# NLP-243: Slot Tagging for Natural Language Utterances using PyTorch

**Kushagra Seth**
University of California, Santa Cruz
kuseth@ucsc.edu

## Abstract

The main objective of this model is to train a Long-Short Term Memory(LSTM) Network to tag slots in natural language utterances of users. 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 in addition to the user's intention to find out about movies, and issue a query to the back-end knowledge graph to get information for formulating a response back to the user. In order to train a model to predict the corresponding relations for a given speech, a training set of utterances and a set of IOB Tags have been provided.

I have trained a deep neural network model using PyTorch that outputs the related set of IOB Tags when given a new utterance.

## 1 Introduction

We can execute supervised learning on data that has labels(tags in this case). Basically, supervised text classification means that we have a set of instances for which we already know the labels or the ground truth.
In this case, there are multiple labels possible for each word in an utterance. My model will attempt to learn input, target pairs from this data set.

Here is an *example* utterance from the data set:

```
show me movies directed by Woody Allen recently.
```

There are two slots in this utterance:

**director** = *Woody Allen*
**release_year** = *recently*

The correct tags in IOB representation for this utterance would then be:

```
show me movies directed by Woody       Allen       recently.
0    0  0      0        0  B_director  I_director  B_release_year
```

This particular data set with which we have been provided with has 27 different tags/labels possible for each word in an utterance.

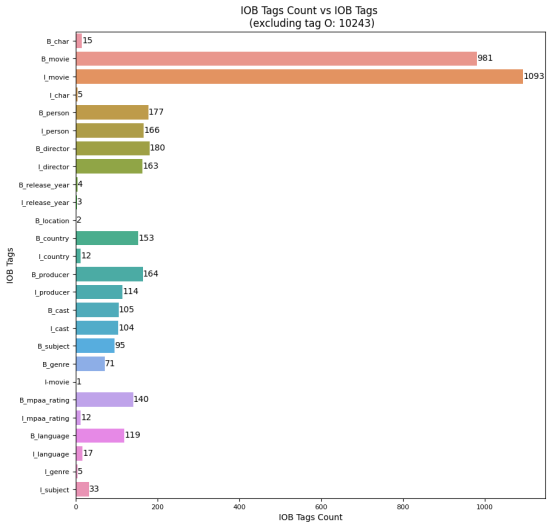


Figure 1: Horizontal Bar Plot that represents the count of each tag in the data set

## 2 Proposed Framework

### 2.1 Encoding Utterances using Word Embeddings

Word Embeddings is an approach to provide a dense vector representation of words to capture something about their meaning. Each word is represented by a point in the vector space and points are learned and moved based on the words that surround them. This allows word embeddings to learn something more about the meaning of the words. Similar meanings are locally clustered within the vector space. Each vector is represented in tens or

hundreds of dimensions.

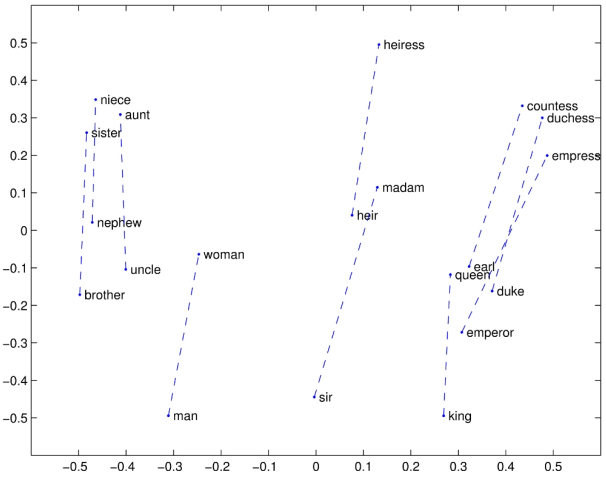


Figure 2: GloVe [2]

I have used **pre-trained GloVe Embeddings** for this problem. I have created a vocabulary in the form of a dictionary from words present in GloVe. Each word in the vocab is represented in 300 dimensions.
I have added embedding representation for 2 tokens namely $\langle pad \rangle and \langle unk \rangle$.

Padding is a pre-requisite for matrix multiplication. I have made sure that all sentences in a batch are of equal length by using $\langle pad \rangle$.

