Slot Tagging of Natural Language Utterances

Steven Au
UCSC NLP 243

Abstract

To learn how to use PyTorch to develop and train a deep neural network model for tagging sequential data using various techniques such as Multilayer Perceptrons (MLP), Convolutional Neural Network (CNN), and Recurrent Neural Networks (RNN) with various optimizers.

1 Introduction

The project revolves around IOB and class tagging for movie utterances. The model must identify what relation the I and B tags are Named Entity Recognizing in the utterances. This is a supervised learning problem of sequence tagging that requires data analysis that best adjusts the model to our features.

1.1 Dataset Overview

The dataset comprises user utterances and their IOB tags. This is part of a Named Entity Recognition problem where each label consists of a Begin token and an Inside token, while the O tag is for an Outside token. It's a multi-classification challenge, where each token corresponds to a single class. These utterances are in lowercase, devoid of punctuation, and can be as short as a word or as long as a paragraph. The tags are separated by spaces and contain 13 different labels totaling 27 unique labels.

Dataset Statistics

- Total number of rows: 981
- Length of utterances:
  - Average length: Approximately 34.71 characters
  - Minimum length: 5 characters
  - Maximum length: 107 characters
- Number of repeated utterances: 11
- Mislabeled I_movie with I-movie

B Tokens	**I Tokens**
B_producer	I_producer
B_movie	I_movie
B_director	I_director
B_person	I_person
B_language	I_language
B_genre	I_genre
B_mpaa_rating	I_mpaa_rating
B_subject	I_subject
B_cast	I_cast
B_country	I_country
B_char	I_char
B_release_year	I_release_year

Table 1: B and I Tokens

2 Models

The allowed models are any neural networks excluding transformer-based models. A Convolutional Neural Network (CNN) is required and were given an example code to work off of and a basic Recurrent Neural Network (RNN) starter code. My goal was to work on a Bidirectional Long Short Term model (BiLSTM) for the Kaggle submission and implement a working token CNN for the assignment.

2.1 CNN

Implemented a Character CNN (CharCNN) without pre-trained embeddings with different architectures. Also, I used word CNN with basic configurations and parameters with starter code for data preprocessing.

2.2 RNN

Used the starter code as a base model for my baseline that used frozen embeddings and dropout.

ID	UTTERANCES	CORE RELATIONS
0	who plays luke on star wars new hope	O O B_char O B_movie I_movie I_movie I_movie
1	show credits for the godfather	O O O B_movie I_movie
3	find the female actress from the movie she's the man	O O O O O O O B_movie I_movie I_movie I_movie
159	how old is steven spielberg	O O O B_person I_person
165	how much did it cost to produce the dark night	O O O O O O O B_movie I_movie I_movie

Table 2: Sample dataset table.

2.3 Bi-LSTM

Converted RNN Model into BiDirectional LSTM keeping RNN layer architecture to capture sequential patterns before and after tokens. I attempted BiLSTM with conditional random field (CRF) as my first model, I was able to get a training model, but unfortunately broke the code and was unable to recover a working version. Analysis will be mentioned in the inference section about its capabilities, using a different loss function.

3 Experiments

3.1 Basic Metrics

- Step loss and accuracy:
- Classification Report: Refer to Table 5
- Metric scores for data split: Refer to Table 4
- Kaggle test accuracy: Refer to Table 2
- Mislabeled Utterances: Refer to Table 7

3.2 Data Split

All models implement the same three-part data split structure for consistency.

- **Training set (90%)**: Used to train the model from train file
- **Validation set (10%)**: Used for hyperparameter tuning and early stopping.
- **Test set**: Used to evaluate the model's performance on unseen data on test file

3.3 Training Configurations

No training configurations are applied to any of the models due to time constraints.

3.4 CNN Configurations

3.4.1 Char CNNs

- **Loss function**: Crossentropy Loss
- **Optimizer**: Adam optimizer
- **Batch Size**: None
- **Epochs**: 10

Embedding Layer:

- 50 Character Dimension Vector

Convolutional Layer:

- 1 Dimension.
- 128 filters
- Kernel Size = 3
- Padding = 1

Fully Connected Layer:

- Linear layer mapping outputs.

3.4.2 CNN

- **Loss function**: Crossentropy Loss
- **Optimizer**: Adam optimizer
- **Batch Size**: 32
- **Learning Rate**: .01
- **Epochs**: 5, 10

Embedding Layer:

- Tuned weights for word embeddings from glove.
- Frozen Embedding to keep some vectors static and introduce robustness.

- Embeddings are concatenated to capture different sequences.

Dropout Layer:

- p = 0.2 to help generalize training

Convolutional Layer:

- 2 Layers
- 1 Dimension.
- 600 filters
- Kernel Size = 3, 5
- Padding = 1, 2

Fully Connected Layer:

- Linear layer mapping outputs.

3.5 RNN/LSTM

Embedding Layer:

- Tuned weights for word embeddings from glove.
- Frozen Embedding to keep some vectors static and introduce robustness.
- Embeddings are concatenated to capture different sequences.

Dropout Layer:

- p = 0.2 to help generalize training

LSTM Layer:

- Hidden Size = 10
- 2 Layer
- Bidirectional for the LSTM implementation

Fully Connected Layer:

- Linear layer mapping outputs the forward (and backward) hidden states.

4 Results

My goal was to understand the PyTorch pipeline from the starter code and implement a BiLSTM CRF as my final submission. I was able to train a model but had padding issues when evaluating metrics. I did not save my working code after trying to fix the bug. My idea was to use a Bidirectional model to capture tokens in both directions and have CRF create probabilities for predicting appropriate IOB tag logic that the model did not capture. I started with this model, before implementing the starter code again after I broke everything.

