HW1: Relation Extraction from Natural Language using PyTorch

Parsa Mazaheri
UC, Santa Cruz pmazaher@ucsc.edu

Abstract

In this assignment we're going to extract relations from natural language using PyTorch. Relation Extraction is a task in Natural Language Processing (NLP) that aims to extract relations between entities in a sentence. Given a sentence, the task is to identify the relation between the two entities in the sentence. The goal of this assignment is to predict the core-relation from the utterance.

1 Introduction

The goal of this homework is to train deep learning models to determine knowledge graph relations that are invoked in utterances. Relation Extraction (RE) is the task of extracting semantic relationships from a text. Extracted relationships usually occur between two or more entities of a certain type (e.g. Person, Organisation, Location) and fall into a number of semantic categories. This subfield on Natural Language Processing falls into **Information extraction** and is one of the hot topics in NLP as the time of writing this report.

2 Problem Description

In this assignment we're using a labed dataset of utterances and relations. Since here our approach is using labed data to train a deep learning model, we're using **Supervised Learning** approach. supervised learning is a machine learning approach where we have a dataset of labeled data and we train a model to predict the label of new data.

3 Dataset

For this assignment, we're going to use the given data by the teaching assistant. The data consists of 2 files: train.csv and test.csv. The train.csv file contains 3 columns: utterance, core relation, and id. The utterance column contains the utterance that the user said to the conversational system. The core-relation column contains the relation that the user is trying to invoke. The test.csv file contains 2 columns: utterance and id. In this assignment, the data is already in a good format for us to use. We don't need to do extensive data cleaning. However, we need to do some preprocessing to the data. For that, we tokenize the data, remove the stop words (such as "a", "an", "the", "of", etc). After that lower case the data and remove the punctuation. Other than that, we remove special characters and numbers, and convert double-spaces to single spaces.
After that, we have a clean dataset for us to use. We can use this dataset to train our model and also for testing our model for evaluation.

4 Hyperparameters

One of the key challanges of every deep learning task is to determine the best hyperparameters. The hyperparameters are the parameters that are used to train the model. The hyperparameters are not learned by the model and therefore aren't updated during the training process. The hyperparameters that I've used for this task are shown in the following table.
The hyperparameters are the learning rate, batch size, number of epochs, number of hidden units, and some related to model architecture such as the number of layers, the dropout rate, and the embedding size. The best hyperparameters are determined by the user by trying different values and choosing the best one. The best hyperparameters for this task that were tested are shown in the following table.

5 Model Architecture

For the task, I've used different models from MLP to biLSTM. The best performance was achieved by the biLSTM 1 hidden layer model. The model

=pt

Hyperparameter	Value
Learning Rate	0.04
Batch Size	32
Number of Epochs	10
Number of Layers	1
Dropout Rate	0.1
Embedding Size	128

Table 1: Hyperparameters

architecture is shown in the following figure. For each of the models, the model takes the utterance as an input and passes it through an embedding layer. The embedding layer converts the words in the utterance to vectors. The vectors are then passed through the rest of the model to be processed.

For more information on each model used, we've provided a little more detailed description in the following subsections.

5.1 Multi-Layer Perceptron (MLP) Model

MLP is a simple model that is used for classification tasks. After getting the embedding vectors, the vectors are then passed through a fully connected layer. The fully connected layer is a linear layer that outputs a vector of size 50. The output of the fully connected layer is then passed through a ReLU activation function and again for two more fully connected layers. The output of the last fully connected layer is then passed through a sigmoid layer to get the probability of each relation.

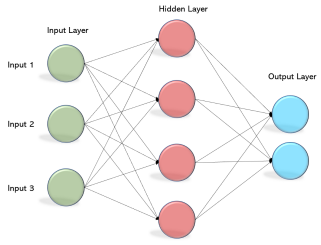

Figure 1: MLP Architecture

5.2 Bi-LSTM Model

LSTM [1] (Hochreiter and Schmidhuber, 1997) is a type of recurrent neural network that is used for sequence modeling tasks. LSTM is a type of RNN that is able to remember information for a long time. LSTM has 3 gates: input gate, forget gate, and output gate. The input gate decides which values from the input to update the cell state. The forget gate decides which values from the cell state to update. The output gate decides which values from the cell state to output. The output of the LSTM layer is a hidden state for each word in the utterance. The hidden states are then passed through a fully connected layer to get the output. The output is then passed through a sigmoid layer to get the probability of each relation.

