HW 2: Slot Tagging for Natural Language Utterances

Parsa Mazaheri
UC, Santa Cruz `pmazaher@ucsc.edu`

Abstract

1 Introduction

The goal of this homework is to train a model to tag slots in natural language utterances of users addressing a virtual personal assistant. The tagged slots and the associated values are traditionally used to accomplish the request of the users.

For example, if a user asks to learn about movies of a specific director, we would need to identify the name of the director, and issue a query to the back-end knowledge graph to get information for formulating a response back to the user. This subfield on Natural Language Processing falls into **Information extraction** and is one of the hot topics in Natural Language Processing.

2 Problem Description

In this assignment we're using a labeled dataset of utterances and tags. Since here our approach is using labeled 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 dataset, we're using the given data by the course instructors. The data consists of 2 files: train.csv and test.csv. The train.csv file contains 3 columns: utterance, tag slots, and id. The utterance column contains the utterance that the user said to the conversational system. The tag slots column contains the tags 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.

The preprocessing phase consist of different actions which has been explained below.

4 Preprocessing

Data preprocessing is a step in data analysis that takes raw data and transforms it into a format that can be understood and analyzed by computers and machine learning models. Good, preprocessed data is more important than the most powerful algorithms. By doing a good preprocessing on the data, we can save a lot on optimizing the model, hyperparamters, and trying to generalize the problem because of the diversity of the data. The preprocessing consists of different actions on the data which has been explained briefly in below.

4.1 Normalization

In first step, we normalize the data which consists on removing all non-alphanumeric characters, removing all single characters, substituting multiple spaces with single space, and lower casing the words. There are also other steps for normalization but as mentioned previously, since we're not dealing with a noisy data, we can skip them.

4.2 Tokenization

In this step, we remove the punctuation which consists of `'!()-[]{};:'"\,<>./?@#$%^&*_~` symbols. After that we remove the stop words (such as "a", "an", "the", "of", etc). After tokenizing and removing punctuation and stop words we pass the data into the last step

4.3 Lemmatization

Lemmatization is a linguistic term that means grouping together words with the same root or lemma but with different inflections or derivatives

=pt

of meaning so they can be analyzed as one item. The aim is to take away suffixes and prefixes to bring out the word's dictionary form. (such as hiking, hike, hiked, which all have the same lemma 'hike')
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.

5 Hyperparameters

One of the key challenges 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.

Hyperparameter	Value
Learning Rate	0.04
Batch Size	32
Number of Epochs	20
Number of Layers	1
Dropout Rate	0.2

Table 1: Hyperparameters

6 Model Architecture

For the task, I've used different models from MLP, GRU (Chung et al., 2014) to LSTM (Hochreiter and Schmidhuber, 1997) and bidirectional LSTM (Huang et al., 2015). The best performance was achieved by the biLSTM 1 hidden layer model. The model 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.

6.1 Multi-Layer Perception Model

MLP [1] 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.

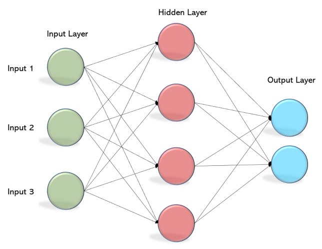

Figure 1: MLP Architecture

6.2 Bi-LSTM Model

LSTM [2] 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 which is shown on figure 2. 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.

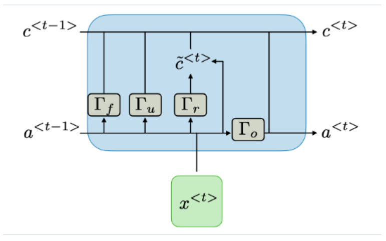

Figure 2: LSTM cell architecture

The advantage of using a bidirectional LSTM 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.

[1]Multi-Layer Perception

[2]Long Short-Term Memory

=pt

6.3 GRU

GRU [3] are very similar to Long Short Term Memory(LSTM). Just like LSTM, GRU uses gates to control the flow of information. They are relatively new as compared to LSTM. This is the reason they offer some improvement over them and have simpler architecture.