4.1 RNN/LSTM

I mainly focused on using the starter code to understand pytorch implementation and as my basis for tuning the mode. Simply enabling the bidirectional tag to True and doubling the output layer improved the performance significantly. The BiLSTM was able to predict most of the unique labels while the out-of-the-box RNN only managed to capture the movie label while not making any training progress elsewhere.

I was able to get CRF working with BILSTM with the starter code architecture. Shown below is the improvements that CRF provides when adding logic during prediction. It was able to remove simple errors of predicting two different labels in a row (B_person I_producer).

Category	Precision	Recall	F1-Score	Support
cast	0.09	0.07	0.08	95
char	0.00	0.00	0.00	10
country	0.35	0.31	0.32	144
director	0.11	0.12	0.11	169
genre	0.40	0.10	0.16	62
language	0.42	0.46	0.44	107
location	0.00	0.00	0.00	2
movie	0.44	0.54	0.49	924
mpaa-rating	0.46	0.33	0.38	124
person	0.17	0.17	0.17	176
producer	0.13	0.10	0.11	140
release-year	0.00	0.00	0.00	5
subject	0.36	0.35	0.35	77

Table 3: BiLSTM Classification Performance Metrics

Category	Precision	Recall	F1-Score	Support
cast	0.28	0.24	0.26	93
char	0.00	0.00	0.00	13
country	0.28	0.17	0.21	141
director	0.38	0.33	0.35	164
genre	0.17	0.06	0.09	63
language	0.47	0.41	0.44	108
location	0.00	0.00	0.00	2
movie	0.61	0.61	0.61	913
mpaa-rating	0.53	0.40	0.45	129
person	0.37	0.46	0.41	173
producer	0.40	0.25	0.30	150
release-year	0.00	0.00	0.00	3
subject	0.54	0.44	0.49	84

Table 4: CRF Classification Performance Metrics

4.2 CNN

I spent a lot of time tuning the convolutional layers but ultimately failed to develop an architecture that learned sequence tagging. I could only implement a simple one-layer CNN without dealing with any of the matrix matching for output layers. I realized I did not have a lot of applicable skills in creating and modifying a CNN model, so I spent a lot of time trying to understand CNN implementations. I started with a single-layer CNN that only learned O tags, and built a CharCNN from scratch to understand the NN pipeline. I was able to learn the CNN parameters and build the CNN while preserving dimensions. I managed to implement a CNN with 2 layers and one with different kernel sizes for the initial layer.

I was able to build 2 CNN models that ran for characters and tokens. I experimented with kernel sizes, reLU, batch normalization, and multi-kernel models. I picked odd kernel sizes since even batching created a mismatch in dimensions and went with odd intervals. The kernel size was important for recognizing sequences with that shapes. In Word CNN it looked at 3-word and 5-word phases for tag slotting. My best kaggle score was from this and manually observed the training rate to prevent overfitting. My previous attempt was 10 epochs and was around 20%. I decided to cut the epoch to around 70 accuracy at epoch 6 and got close to 40% accuracy on the test set.

Category	Precision	Recall	F1-Score	Support
cast	0.41	0.49	0.45	98
char	0.64	0.50	0.56	14
country	0.79	0.88	0.83	135
director	0.55	0.68	0.61	166
genre	0.87	0.91	0.89	65
language	0.92	0.92	0.92	108
location	0.00	0.00	0.00	1
movie	0.91	0.92	0.91	908
mpaa-rating	0.83	0.84	0.83	130
person	0.40	0.55	0.46	173
producer	0.54	0.50	0.52	144
release-year	0.00	0.00	0.00	5
subject	0.93	0.97	0.95	89
micro avg	0.76	0.81	0.78	2036
macro avg	0.60	0.63	0.61	2036
weighted avg	0.77	0.81	0.79	2036

Table 5: Detailed Classification Performance Metrics

5 Inference

CNN is great at capturing spatial patterns but requires a good knowledge of the model architecture on how the model is capturing features. RNN/LSTM is easier to implement as sequence length is not a huge issue as than can recognize variable sequence length. CNN is constrained on how the model develops as it can only recognize patterns it seems before. RNN/LSTM can be tuned to carry semantic information better from previous strings. CNN strength in temporal locality was not best for this task as the utterances carried a similar utterance structure. Movies were mostly mentioned at the end and did not need a CNN to recognize movie titles at the beginning of the sentence. The problem was a lot simpler since the utterances were short and followed a general sentence structure. This made predictions based on context a lot harder. Deciphering between movie titles, or outside tags, is where a lot of the mislabels came from. The other mislabel was where the model generalized too many on-label predictions that broke the IOB structure. This can be fixed with a different lost function using hidden Markov models and implementing CRF.

If I had more time, I would like to explore more of the CRF and understand how it works, as we started learning it in 201. I also want to explore a combination model of CNN and LSTM; Char CNN to explore morphology with similar phrases or misspelled utterances, and LSTM to preserve n-gram semantics for word embeddings.

Model	Kaggle Score
BiLSTM Base	0.157
CNN 2 Layer	0.239
CNN Tune	0.397

Table 6: Model Performance Comparison

6 References

Look at tutorials to understand the slot tagging problem and its solutions. The references were only for optional models.

Github Repo Tutorial:
github.com/TheAnig/NER-LSTM-CNN-Pytorch/

PyTorch BiLSTM CRF Tutorial:
pytorch.org/tutorials/beginner/nlp/advanced_tutorial.html