NLP 243 Assignment 2 - Slot Tagging with Natural Language Utterances

Ishika Kulkarni
ikulkar1@ucsc.edu

Abstract

This assignment addresses slot tagging for natural language queries focusing on slot extraction like movies, directors, and genres on a sample dataset containing IDs, utterances, and IOB (Inside, Outside, beginning) Slot Tags. The goal is to understand the importance of sequential models and contextual embeddings in advancing slot tagging.

1 Introduction

In this supervised learning task, we aim to train a model that predicts sequential labels for a given input string, which is a part of Named Entity Recognition (NER). These sequential labels are also called slots, and the task is slot tagging. This is very useful in virtual personal assistants or recommender systems.

Slots equip us with crucial information that users provide, and tagging them helps us generate accurate responses. For example, 'Twilight' - the movie in a query would be tagged as 'B_Movie.' As mentioned in the abstract, this type of slot tagging is called IOB slot tagging. These entities correspond to slots that need to be identified.

Consider another example: B_director would be the beginning of the director's name, and I_director would be the continuation.

We will use models like RNNs, LSTMs, BiLSTMs, and embeddings like GloVe, Bag of Words, etc. We will use normalization, hyperparameter tuning, and others to ensure the model is not generalized or overfitted.

We have the SeqEval library for evaluation since this is a sequence-to-sequence task.

2 Dataset

2.1 Training Data

The training CSV contains three columns - ID, Utterances, and IOB Slot Tags.

Column	Description
id	A unique identifier
utterances	Query requiring slot tagging
IOB Slot Tags	The sequence of IOB labels

Table 1: Description of the Train Dataset Columns

	ID	utterances	IOB Slot tags
0	1	who plays luke on star wars new hope	O O B_char O B_movie I_movie I_movie I_movie
1	2	show credits for the godfather	O O O B_movie I_movie
2	3	who was the main actor in the exorcist	O O O O O O B_movie I_movie
3	4	find the female actress from the movie she 's ...	O O O O O O O B_movie I_movie I_movie I_movie
4	5	who played dory on finding nemo	O O B_char O B_movie I_movie
5	6	who was the female lead in resident evil	O O O O O O B_movie I_movie
6	7	who played guido in life is beautiful	O O B_char O B_movie I_movie I_movie
7	8	who was the co-star in shoot to kill	O O O O O B_movie I_movie I_movie
8	10	cast and crew of movie the campaign	O O O O O B_movie I_movie
9	11	cast and crew of the campaign	O O O O B_movie I_movie

Figure 1: Train Dataset

To understand the data, we need to understand the distribution of tags. We can see that the data is very unbalanced, and the frequency distribution of the tags is as follows:

2.2 Test Data

The test CSV contains only two columns - ID and utterance.

2.3 Submission CSV

The submission CSV, the output file, contains two columns: ID and IOB Slot Tags.

IOB Slot Tag	Frequency
O	10,428
I_movie	1,132
B_movie	1,009
B_person	193
B_director	184
I_person	174
I_director	167
B_producer	164
B_country	153
B_mpaa_rating	141
B_language	117
I_producer	114
B_cast	105
I_cast	104
B_subject	95
B_genre	70
I_subject	33
I_language	17
B_char	14
I_country	12
I_mpaa_rating	12
I_char	5
B_release_year	5
I_genre	5
I_release_year	3
B_location	2
I-movie	1

Table 2: Frequencies of IOB Slot Tags in the Dataset

	ID	utterances
0	1	star of thor
1	2	who is in the movie the campaign
2	3	list the cast of the movie the campaign
3	4	who was in twilight
4	5	who is in vulguria
5	6	actor from lost
6	7	who played in the movie rocky
7	8	who played in the movie captain america
8	9	cast and crew for in july
9	10	who is in movie in july

Figure 2: Test Dataset

Column	Description
id	A unique identifier
utterances	Query requiring slot tagging

Table 3: Description of the Test Dataset Columns

Column	Description
ID	A unique identifier
IOB Slot Tags	Predicted sequence of IOB labels

Table 4: Submission CSV

3 Implementation and Result

Sequence tagging involves assigning labels to tokens in a sequence. Thus, we need a model that is aware of context and can handle sequences of varying lengths and capture dependencies. Therefore, we use LSTMs (Long Short-Term Memory Networks).

LSTMs are the types of RNNs (Recurrent Neural Networks) that capture dependencies for sequential data and work better than them regarding the vanishing gradient problem.

