Relation Extraction From Natural Language Processing Using Pytorch

Shriya Sravani Y

Abstract

This paper presents a deep learning-based approach for multi-label relation extraction from user utterances in conversational systems, with the underlying relations based on the Freebase schema. We employ a Multi-Layer Perceptron (MLP) model combined with different word embedding techniques to predict relations for each utterance. This approaches use binary cross-entropy loss for multi-label classification, and regularization techniques like dropout to prevent overfitting. The model is trained and evaluated on utterances paired with their corresponding relations, achieving satisfactory performance for relation prediction on unseen test data.

1 Introduction

The rise of conversational agents and virtual assistants has increased the need for accurate natural language understanding, particularly in extracting semantic relations from user utterances. This task is multi-label, meaning an utterance can invoke multiple relations, such as the example "Show me movies directed by Woody Allen recently," which might invoke 'movie.directed_by' and 'movie.initial_release_date'.

The problem at hand is a supervised multi-label classification task, where the goal is to classify utterances into one or more 'CORE RELATIONS' categories. Each utterance can be associated with multiple relations from a predefined set, which requires the model to handle the inherent complexity of multi-label output. This task is particularly important in relation extraction, where the objective is to identify relevant relations from a given input, especially in the domain of movie-related data. The labels correspond to different movie attributes, such as actors, directors, languages, genres, and other movie-specific details.

The dataset consists of 2312 utterances, each mapped to one or more relation labels from a set of predefined categories. There are no missing values, and the dataset has five key features, namely, ID, UTTERANCES, CORE RELATIONS, utterance_length and num_labels.

The statistics of the dataset are given as below:

Utterance Length Distribution: The number of words per utterance ranges from 1 to 20 words, with the majority of utterances having between 5 and 10 words.

Label Distribution: The dataset contains a broad variety of relation labels, with movie.starring.actor, movie.directed_by, and none being the most frequent labels. Some utterances contain multiple labels, and a few rare labels include combinations like movie.gross_revenue, gr.amount and movie.production_companies, movie.genre.

Multi-label Distribution: The number of labels per utterance varies, with some utterances having more than one label, indicating the multi-label nature of the problem.

2 Models

The Multi Layer Perceptron is a fully connected neural network designed to learn complex patterns by connecting each neuron in one layer to all neurons in the next. In this task, the MLP is used to map vectorized utterances to multiple relation labels.

In the task of relation extraction, the goal is to identify and classify relevant relations from user-provided utterances, particularly in the context of movie-related data. To address this problem, a Multi-Layer Perceptron (MLP) model has been chosen due to its strong performance in capturing complex, non-linear relationships in data. The first model implemented uses CountVectorizer.

2.1 Approach 1(Using CountVectorizer)

To prepare the input for the MLP, CountVectorizer is used for converting the textual utterances into numerical vectors. Specifically, it creates a Bag-of-Words (BoW) representation with a vocabulary size of 5000, which is the maximum number of features. The binary option is enabled to indicate whether a word is present in an utterance, without considering the frequency. This approach provides a simple yet effective way to encode text data for training neural networks. The choice of CountVectorizer is based on its efficiency in converting text into a format that can be directly fed into machine learning models, especially when the dataset is relatively small, like in this case.

Model Architecture

Input Layer: The input to the MLP is the vectorized representation of the utterances, with 5000 dimensions corresponding to the features created by the CountVectorizer.

Hidden Layers: The first hidden layer consists of 512 neurons followed by Batch Normalization to normalize the activations and improve training stability. A ReLU (Rectified Linear Unit) activation function is applied to introduce non-linearity. A Dropout layer with a 50% rate is added after the ReLU activation to prevent overfitting by randomly disabling 50% of the neurons during training. The second hidden layer has 256 neurons, followed by another Batch Normalization and ReLU activation. Again, Dropout is applied to reduce overfitting.

Output Layer: The final output layer consists of neurons corresponding to the number of unique relation labels (multi-label classification). A sigmoid activation function is applied to each output neuron, producing probabilities for each label, which are later thresholded to classify the utterance.

2.2 GloVe

The next model leverages GloVe(GLobal Vectors for Word Representation) embeddings to represent the textual input and a MLP model is used for learning these relations from the utterances.

GloVe embeddings are pre-trained on a large corpus and are capable of capturing semantic relationships between words.Each word in the utterance is mapped to a 300-dimensional embedding vector, and the embeddings of all words in an utterance are averaged to create a single fixed-length representation for each utterance.

Model Architecture

Input Layer: The input to the MLP is the 300-dimensional GloVe embedding of each utterance.

Hidden Layers: The model contains four hidden layers with 2048, 1024, 512, and 256 neurons, respectively. Each layer is followed by Batch Normalization to stabilize and accelerate training, and Leaky ReLU activations are applied to introduce non-linearity while allowing a small gradient to pass even when the input is negative. This prevents neurons from "dying" and helps maintain gradient flow.

Dropout: Dropout regularization with a rate of 0.4 is applied to prevent overfitting by randomly disabling a portion of the neurons during training.