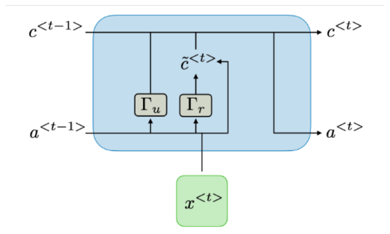

Figure 3: GRU cell architecture

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 tags.

7 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.

7.1 Hyperparameters

For learning rate, a too high learning rate will make the learning jump over minima but a too low learning rate will either take too long to converge or get stuck in an undesirable local minimum. I've tested `learning_rate` from `0.01` to `0.004` and the best value is `0.01`. A sample comparison is shown for to of them in table 2.
For batch size, 32 here is pretty standard and based on the size of our dataset, it's good choice to use. we can use 16 as well here.for the right learning rate, batch size 16 and 32 doesn't have much difference but 32 is a little bit higher.

For number of layers in this experiment, I tried different values from 1 to 3 hidden layers. The 3 layers had slightly better performance than 2 hidden layers and in the end we used 3 hidden layers for the model.

[3]Gated Recurrent Unit

Learning rate and batch size	F1 Score
lr=0.01, layers=3	0.9789
lr=0.004, layers=3	0.9614
lr=0.01, layers=2	0.9772
lr=0.004, layers=2	0.9541

Table 2: learning rate and batch size influence on score

For epochs, again since we're not handling a big data and a complex model architecture, 10 would suffice, but 20 can also be used but may cause the data to over-fit.

7.2 Optimizer

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.

7.3 Number of Hidden layers

When increasing the number of layers, the model is able to capture more complex relations. However, increasing the number of layers too much, 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 3 hidden layer.

7.4 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.

7.5 Model Architectures

As mentioned for model, I tried different models from LSTM, GRU, and MLP but MLP didn't had

=pt

Number of Layers	F1 Score
LSTM 3 layer	0.9689
Bi-LSTM 3 layers	0.9789
GRU 2 layer	0.9732
GRU 1 layer	0.9711

Table 3: Different architectures and their F1 scores

Dropout Rate	F1 Score
without	0.9371
0.2	0.9789
0.3	0.9583
0.5	0.8763

Table 4: Dropout influence on the score

very good result, and so I removed most of the code for it since it wasn't promising and right now the only legacy of it in the code is the architecture of the MLP model. Other than this, I used GRU and LSTM both in bidirectional and non-bidirectional modes.

8 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 0.9789. This is a good score for a model that was trained on a small dataset without very intensive hyperparameter tuning and model optimization.

By using F1-Score as the evaluation metric, the model is able to achieve a score of **0.9789**. This is a good score for a model that was trained on a small dataset without very intensive hyperparameter tuning and model optimization.

The result of using dropout on the model is shown on table 3. As mentioned in section 7.5, 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=0.2`, the accuracy of the result improves but if we increase dropout rate to 0.5, the accuracy drops very quickly and noticeable.

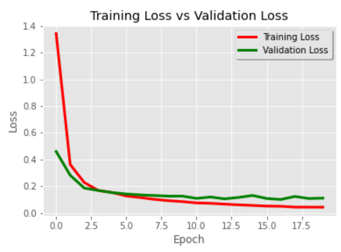

Figure 4: Training Loss vs Validation Loss

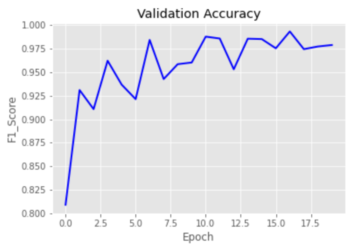

Figure 5: F1 score / Epoch

As shown in 4, we can see the training vs validation loss over the training time. and as shown in 5, we can see the F1 score over the 10 epochs.

References

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555.

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