A slightly upgraded version of LSTMs is BiLSTMs (BiDirectional LSTMs), which we use for this assignment. While LSTMs process inputs sequentially, BiLSTMs do it both ways. If any relation or context is missed unidirectionally, it can capture it with the proceeding and the succeeding word.

Let us consider an example where the utterance is 'What was the budget for The Avengers'. An RNN would process this sentence left to right, but it might not retain information that it captured at 'what' by the time it processes 'for.' Let us now consider an LSTM. It would also process it left to right and tag 'budget' right, but it may tag 'the Avengers' right. It might consider 'the' as '0' and not 'B_movie.' However, this would not be the case in BiLSTMs; it would capture the relation both ways and thus provide a richer understanding of the query.

3.1 Exploratory Phase / Model Selection

Before starting with a baseline approach, we assessed the accuracies and performances of various models on the validation set to see how it goes, how it can be improved further, and which embedding can be used with what model and how to tune it. This helps us understand potential improvements and suitable combinations for optimal results.

Based on the experiments and analysis, we chose BiLSTMs with GloVe embeddings as they provided the best performance, achieving an accuracy of 0.93 without requiring extensive hyperparameter tuning. In contrast, RNNs and CNNs did not perform as well. While RNNs could potentially improve with tuning, they needed three layers to handle this task, making them less efficient than BiLSTMs. Other techniques, like word-based tokenization and FastText embeddings, did not yield significant improvements. Thus, combining BiLSTM and GloVe proved to be the most effective for this task.

Model	**Embedding**	**Layers**	**Data Split**	**Hidden Dimensions**	**Accuracy**	**Notes**
RNN	GloVe	2	80-20	64	0.89	-
CNN	GloVe	1	80-20	64	0.65	-
RNN	GloVe	3	90-10	128	0.90	Grid search for tuning
BiLSTM	GloVe	2	90-10	64	0.93	-
BiLSTM	FastText	1	90-10	64	0.63	Added word-based tokenization

Table 5: Model Configurations and Performance

3.2 Base Model Architecture

- **Dataset Preparation:**
  - The input dataset consists of sentences (utterances) and their corresponding IOB tags like I_movie, B_director, O.
  - The data is split into training and validation sets.
- **Tokenization:**
  - Each sentence is split into individual words using simple tokens.
- **Embedding Layer:**
  - The model uses GloVe embeddings, which map each word to a 300-dimensional vector. If a word is not found in the embeddings, it is assigned a random vector.
- **Encoding Tags:**
  - The named entity tags are converted to numerical labels using a tag-to-index dictionary to process and predict tags.
- **Padding:**
  - Sentences of varying lengths are padded with a special token (<pad>) to make them equal in length for consistent batch processing.
- **BiLSTM Architecture:**
  - The core of the base model is a BiLSTM.
- **Output Layer:**