Output Layer: The output layer uses a sigmoid activation function, producing probabilities for each relation category, which are later thresholded to obtain binary multi-label predictions.

Model Training and Hyperparameters

Loss Function: Binary Cross-Entropy Loss (BCELoss) is used, as it is well-suited for multi-label classification tasks where each relation is treated as an independent binary classification problem.

Optimizer: AdamW optimizer with a learning rate of 0.0005 and weight decay (1e-5) is employed to minimize the loss. AdamW's adaptive learning rate and momentum properties make it a popular choice for training deep neural networks.

Learning Rate Scheduler: A Cosine Annealing Learning Rate (LR) Scheduler is used to reduce the learning rate over time, helping the model converge better. This scheduler gradually lowers the learning rate from the initial value to a small minimum (eta_min=1e-6), which stabilizes the training process.

Early Stopping: To prevent overfitting, early stopping is implemented with a patience of 10 epochs, meaning training stops if the validation loss does not improve for 10 consecutive epochs.

3 Experiments

Approach-1

The data was preprocessed by splitting the labels into a multi-label format and vectorizing the textual utterances using CountVectorizer, which converts the text into a sparse Bag-of-Words (BoW) matrix. The vectorizer is limited to a maximum of 5000 features, with binary encoding indicating the presence of each word.

Dataset split and preprocessing

The dataset consists of utterances that are labeled with one or more movie-related 'CORE RELATIONS' (e.g., movie.directed_by, movie.starring.actor).

The dataset is split as follows:

- Training Set: 80% of the data is used for training.
- Validation Set: 20% is used for validation during training to tune hyperparameters and for early stopping.
- Test Set: A separate test set is used for generating predictions for final evaluation and submission.

Handling Data Sparsity and Imbalance

The dataset shows data imbalance with some relation labels occurring much more frequently than others. To handle this, a binary cross-entropy loss function (BCELoss) is used, which treats each label as an independent binary classification problem. The binary nature of the task means that we do not require specific class weights for the imbalance; however, dropout regularization and early stopping were implemented to avoid overfitting to majority classes.

Hyperparameter Selection

Hyperparameter tuning is performed by monitoring the validation loss and performance on the validation set.

- Learning Rate: Initial learning rates of 0.001 and 0.0005 were tested, with the final value of 0.001 providing the best results in terms of loss reduction and convergence speed.
- Batch Size: Multiple batch sizes were explored, with 32 being selected for the final model as it balanced training speed and performance.
- Dropout Rate: Dropout rates of 0.3, 0.4, and 0.5 were tested, and 0.5 was chosen to effectively mitigate overfitting while retaining useful features.
- Weight Decay (L2 Regularization): To prevent overfitting and manage large weights, an L2 regularization (weight decay) of 1e-5 was added to the Adam optimizer.

Model Descriptions and Tested Configurations

- The input consists of 5000 binary features derived from the BoW vectorization. The layers consist of 512 and 256 neurons, along with Batch Normalization, ReLU activation, and Dropout (0.5). A sigmoid activation function is used to output probabilities for each relation label. Binary cross-entropy loss (BCELoss) for multi-label classification. Adam optimizer with a learning rate of 0.001 and weight decay of 1e-5.

Early stopping with a patience of 3 epochs was introduced to prevent overfitting. If the validation loss does not improve for 3 consecutive epochs, training stops. The learning rate of 0.001 was selected after validation performance tuning. A batch size of 32 was used to balance computational efficiency and model performance.

The models were evaluated using the following metrics:

Binary Cross-Entropy Loss: The loss function measures how well the model is predicting the presence or absence of each label in a multi-label setting. Both training and validation losses were monitored to detect overfitting. Accuracy: Accuracy is computed based on the proportion of utterances for which the model correctly predicted all the relevant labels. F1-Score (Micro-Averaged): The F1-score was used to evaluate the tradeoff between precision and recall across all labels, with the micro-averaged F1 being more appropriate for imbalanced datasets.

Hyperparameter	**Values**	**Final Value**
Learning Rate	0.001, 0.0005	0.001
Batch Size	16, 32, 64	32
Dropout Rate	0.3, 0.4, 0.5	0.5
Weight Decay	$1e-4$, $1e-5$	$1e-5$
Patience	3, 5, 7	3

Table 1: Hyperparameters Tested and Selected

Approach-2

The training and validation set is divided into an 80-20 split.

The utterances are first transformed into dense 300-dimensional embeddings using GloVe embeddings.

Data Imbalance Handling

The dataset exhibits some level of label imbalance, with certain relations like movie.directed_by and movie.starring.actor occurring more frequently than others. To address this, binary cross-entropy loss (BCELoss) is used. The combination of binary cross-entropy and the adaptive learning rate provided by the Adam optimizer helped the model handle label imbalance effectively.