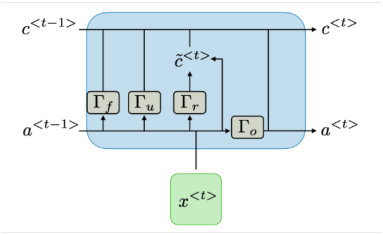

Figure 2: LSTM cell architecture

The advantage of using a bidirectional LSTM (Huang et al., 2015) layer is that it can capture the context of the utterance in **both directions**. and by doing so, it can capture the relations in the utterance better.

After getting the embedding vectors, the vectors are then passed through a biLSTM layer. The biLSTM layer outputs a hidden state for each word in the utterance. The hidden states are then passed through a fully connected layer to get the output. The output is then passed through a sigmoid layer to get the probability of each relation.

6 Experiments

Since for this research project, there wasn't a lot of time to do experiments, I've only done a few experiments. The experiments that I've done are shown in the following subsections.

6.1 Optimizer

Since this research project was done in only one day, I didn't have enough time to test different optimizers. The optimizer that I've used is Adam optimizer. Adam optimizer is a type of stochastic gradient descent optimizer. Adam is a method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients, Adam also keeps an exponentially decaying average of past gradients, similar to momentum.

6.2 Number of Hidden layers

When increasing the number of layers, the model is able to capture more complex relations. However,

[1] Long Short-Term Memory

=pt

increasing the number of layers also increases the number of parameters in the model and the context may be lost. As shown in Table 2, here that we have a small dataset, increasing the number of layers doesn't help much. The best performance was achieved by the model with 1 hidden layer.

6.3 Dropout

Dropout (Srivastava et al., 2014) is a technique for regularization. Dropout is a technique where randomly selected neurons are ignored during training. by using dropout in this problem, I didn't encountered much difference, since we're dealing with a small number of data and don't have very deep neural networks, that help with data loss.
Also, Dropout is used when we're trying to prevent overfitting. (When the model does very well on training but bad on test or evaluation set). But here the model isn't working as a perfect state-of-the-art and doesn't require dropout.

7 Result

Evaluation is scored based on mean F-1 score. This is a common evaluation metric across NLP, because it weights both precision and recall.

$$F1 = \frac{2.P.R}{P+R}$$

in which Precision is a ratio of True Positives (TP) to total positives guessed (TP + FP). Recall is the ratio of True Positives to actual positives (TP + FN).

$$P = \frac{TP}{TP+FP}, P = \frac{TP}{TP+FN}$$

By using F1-Score as the evaluation metric, the model is able to achieve a score of 0.7731. This is a good score for a model that was trained on a small dataset whitout very intensive hyperparameter tuning and model optimization.

By using F1-Score as the evaluation metric, the model is able to achieve a score of **0.7731**. This is a good score for a model that was trained on a small dataset whitout very intensive hyperparameter tuning and model optimization.

The result of using dropout on the model is shown on table 3. As mentioned in section 6.3, since here we don't have a deep network and a large dataset, dropout doesn't help very much and as seen in in the Table 3, with dropout `value=0.1`, the accuracy of the result improves by a very small amount.

Number of Layers	F1 Score
Bi-LSTM 1 layer	0.7731
Bi-LSTM 3 layers	0.7402
Bi-LSTM 5 layers	0.3765
MLP 1 layer	0.7546
MLP 2 layer	0.7422
MLP 3 layer	0.6793

Table 2: Number of Hidden Layers

Dropout Rate	F1 Score
without	0.7754
0.1	0.7755
0.2	0.7712
0.5	0.7538

Table 3: Dropout influence on the score

Figure 3: Training Loss vs Validation Loss

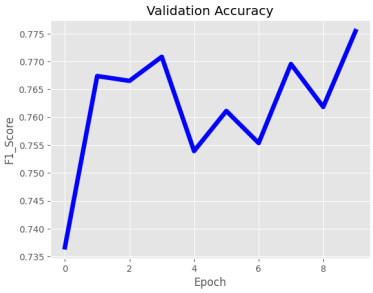

Figure 4: F1 score / Epoch

=pt

As shown in 3, we can see the training vs validation loss over the training time. and as shown in 4, we can see the F1 score over the 10 epochs.

References

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF models for sequence tagging. *CoRR*, abs/1508.01991.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958.

=pt