All words which are not present in the vocab are represented using token $\langle unk \rangle$.

```
['<pad>' '<unk>' ',' '.' 'the']
[[ 0.          0.          0.         ...  0.          0.
   0.        ]
 [ 0.22418656 -0.28881998  0.13854443 ...  0.19310581 -0.07767605
  -0.1448164 ]
 [-0.082752    0.67204    -0.14987    ... -0.1918     -0.37846
  -0.06589   ]
 [ 0.012001    0.20751    -0.12578    ...  0.13871    -0.36049
  -0.035     ]
 [ 0.27204    -0.06203    -0.1884     ...  0.13015    -0.18317
   0.1323    ]]
```

Figure 3: Word Embedding Representation of first 5 words in the vocab

### 2.2 Encoding IOB Tags by using Enumerated Dictionary Id's

A classification task with more than two classes is called Multi-class classification. It makes the assumption that each word in an utterance is assigned to one and only one label.
There are 27 possible labels for each word in an utterance. I have created a dictionary that represents the different possible tags for each word.

There is one tag for each word in an utterance, i.e., the number of words and tags for an utterance should be equal.

tag2idx dictionary: { B_language: 0, I_cast: 1, I_mpaa_rating: 2, I_char: 3, B_genre: 4, I_country: 5, B_release_year: 6, B_producer: 7, B_person: 8, B_cast: 9, B_mpaa_rating: 10, I_director: 11, I_release_year: 12, I_subject: 13, B_country: 14, I_movie: 15, I_person: 16, I_genre: 17, I_language: 18, O: 19, B_location: 20, I_producer: 21, B_char: 22, I-movie: 23, B_movie: 24, B_director: 25, B_subject: 26 }

### 2.3 Long-Short Term Memory Network (LSTM)

Long Short Term Memory Networks, most commonly referred to as "LSTMs," are a unique class of RNN that can recognize long-term dependencies. LSTMs are explicitly designed to avoid the long-term dependency problem. They don't struggle to learn; rather, remembering information for extended periods of time is basically their default behavior. All recurrent neural networks have the shape of a series of neural network modules that repeat. This recurring module in typical RNNs will be made up of just one tanh layer. Although the repeating module of LSTMs also has a chain-like topology, it is structured differently. There are four neural network layers instead of just one, and they interact in a very unique way.

### 2.4 Bi-Directional LSTM

Bidirectional Long-Short Term Memory(Bi-LSTM) is the process of making any neural network to have the sequence information in both directions backwards (future to past) or forward (past to future).

The Bi-LSTM differs from the conventional LSTM in that our input flows in two directions. With the regular LSTM, we can make input flow in one direction, either backwards or forward. However, in Bidirectional LSTM, we can make the input flow in both directions to preserve the future and the past information. BI-LSTM is usually employed where the sequence to sequence tasks are needed.

## 3 Experiments

I came up with a Baseline model with certain Hyper Parameters. Then, I have tuned the model by varying one of these parameters to increase the accuracy of this LSTM based model. I got the highest accuracy with the optimizer: AdamW while the

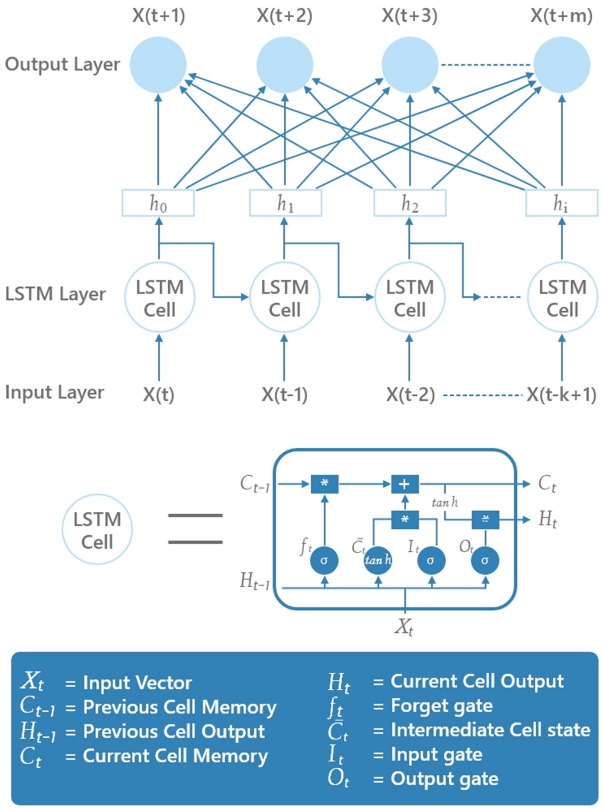


Figure 4: LSTM Model [3]