Hyperparameter Selection

Initially, a simple MLP with two hidden layers was tested, but later, deeper models with four hidden layers and more neurons were explored. The final configuration uses five layers, with the number of neurons set to 2048, 1024, 512, and 256 in the hidden layers, respectively. Learning Rate: Multiple learning rates were tested, including 0.001, 0.0005, and 0.0001. The final model uses a learning rate of 0.0005, which was selected based on validation loss. Optimizer: The AdamW optimizer was used throughout with L2 regularization (weight decay of 1e-5). This combination was found to provide good convergence while preventing overfitting. Dropout: Dropout regularization rates of 0.3, 0.4, and 0.5 were tested, with 0.4 providing the best tradeoff between preventing overfitting and allowing the model to retain useful features. Activation Functions: Both ReLU and LeakyReLU were explored. The final model uses LeakyReLU, which provides better gradient flow and stability in deeper networks. Learning Rate Scheduler: A Cosine Annealing Learning Rate Scheduler was employed with a T_max of 10 and eta_min set to 1e-6. This strategy allows the learning rate to decrease gradually, preventing oscillation around local minima and ensuring smoother convergence. Batch Size: Different batch sizes (32, 64, 128) were tested. A batch size of 64 provided the best performance in terms of both training speed and validation accuracy. Evaluation and Model Selection Metrics The following metrics were used to evaluate model performance:

Loss: Binary cross-entropy loss was computed for both the training and validation sets. The model with the lowest validation loss was selected for generating test set predictions. Accuracy: Accuracy was calculated based on the proportion of correctly predicted labels for each utterance. F1-Score (Micro-Averaged): The F1-score was used to measure the balance between precision and recall. Since this is a multi-label classification problem, the micro-averaged F1-score was chosen, as it aggregates the contributions of all classes to compute the average F1. Early Stopping To prevent overfitting, early stopping was implemented, with a patience of 10 epochs. If the validation loss did not improve after 10 consecutive epochs, training was halted. This ensured that the model did not overfit to the training data while ensuring optimal performance on the validation set.

The model originally had 2 hidden layers (512, 256 neurons), dropout of 0.5, learning Rate of 0.001, and optimizer Adam. It was modified to have 4 hidden layers (2048, 1024, 512, 256 neurons), dropout of 0.4, learning Rate of 0.0005, Optimizer AdamW with L2 regularization (1e-5) and scheduler Cosine Annealing LR Scheduler (T_max=10, eta_min=1e-6) The final model demonstrated significantly improved performance in both loss reduction and F1-score, achieving the best generalization.

Hyperparameter	**Value**
Learning Rate	0.0005 (AdamW optimizer)
Weight Decay	$1e-4$
Dropout Rate	0.4
Loss Function	BCELoss
Batch Size	64
Optimizer	AdamW optimizer
Scheduler	Cosine Annealing LR

Table 2: Hyperparameters used in the MLP model

4 Results

The above plots compare the performance of the CountVectorizer and GloVe approaches across three metrics: Training Accuracy, Validation Accuracy, and Validation F1-Score.

Training Accuracy: Both models reach near-perfect accuracy, with the GloVe model reaching higher accuracy more quickly, suggesting that the pre-trained GloVe embeddings help the model converge faster and learn more effectively than the CountVectorizer approach.

Validation Accuracy: GloVe embeddings provide a noticeable improvement over CountVectorizer. While both models show strong performance, the GloVe model maintains a higher validation accuracy throughout most of the training epochs, especially in the later epochs.

Validation F1-Score: The GloVe model outperforms the CountVectorizer model consistently in terms of F1-Score. The GloVe approach captures a better balance between precision and recall, which is crucial in a multi-label classification problem like this one. It demonstrates better generalization as it reaches and sustains higher F1 scores compared to CountVectorizer.

In summary, the GloVe embedding approach outperforms the CountVectorizer model in terms of validation accuracy and F1-score, making it a more suitable choice for this task.

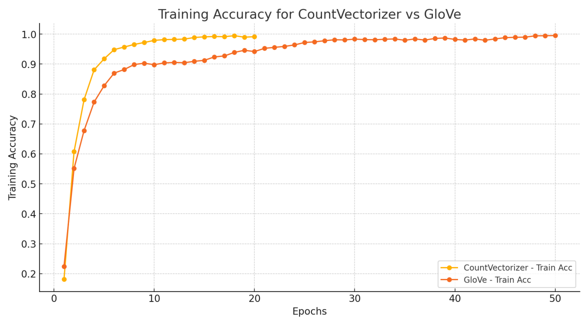

Figure 1: A figure with a caption that runs for more than one line. Example image is usually available through the `mwe` package without even mentioning it in the preamble.

4.1 Citations

Andrew Skabar, Dennis Wollersheim, Tim Whitfort *Multi-label Classification of Gene Function using MLPs*

Hamza Haruna MOHAMMED,Erdogan DOGDU,Abdul Kadir GORUR *Multi-Label Classification of Text Documents Using Deep Learning*

Shabnam Nazmia, Mohammad Razeghi-Jahromib, Abdollah Homaifarc*Multi-label Classification With Weighted Labels Using Learning Classifier Systems*