Slot Tagging for Natural Language Utterances

Shannon Rumsey

Abstract

This paper addresses IOB slot tag classification using two sequential models: A bidirectional LSTM and its simpler counterpart, the bidirectional GRU model. The hyperparameters of the two models are tailored in such a way to get better validation loss and F1 scores.

1 Introduction

1.1 Task

The goal is to assign IOB slot tags to movie-related utterances. IOB is an acronym for "inside-outside-beginning" and is a method of tagging tokens in a sentence. Most words will be labeled with "O", to denote being outside of a relation category, but for words that are related under a specific category (e.g., both referencing the same director), the first word would be tagged with B (for beginning), followed by an underscore and the category (e.g., B_director), and the remaining related words would be tagged with I (for inside), also followed by and underscore and the category. An example of this tagging schema is the utterance "show me movies by Woody Allen recently" which would have the tags "O O O O B_director I_director B_release_year". For this task, the IOB tag and the movie-relation tag, are kept together instead of classified separately. The tagging classification can be thought of as a three-fold process: the first being to identify entities (e.g. Woody Allen), the second to find what movie relation the entity is associated with (e.g., director), and the third is to determine the word's relative relationship to the words around it (e.g., Woody is the beginning of the director's name, Allen is the inner portion of the director's name). This way of tagging will be used on the movie-related utterances as a way of determining the kind of query issued to a knowledge-graph.

1.2 Dataset

The training dataset used for this task is a CSV file of movie related utterances that a user might ask a virtual assistant along with their respective IOB slot tags, making this a supervised classification task. The test dataset contains only utterances, as the model built should identify slot tags. While there are many possible IOB slot tags, some are more frequent than others. The tag "O O B_movie I_movie" occurs 60 times in the training dataset, making it the most frequent of all tag combinations. Some tag combinations occur between 50 and 10 times, while others occur only once or twice.

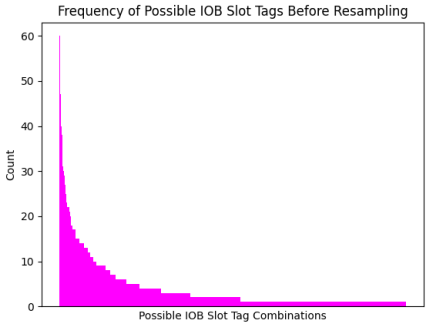

Figure 1: Distribution of the different IOB combinations in the training dataset.

Individual IOB slot tags also appears at different frequencies. The "O" tag occurs $10,428$ times in the dataset, whereas "B_location" only occurs twice (see Table 1 to see more frequencies).

2 Models

Two gated-sequential model architectures were explored for this classification task. This type of architecture was chosen for its long term memorization, which allows for movie-related utterances to be fed into a model and tagged sequentially without losing information found earlier in the utterance. First, a bidirectional LSTM model is explored. Then, a

Tag	Count
O	10,428
I_movie	1,133
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

Table 1: IOB Slot Tag Counts

bidirectional GRU model is created to determine if this simplified version offers comparable or superior performance.

2.1 Embeddings

Instead of using pre-trained static embeddings like GloVe or Word2Vec, both models used learned-dynamic embeddings initialized and stored by PyTorch's `nn.Embedding()` class. This embedding method was chosen due to its superior performance when compared to static embeddings (Wang et al., 2020). Dynamic embeddings allow for the use of context when deriving word vectors, meaning that they can distinguish different meanings for the same word (polysemy) (Hofmann et al., 2021). This is especially important in movie-related utterances where words such as "star" should mainly be associated with actors, not astronomy.

2.2 BiLSTM Model

This model was inspired by the work of Gupta et al. (2019) which found that a bidirectional LSTM model showed excellent performance, even when compared to the performance of a CNN. The number of epochs that the model in this task runs on is based on convergence, which is determined by the validation loss and accuracy plateauing. At the epoch with the lowest validation loss and highest validation accuracy, the model is saved and only updated if there is better performance on a future epoch. LSTMs are a better version of the typical RNN model because they are better at capturing long range dependencies and complex sequences. While the utterances for this task are not necessarily long, the class categories being a combination of relations and IOB slot tags may make learning relationships complex, rendering LSTMs a suitable choice. The bidirectionality of this model is an added benefit because it allows the model to learn past and future relationships, which may be helpful for determining cases where there is a beginning and inner IOB tag.

The model is implemented using PyTorch's `nn.LSTM`, is preceded by an embeddings layer, and proceeded by a linear feed-forward layer. The hidden size is doubled to be compatible for bidirectionality. The hyperparameters experimented with are the learning rate, number of layers, hidden size, embedding size, and dropout.

