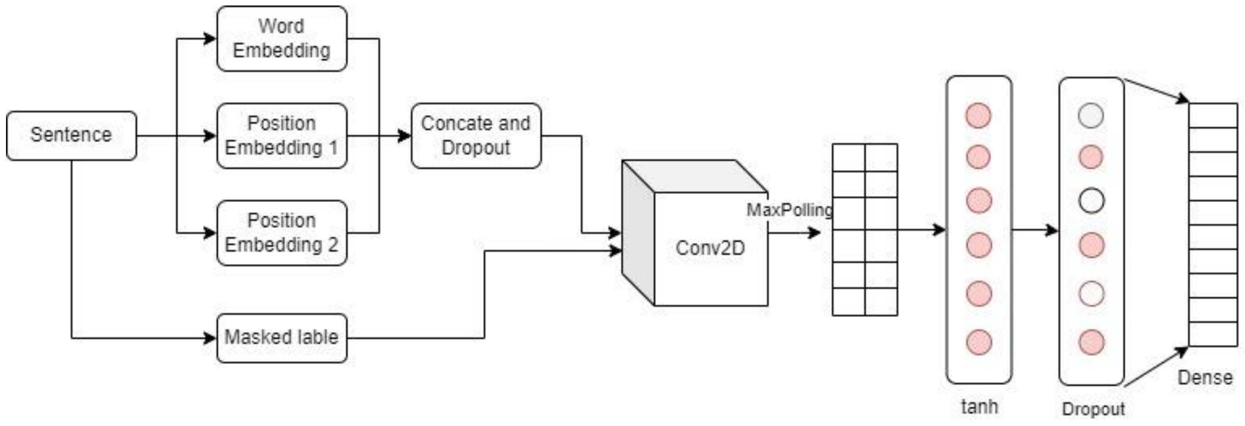
The model implementation uses bidirectional LSTM layers with a hidden dimension of 1024, dot product attention, and weighted sum of LSTM outputs. Two dense layers (512 and 256 units), batch normalization, ReLU activation, and a dropout layer (0.5 rate) are included. The output layer with softmax activation aligns with the number of classes. L2 regularization (coefficient: 0.0001) targets LSTM layers for robustness. Glove embeddings (6B_50D & 6B_200D) were used for generating the embedded vector representation. Below is the visual representation of the underlying architecture.



```
Model: "model"
_______________________________________________________________________________________________
 Layer (type)                  Output Shape                 Param #   Connected to
===============================================================================================
 input_1 (InputLayer)          [(None, 100)]                0         []

 embedding (Embedding)         (None, 100, 200)             3912600   ['input_1[0][0]']

 bidirectional (Bidirection    (None, 2048)                 1003520   ['embedding[0][0]']
 al)                                                        0

 bidirectional_1 (Bidirecti    (None, 100, 2048)            1003520   ['embedding[0][0]']
 onal)                                                      0

 dot (Dot)                     (None, 100)                  0         ['bidirectional[0][0]',
                                                                       'bidirectional_1[0][0]']

 lambda (Lambda)               (None, 100)                  0         ['dot[0][0]']

 repeat_vector (RepeatVecto    (None, 2048, 100)            0         ['lambda[0][0]']
 r)

 permute (Permute)             (None, 100, 2048)            0         ['repeat_vector[0][0]']

 multiply (Multiply)           (None, 100, 2048)            0         ['permute[0][0]',
                                                                       'bidirectional_1[0][0]']

 lambda_1 (Lambda)             (None, 2048)                 0         ['multiply[0][0]']

 dense (Dense)                 (None, 512)                  1049088   ['lambda_1[0][0]']

 batch_normalization (Batch    (None, 512)                  2048      ['dense[0][0]']
 Normalization)

 activation (Activation)       (None, 512)                  0         ['batch_normalization[0][0]']

 dense_1 (Dense)               (None, 256)                  131328    ['activation[0][0]']

 batch_normalization_1 (Bat    (None, 256)                  1024      ['dense_1[0][0]']
 chNormalization)

 activation_1 (Activation)     (None, 256)                  0         ['batch_normalization_1[0][0]'
                                                                      ]

 dropout (Dropout)             (None, 256)                  0         ['activation_1[0][0]']

 dense_2 (Dense)               (None, 19)                   4883      ['dropout[0][0]']

===============================================================================================
```