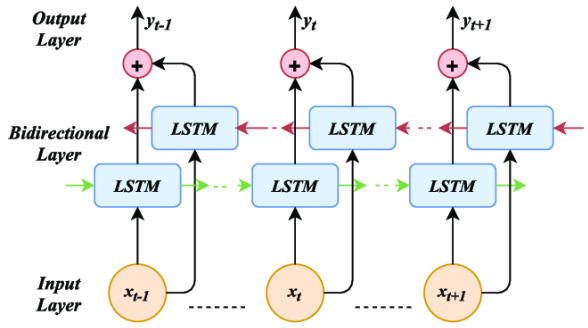


Figure 5: Bi-Directional LSTM Model [1]

performance with other optimizers such as SGD and AdaGrad was way worse and could'nt even crack 0.5 F1_Score.

### 3.1 Experiment-1: Varying the Learning Rate

The *learning rate* or *step size* refers to how frequently the weights are updated during training.

In particular, the learning rate is a hyper-parameter that can be customized and is used to train neural networks. Its value is typically small and positive, falling between 0.0 and 1.0.

How quickly the model adapts to the challenge is determined by the learning rate. Given the smaller changes to the weights made with each update, smaller learning rates necessitate more training epochs, whereas bigger learning rates produce quick changes and necessitate fewer training epochs.

A model may converge to a less-than-ideal solution too soon if the learning rate is too high, whereas the opposite is true if the learning rate is too low.

Table 1: Fixed Hyper Parameters

| Hyper Parameters | Values |
|---|---|
| Batch Size | 32 |
| Hidden Dimensions | 20 |
| Number of LSTM Layers | 2 |
| Epochs | 25 |

Table 2: Varying the Learning Rate Parameter

| **Learning Rate** | 0.001 | 0.04 | **0.01** |
|---|---|---|---|
| **F1_Score** | 0.95 | 0.9705 | ***0.9724*** |

### 3.2 Experiment-2: Varying the Batch Size

Different optimization techniques are used to train neural networks.
In this case, a prediction is made based on the model's current state, and the difference between the predicted and the actual values is used to estimate the error gradient. The model weights are then updated using this error gradient, and the procedure is repeated. The *batch size*, often known as the *batch*, is a hyper-parameter for the learning algorithm that determines how many training samples are utilized to estimate the error gradient.

Table 3: Fixed Hyper Parameters

| Hyper Parameters | Values |
|---|---|
| Learning Rate | 0.01 |
| Hidden Dimensions | 20 |
| Number of LSTM Layers | 2 |
| Epochs | 25 |

Table 4: Varying the Batch Size Parameter

| **Batch Size** | 16 | **32** | 64 |
|---|---|---|---|
| **F1_Score** | 0.965 | ***0.9724*** | 0.935 |

### 3.3 Experiment-3: Varying the Hidden Dimensions

Hidden dimension determines the feature vector size of the h_n (hidden state). At each timestep,

lstm will take a h_n and input.
I have used multiple variations of hidden dimensions to get the maximum accuracy on the validation set.

Table 5: Fixed Hyper Parameters

| Hyper Parameters | Values |
|---|---|
| Learning Rate | 0.01 |
| Batch Size | 32 |
| Number of LSTM Layers | 2 |
| Epochs | 25 |

Table 6: Varying the Hidden Dimensions Parameter

| **Hidden Dimensions** | 10 | **20** | 50 |
|---|---|---|---|
| **F1_Score** | 0.9712 | ***0.9724*** | 0.952 |

### 3.4 Experiment-4: Varying the Epochs Parameter

In machine learning, an epoch refers to one full iteration of the algorithm over the training data set. An essential hyper-parameter for the algorithm is this epoch's number.

The amount of complete runs of the whole training data set during the algorithm's training or learning process is specified in terms of epochs. The internal model parameters of the data set are modified at each epoch.

Table 7: Fixed Hyper Parameters

| Hyper Parameters | Values |
|---|---|
| Learning Rate | 0.01 |
| Batch Size | 32 |
| Number of LSTM Layers | 2 |
| Hidden Dimensions | 20 |

Table 8: Varying the Epochs

| **Epochs** | 25 | 50 | **100** |
|---|---|---|---|
| **F1_Score** | 0.9724 | 0.9774 | ***0.9812*** |

### 3.5 Experiment-5: Varying Fully Connected Layers

I have used a Feed-Forward Network on the output of the LSTM layer.
I have tried two configurations of this feed-forward network with 1 and 2 fully connected layers.