2.3 BiGRU Model

Inspired by Korpusik et al. (2019), the second model created is a bidirectional GRU. This model is very similar to the BiLSTM one in that it is bidirectional and is implemented in PyTorch. The only difference between the two models is the PyTorch class used (`nn.LSTM` and `nn.GRU`). Like the BiLSTM model, there is an embeddings layer before the GRU model and a feed-forward layer after it. A GRU model is very similar to an LSTM in that is also has a gating mechanism, however, the GRU model has a streamlined version of the LSTM memory cell, making it more efficient. Like with the BiLSTM model, the GRU model is trained until the validation accuracy and loss plateaus and the hyperparameters observed are the learning rate, number of layers, hidden size, embedding size, and dropout.

3 Experiments

To address the tag imbalances shown in Figure 1, resampling was applied to all tag combinations that occur fewer than 20 times, accounting for most of the infrequent cases seen in the figure. Resampling was done on the minority set (frequencies less than 20) until the maximum frequency of the set matched the maximum of the majority class (frequencies more than 20), or, 60. The dataset was split into a train (80%) and validation (20%) set using Scikit-Learn's `traintestsplit`. These two datasets and the test set can be fed into the `IOBDataset` class individually. In this class, each row in the `utterances` and `IOB Slot tags` columns are converted into tensors of numbers, where the numbers are determined by simply counting them off in the order that they occur in the training dataset. This is done by iterating through all of the words and tags in the training dataset and adding them one-by-one to a dictionary. This results in two dictionaries, one for words in the utterances and one for the IOB slot tags. The keys in the dictionaries are the words or tags, and the values are the count at which the word was found in the training dataset (e.g. the first word found would have a numeric value 0). Tokens for padding (<PAD>) and unknown tokens (<UNK>) were added to the word vocabulary and only the unknown token was added to the tag vocabulary. The padding and unknown tokens start the vocabularies and have the numeric values of 0 and 1 respectively, meaning that all proceeding tokens have values 2 and onward, based on their order of occurrence in the training dataset. The sentences and tag combinations are replaced by their numeric counterparts by doing a simple lookup in the dictionaries created. A reversed version of the tag-to-numeric dictionary was created as a way to convert the model's numeric tag predictions back into tag form.

PyTorch's `DataLoader` class groups the data into batches of 64 rows to be trained and predicted on, allowing for more stable gradient updates. The default `collate_fn` parameter in this class is responsible for combining data. This parameter is modified to split the dataset into two sequences, numeric words and tags, and pad them with the <PAD> token

Both models utilize the AdamW optimizer (Loshchilov and Hutter, 2019) due to its improvements upon the original Adam optimizer. A learning rate scheduler was also incorporated to lower the learning rate when loss stops decreasing.

The hyperparameters that result in the lowest validation loss and highest weighted validation F1 score are selected, which was the same process for selecting the final model to be used to generate test predictions. The F1 score is an important distinction from accuracy because it takes into account class imbalances. While resampling was introduced to help uniformly distribute the dataset, there still exists some imbalances that need to be accounted for during evaluation. Because dropout is only applied to the layers that are not the final layer, the number of layers experimented with are more than 1.

3.1 BiLSTM Model

The baseline model had a learning rate of 0.001, 2 layers (Gupta et al., 2019), a hidden size of 512, an embedding size of 512, and no dropout. The values of the learning rate that were experimented with was 0.001 and 0.005, the number of layers was 2 and 3, dropout was 0 and 0.1, the embedding size was 512 and 256, and the hidden size was 512, 256, and 128. Again, these values were chosen through experimentation with the model, and were adjusted based on how the model reacted to such values. For example, if a lower learning rate boosted accuracy, then even lower learning rates would be explored until model performance degrades.

3.2 BiGRU Model

Hyperparameter selection for this model is almost identical to the process for the BiLSTM model, the only differences are the hyperparameter values themselves because the model reacts differently. The baseline model was the same as the one for the BiLSTM model. The learning rate values observed were 0.001 and 0.005, the number of layers were 2 and 3, the hidden sizes were 512, 256, and 1024, the embedding sizes were 1024 and 512, and lastly, the dropout was 0, 0.1, and 0.2.

4 Results

The best parameters for each model were selected based on having the highest accuracy while not having too high of a loss value. The parameters that result in the accuracy of 0.97571 in Table 2 also produce a loss that is 0.03 higher than that of the second highest accuracy, so the parameters with the second highest accuracy were chosen for the LSTM model. In Table 3, the highest accuracy has a loss

only 0.003 higher than the second-highest, so the parameters with the highest accuracy were selected for the GRU model. The GRU model with the selected parameters outperforms the LSTM model on the test dataset and was selected as the final model to make predictions on the test set, which is suprising given that it is a simplified version of the LSTM model.