4.2 CNN

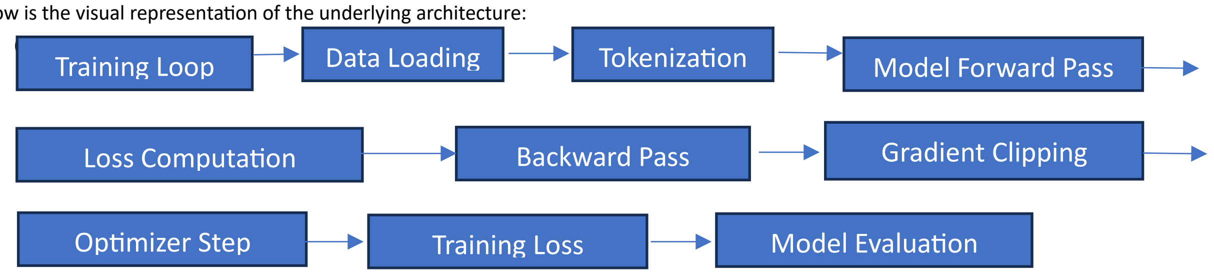
The CNN model is tailored for sequence classification tasks with input sequences of dimensions Batch_size * 4 * L, where L represents the sequence length. For Embeddings, we have used Glove-6B-50D Incorporating word embeddings and position embeddings for entity positions, the model employs a 2D convolutional layer with 200 filters, 'same' padding, Tanh activation, and dropout. Subsequent max pooling reduces the sequence length dimension to 1. A linear layer with Tanh activation and dropout further transforms the data to Batch_size * hidden_size. The final linear layer serves as the output layer, producing logits for each class in Batch_size * class_num (10 classes). This architecture is designed for effective feature extraction and classification in sequential data.



4.3 BERT

A fine-tuning pipeline for training BERT-based models on a text classification task is implemented using PyTorch. Two pre-trained BERT models, 'bert-base-uncased' and 'bert-large-uncased', are employed, and their corresponding tokenizers are utilized to process and tokenize the training, validation, and test datasets. The training loop involves the use of the AdamW optimizer with specified hyperparameters, such as a learning rate of 2e-5 and weight decay of 5e-3. The training process occurs over five epochs, with each epoch displaying the average training loss. The code efficiently organizes data using DataLoader instances and ensures proper gradient clipping during model training.

Below is the visual representation of the underlying architecture:



5. Results

For each of the implementations, we generated a predict.txt file containing the estimated relations for the sentences in the train data and compared with the answer key. Shared below is the Evaluation Metrics table containing the models and their corresponding performance details.

Model	Evaluation Metrics				
	Accuracy	Balanced Accuracy (Test)	Precision	Recall	F1-Macro score
Bi-LSTM	0.68	0.72	0.69	0.70	0.69
CNN	0.78	0.75	0.78	0.75	0.76
BERT base	0.82	0.82	0.85	0.81	0.82
BERT large	0.84	0.84	0.88	0.83	0.85

Below are the classification reports for each of the implementations.

Classification Report for Bi-LSTM:

```
                    precision    recall  f1-score   support

     Message-Topic       0.76      0.90      0.82       328
             Other       0.75      0.70      0.72       312
      Cause-Effect       0.77      0.83      0.80       192
  Product-Producer       0.77      0.88      0.82       292
   Component-Whole       0.69      0.82      0.75       258
     Entity-Origin       0.67      0.58      0.62       156
  Instrument-Agency       0.75      0.78      0.76       233
 Content-Container       0.76      0.79      0.77       261
Entity-Destination       0.46      0.37      0.41       454
 Member-Collection       0.71      0.55      0.62       231
```

Classification Report for CNN:

```
                   precision    recall  f1-score   support

 Instrument-Agency       0.90      0.88      0.89       335
Entity-Destination       0.80      0.71      0.75       353
     Message-Topic       0.82      0.83      0.82       190
      Cause-Effect       0.93      0.80      0.86       341
     Entity-Origin       0.86      0.81      0.83       274
   Component-Whole       0.65      0.70      0.67       145
 Member-Collection       0.83      0.76      0.80       254
 Content-Container       0.82      0.78      0.80       274
             Other       0.41      0.52      0.46       354
  Product-Producer       0.64      0.75      0.69       197
```

Classification Report For BERT Large:

```
                    precision    recall  f1-score   support

             Other       0.91      0.95      0.93       328
   Component-Whole       0.82      0.88      0.85       312
 Instrument-Agency       0.90      0.91      0.90       192
  Product-Producer       0.92      0.92      0.92       292
     Message-Topic       0.92      0.87      0.90       258
     Entity-Origin       0.85      0.72      0.78       156
 Content-Container       0.86      0.85      0.85       233
      Cause-Effect       0.87      0.89      0.88       261
Entity-Destination       0.63      0.63      0.63       454
 Member-Collection       0.86      0.84      0.85       231
```

6. Conclusion

Below are few conclusions based on the implementations used in our project.

1) All the proposed method, We have used Adam optimizer. Overall for development, we have used mainly Keras and Torch library with some other supportive libraries(Numpy , Pandas).
2) Performance has increased with increased embedding dimensions (200D vs 50D).
3) Overall, BERT is the best performing model with an accuracy of 0.84.
4) Out of the classes, entity-destination class having low predictability.
5) Owing to advancement in transformer models, it is good if we can explore more on Transformer models for Relation-Extraction task.

7. References

[1] "Bidirectional Long Short-Term Memory Networks for Relation Classification."
URL: https://aclanthology.org/Y15-1009.pdf
[2] “Relation Classification via Convolutional Deep Neural Network”
URL: https://aclanthology.org/Y15-1009.pdf
[3] “Relation Extraction: Perspective from Convolutional Neural Networks”
URL: https://aclanthology.org/W15-1506.pdf
[4] “Relation classification via BERT with piecewise convolution and focal loss”
URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8432804/pdf/pone.0257092.pdf