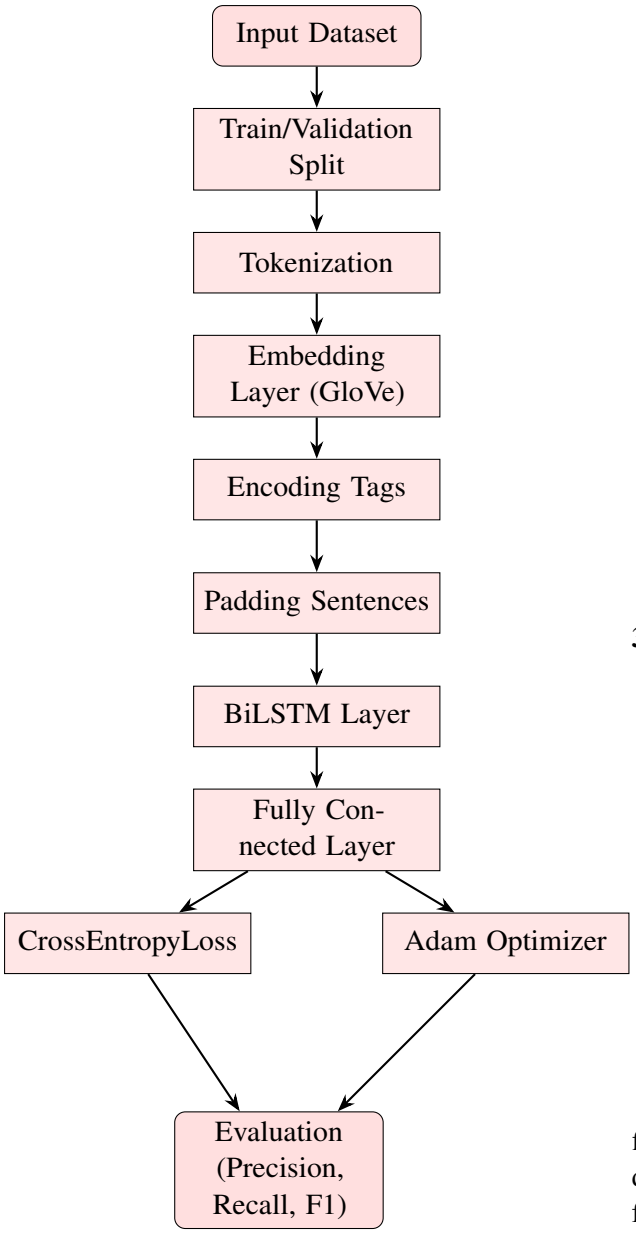

Figure 3: UML Diagram of the BiLSTM Model with GloVe Embeddings

  - A fully connected layer maps the LSTM output to the desired number of tags.

- **Loss Function:**
  - The model uses CrossEntropyLoss, which calculates the difference between the predicted and actual tags for each word in a sentence.

- **Training Process:**
  - The model is trained using an Adam optimizer.

- **Evaluation:**
  - The model's performance is evaluated using metrics like precision, recall, and F1 score on the validation set. The predicted tags are compared against the actual tags to compute these metrics.

- **Summary:**
  - In essence, the base model uses a BiLSTM network with GloVe embeddings to perform slot tagging, focusing on capturing each word's context through bidirectional processing.
  - The use of padding and a simple dense output layer makes it a straightforward implementation without advanced features like batch normalization or dropout.

3.3 Approach 1

Parameter	Value
Train-Test Split	90%-10%
Batch Size	32
Hidden Layer Dimension	64
Epochs	10
Layers in Model	2
Optimizer	Adam
Learning Rate	0.001

Table 6: Values for Approach 1

In this approach with BiLSTMs and GloVe, we first load and understand the data, followed by downloading and extracting the GloVe embeddings for word representation. Once our embedding is ready, we map them to their corresponding vectors in a dictionary.

We then use a vocabulary checker class, which takes the previous embeddings and handles padding and unknown values for our vectors. It generates a word-to-index and an index-to-word dictionary for slot tagging.

Next, another class will handle the IOB tagging format once we finish the above computation. This class works explicitly with mapping the word to its corresponding tag and mapping it back.

For the main computation, we use BiLSTM with layers, and then the output is passed through a fully connected layer to get predictions for each word in a sentence.

Continuing, we process 90% data for training, 10% for validation, and the batch size to process is 32. Next up, inside the model, we have a hidden

layer of 64 units that runs for ten epochs. Further, we have an Adam optimizer for the parameters.

To evaluate the model, we use cross-entropy loss as this is a tag classification problem. Moreover, we check the proper and predicted tags to see how our model performs. Here, we use a tag sequencer to convert the predicted and true vectors to their corresponding tags for better understanding.

The loss over ten epochs is as follows:

Epoch	Cross Entropy Loss
1	2.1885
2	1.8438
3	1.7117
4	1.7242
5	1.2628
6	1.1003
7	1.0723
8	1.3958
9	1.0699
10	0.9878

Table 7: Epoch vs Loss

The steadily decreases for the epochs; however, several values are still too close.

The classification report is as follows:

Class	Precision	Recall	F1-Score	Support
_cast	0.44	0.50	0.47	8
_char	1.00	1.00	1.00	1
_country	0.83	0.83	0.83	12
_director	0.74	0.93	0.82	15
_genre	0.78	0.70	0.74	10
_language	0.79	0.92	0.85	12
_movie	0.92	0.93	0.92	116
_mpaa_rating	1.00	1.00	1.00	13
_person	0.73	0.65	0.69	17
_producer	0.83	0.56	0.67	18
_release_year	0.00	0.00	0.00	1
_subject	0.69	0.82	0.75	11
Micro avg	0.84	0.85	0.84	234
Macro avg	0.73	0.74	0.73	234
Weighted avg	0.84	0.85	0.84	234

Table 8: Classification Report for Approach 1

Once we are done with this, we get the submission CSV. For this particular approach, the Kaggle score was 0.80020.

3.4 Approach 2

Approach 2 is quite similar to approach one regarding data processing and handling.

This approach received a score of 0.81388 on the Kaggle leaderboard.