4.1 BiLSTM Model

The BiLSTM model performs better with no dropout, a larger learning rate, a smaller hidden size, and a moderate embedding size. Dropout applied directly to the weights, which is what PyTorch's LSTM dropout parameter uses, degrades performance (Semeniuta et al., 2016). Instead of using dropout in the same manner as a feed forward network, it should be applied to the recurrent connections or used in other more sophisticated approaches (Semeniuta et al., 2016) (Gal and Ghahramani, 2016). A larger learning rate allows the model to learn much faster while not overfitting because the output gates are pushed to zero Hochreiter and Schmidhuber (1997). There isn't a strong consensus regarding how many layers an LSTM model should have or the size of the hidden and embedding dimensions, possibly because these hyperparameter values are dependent on the complexity and number of features in the data.

The best model has 2 layers, a learning rate of 0.005, a hidden size of 256, an embedding size of 512, and no dropout. This model achieves A 72% accuracy on test data, takes 62.64638 seconds to train on 15 epochs, and 2.39482 seconds to predict on the test set.

Layers	LR	H. Size	E. Size	Dropout	Loss	F1
2	0.001	512	512	0	0.10515	0.96968
2	0.001	512	512	0.1	0.11210	0.96502
2	0.005	512	512	0	0.11561	0.97061
2	0.005	256	512	0	0.09655	0.97356
2	0.005	128	512	0	0.11975	0.97003
3	0.005	256	512	0	0.12319	0.97571
2	0.005	256	256	0	0.14540	0.96340
2	0.005	256	1024	0	0.12627	0.96884

Table 2: Hyperparameter values experimented with and their effects on validation F1 and loss in the BiLSTM model.

4.2 BiGRU Model

The bidirectional GRU does better with 2 layers, a smaller learning rate, a large hidden size, a moderate embedding size, and some dropout. Having a higher hidden size and moderate embedding size suggests that the data is complex and has many features, though it is interesting that this model had a preference towards higher hidden and embedding sizes compared to the LSTM model, despite the data being fixed. While dropout does not necessarily perform as expected in recurrent networks (Semeniuta et al., 2016), the GRU model has a preference towards a very slight dropout rate. Contradictorily, Cahuantzi et al. (2023) suggests that a higher learning rate would be beneficial for recurrent networks, whereas this GRU model shows better performance on lower learning rates. Cahuantzi et al. (2023) also recommends a single layer for recurrent networks, however this was not explored due to the inclusion of dropout, but is consistent with the findings that 2 layers are better than 3.

The best model has 2 layers, a 0.001 learning rate, a hidden size of 1024, an embedding size of 512, and a dropout of 0.1. The model gets an accuracy of 74% on the test set, takes 462.55890 seconds to train, and 12.36858 seconds to predict. Surprisingly, the GRU model took much longer to train despite the architecture theoretically being more efficient, though this could be attributed to the increased hidden size.

Layers	LR	H. Size	E. Size	Dropout	Loss	F1
2	0.001	512	512	0	0.11359	0.97051
2	0.001	512	512	0.1	0.10394	0.97334
2	0.001	512	512	0.2	0.10718	0.97307
2	0.005	512	512	0.1	0.17143	0.97225
2	0.001	256	512	0.1	0.11337	0.97003
2	0.001	1024	512	0.1	0.11607	0.97351
2	0.001	1024	1024	0.1	0.12520	0.96630
3	0.001	1024	512	0.1	0.13442	0.95966

Table 3: Hyperparameter values experimented with and their effects on validation F1 and loss in the BiGRU model.

References

Roberto Cahuantzi, Xinye Chen, and Stefan Güttel. 2023. *A Comparison of LSTM and GRU Networks for Learning Symbolic Sequences*, page 771–785. Springer Nature Switzerland.

Yarin Gal and Zoubin Ghahramani. 2016. A theoretically grounded application of dropout in recurrent neural networks.

Arshit Gupta, John Hewitt, and Katrin Kirchhoff. 2019. Simple, fast, accurate intent classification and slot labeling for goal-oriented dialogue systems.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. *Long Short-Term Memory*. Massachusetts Institute of Technology.

Valentin Hofmann, Janet Pierrehumbert, and Hinrich Schütze. 2021. Dynamic contextualized word embeddings. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6970–6984, Online. Association for Computational Linguistics.

Mandy Korpusik, Zoe Liu, and James Glass. 2019. A comparison of deep learning methods for language understanding. pages 849–853.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization.

Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. 2016. Recurrent dropout without memory loss.

Yile Wang, Leyang Cui, and Yue Zhang. 2020. How can bert help lexical semantics tasks?