Table 9: Fixed Hyper Parameters

| Hyper Parameters | Values |
|---|---|
| 1 Fully Connected Layer | 0.9812 |
| **2 Fully Connected Layers** | ***0.9825*** |

### 3.6 Experiment-6: Using Bi-LSTM

Using Bi-LSTM we can make the input flow in both directions to preserve the future and the past information which provided a higher F1_Score on this data set.

Table 10: Fixed Hyper Parameters

| Hyper Parameters | Values |
|---|---|
| LSTM | 0.9825 |
| **Bi-LSTM** | ***0.9835*** |

### 3.7 Experiment-7: Varying Dropout

I tried different values of Dropout in range 0.2 - 0.5 but got the best results by fixing value of dropout to 0.2. I have tried applying Dropout before LSTM Layer and before the output layer.

Table 11: Varying Dropout

| Dropout | F1_Score |
|---|---|
| Dropout before LSTM Layer | 0.9791 |
| **Dropout before O/P Layer** | ***0.9835*** |
| Dropout before LSTM and O/P Layers | 0.9822 |

### 3.8 Experiment-8: Varying the Optimizers

I tried using SGD and Adagrad for this model but the performance was alot worse than that compared to Adam. The Accuracy with the these two optimizers didn't even crack 0.6 F1_Score.
I have used different versions of Adam provided in the PyTorch library namely Adamax and AdamW.

Table 12: Fixed Hyper Parameters

| Hyper Parameters | Values |
|---|---|
| Adam | 0.9835 |
| Adamax | 0.9901 |
| **AdamW** | ***0.9925*** |

## 4 Results and Conclusion

In this project, I have addressed the task of Multi-Class Classification on the data set containing Utterances and the corresponding IOB Tags for each word in the utterance.
I created a Bi-LSTM model using PyTorch in python for this data set and evaluated its performance by tuning different hyper-parameters.

### 4.1 F1_Score

**Val F1_Score** = *0.9925*
**Test F1_Score** = *0.808*

Table 13: Hyper Parameters Configuration that produced highest F1_Score

| Hyper Parameters | Values |
|---|---|
| Type of LSTM | Bi-LSTM |
| Batch Size | 32 |
| Epochs | 100 |
| Hidden Dimensions | 20 |
| Learning Rate | 0.01 |
| Fully Connected Layers | 2 |
| Dropout before O/P Layer | 0.2 |
| Optimizer | AdamW |

F1_Score is calculated using the below-mentioned formula:

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Figure 6: F1_Score Formula

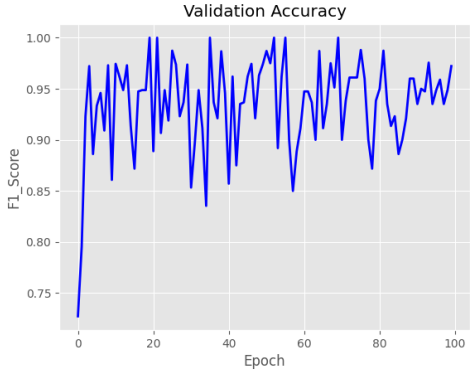


Figure 7: F1_Score vs Epoch Graph

### 4.2 Average Loss

The training loss indicates how well the model is fitting the training data, while the validation loss indicates how well the model fits new data.

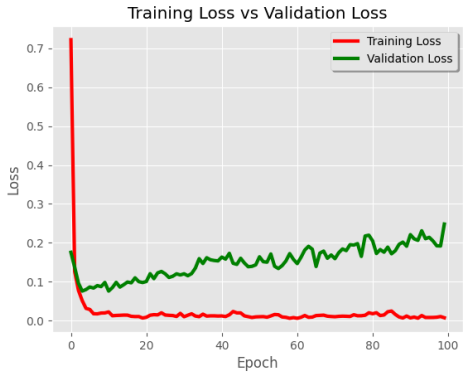


Figure 8: Visualization of Training and Validation Loss

## 5 Future Work

I analyzed the data set and found that the data is highly imbalanced as it contains large number of 'O', 'B_movie' and 'I_movie' labels as compared to the other 24 labels. I can perhaps try to improve the F1_Score by Over or Under sampling the data set to get a better F1_Score.

## Acknowledgments

I would like to thank Nilay Patel (Teaching Assistant for the this course, i.e., NLP-243) who provided me with constant support and help during this project.

## References

[1] https://www.researchgate.net/publication/344554659_A_Deep_Learning_Approach_for_Human_Activities_Recognition_

[2] https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0262708

[3] https://nlp.stanford.edu/projects/glove/