Parameter	Value
Train-Test Split	95%05%
Batch Size	16
Hidden Dimensions	256
Number of Epochs	30
Number of Layers in Model	2
Optimizer	Adam
Learning Rate	0.0001

Table 9: Model Parameters and Hyperparameters

The key changes are in the model and how we decide to tune it. They are listed as follows:

- **Train-Test Validation Split:**
  - The dataset is split into training and validation sets, ensuring that the model is evaluated on data it has not seen during training. Here, the data is split 95%-5%, which gives us more training data; thus, the data would not overfit with a small dataset.
- **GloVe Embeddings:**
  - For GloVe, the embeddings were not frozen in the earlier approach, leading to a high computation time.
- **Training Stability:**
  - Comprises dropout and batch normalization to stabilize training and accelerate convergence.
- **Loss Function and Training Setup:**
  - Uses a lower learning rate (0.0001) and includes dropout and batch normalization, making the training process more stable and controlled.
- **Batch Normalization:**
  - Introduces batch normalization after the LSTM layer to stabilize training and improve generalization.
- **Model Complexity:**
  - A more complex model includes LSTM, batch normalization, and dropout, enhancing generalization and preventing overfitting.
- **Evaluation Metrics:**

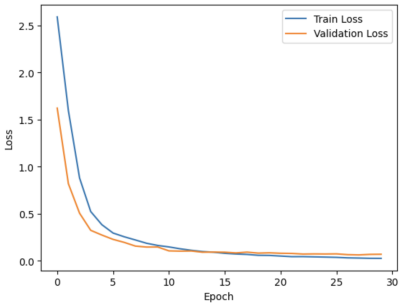

Figure 4: Train vs Validation Loss

– Computes precision, recall, and F1 score, providing a clearer understanding of the model's performance.

Attached is the performance for the same :

Epoch	Train Loss	Precision	Recall	F1 Score	Validation Loss
1	2.5918	0.1938	0.4386	0.2688	1.6230
2	1.5967	0.3631	0.5702	0.4437	0.8201
3	0.8813	0.5310	0.6754	0.5946	0.5070
4	0.5240	0.6667	0.7368	0.7000	0.3250
5	0.3844	0.6165	0.7193	0.6640	0.2746
6	0.2961	0.7213	0.7719	0.7458	0.2288
7	0.2563	0.7339	0.7982	0.7647	0.1966
8	0.2223	0.7438	0.7895	0.7660	0.1575
9	0.1874	0.8362	0.8509	0.8435	0.1472
10	0.1641	0.7851	0.8333	0.8085	0.1475
11	0.1485	0.8661	0.8509	0.8584	0.1067
12	0.1289	0.8957	0.9035	0.8996	0.1039
13	0.1125	0.8621	0.8772	0.8696	0.1054
14	0.0991	0.8696	0.8772	0.8734	0.0923
15	0.0924	0.8684	0.8684	0.8684	0.0938
16	0.0807	0.8938	0.8860	0.8899	0.0924
17	0.0734	0.8761	0.8684	0.8722	0.0850
18	0.0689	0.8870	0.8947	0.8908	0.0932
19	0.0594	0.8870	0.8947	0.8908	0.0828
20	0.0582	0.8870	0.8947	0.8908	0.0858
21	0.0517	0.9035	0.9035	0.9035	0.0811
22	0.0450	0.8957	0.9035	0.8996	0.0794
23	0.0454	0.9211	0.9211	0.9211	0.0728
24	0.0429	0.9123	0.9123	0.9123	0.0744
25	0.0405	0.9035	0.9035	0.9035	0.0735
26	0.0377	0.9204	0.9123	0.9163	0.0748
27	0.0323	0.9123	0.9123	0.9123	0.0663
28	0.0302	0.8898	0.9211	0.9052	0.0636
29	0.0274	0.9043	0.9123	0.9083	0.0691
30	0.0270	0.9123	0.9123	0.9123	0.0705

Table 10: Training and Validation Metrics for Each Epoch

4 Conclusion

Based on the above results, we can conclude that BiLSTMs work the best compared to CNNs, RNNs, and LSTMs regarding sequential slot tagging.

References

[1] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional LSTM-CRF Models for Sequence Tagging. *arXiv preprint arXiv:1508.01991*, 2015. https://arxiv.org/abs/1508.01991.

[2] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. https://aclanthology.org/D14-1162.

[3] Bing Liu and Ian Lane. Joint Online Spoken Language Understanding and Language Modeling with Recurrent Neural Networks. https://arxiv.org/abs/1609.01462.

[4] Sebastian Ruder. SeqEval: A Python library for sequence labeling evaluation. 2017. https://github.com/chakki-works/seqeval.

[5] Andrej Karpathy. The Unreasonable Effectiveness of Recurrent Neural Networks. *Blog post*, 2015. https://karpathy.github.io/2015/05/21/rnn-effectiveness/.