Semantic Slot Filling – Project Report

GitHub URL

Yesha Shah and Richa Patel
Tagliatela College Of Engineering
University of New Haven
West Haven, CT, 06516
Under the guidance of Prof. Vahid Behzadan

Abstract

Conversational dialogue systems like Amazon Alexa, Google Assistant, Apple Siri, and Microsoft Cortana have exploded in popularity thanks to the proliferation of mobile internet and smart gadgets. These systems rely heavily on natural language understanding (NLU). Slot filling is often considered as a sequence labeling issue, in which semantic labels are applied to contiguous sequences of words (slots). Deep learning has outperformed most standard techniques in the slot filling problem in NLU. Deep learning, on the other hand, is infamous for doing badly when given minimal labelled data.

1. What slot filling actually means

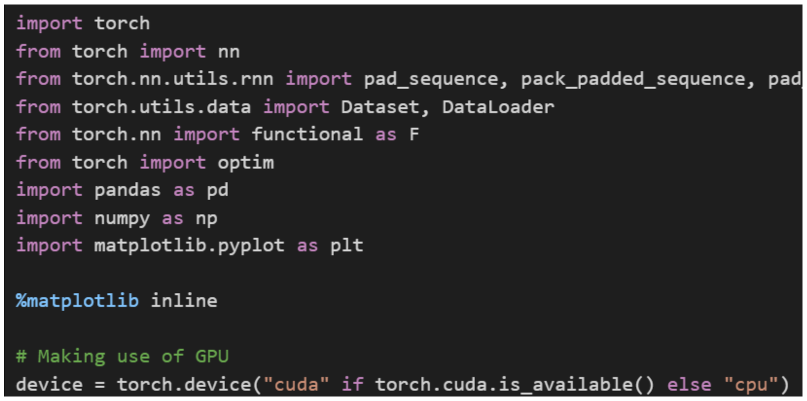
Slot filling is the process of identifying continuous sequence of words that correspond to certain parameters of a user request/query. It is one of the most challenging problems in spoken language understanding (SLU). There are many different approaches and algorithms already implemented. They are Rule based approach, ML based, DL based.

2. Statement of purpose:

Currently, all the models that exist require high computation and are very complex. They require more resources and some rely on fine-tuning pretrained models such as BERT. Our goal is to achieve similar outputs even with a simple DL based model. We plan on using the same dataset as used in other pre-existing models. The dataset is SNIPS by Snips.ai

3. Approach:

The primary task is filling systems, which extracts a collection of characteristics or "slots" and their associated values automatically. The key concepts used in this project are – vocabulary building, word to index and index to word mapping, padded sequences, word embeddings, bidirectional variants. We have used the dataset SNIPS - dataset by Snips.ai for Intent Detection and Slot Filling benchmarking. It contains several day-to-day user command categories - such as play a song, book a restaurant, etc. For the actual network itself, we decided to go ahead with GRU along with segment tagging and Named Entity Recognition. The major packages used throughout the project are pytorch, pandas, numpy, and matplotlib.

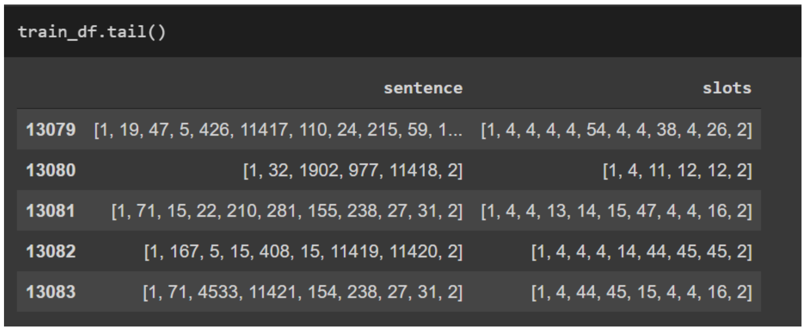


```
import torch
from torch import nn
from torch.nn.utils.rnn import pad_sequence, pack_padded_sequence, pad
from torch.utils.data import Dataset, DataLoader
from torch.nn import functional as F
from torch import optim
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline

# Making use of GPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

Figure 1: Importing packages

4. Data preprocessing:

First, added beginning of statement and end of statement markers sentences inputs and outputs. Then, using all the input statements in the training data, create a vocabulary. Similarly, the slots are created using the words from the output sequences. These vocabulary and slots are mappings from words/slots to numbers. Then, created train, tests, and validation datasets from the SNIPS dataset. This is done by splitting the data in train and validation and load the trading dataset using a data frames with 2 columns – input sentence, and output slots. For test and validation data, if there missing words in vocabulary, we fill the place with <unk> i.e., Unknown. We created custom class to handle the SNIPS dataset and dataloader where the inputs are sentence and outputs are slots and within a batch, padded variable length sentence slots to the batch's maximum length.



```
train_df.tail()
```

	sentence	slots
13079	[1, 19, 47, 5, 426, 11417, 110, 24, 215, 59, 1...	[1, 4, 4, 4, 4, 54, 4, 4, 38, 4, 26, 2]
13080	[1, 32, 1902, 977, 11418, 2]	[1, 4, 11, 12, 12, 2]
13081	[1, 71, 15, 22, 210, 281, 155, 238, 27, 31, 2]	[1, 4, 4, 13, 14, 15, 47, 4, 4, 16, 2]
13082	[1, 167, 5, 15, 408, 15, 11419, 11420, 2]	[1, 4, 4, 4, 14, 44, 45, 45, 2]
13083	[1, 71, 4533, 11421, 154, 238, 27, 31, 2]	[1, 4, 44, 45, 15, 4, 4, 16, 2]

Figure 2: Training dataset - dataframe

4.1 From the preprocessing

After the preprocessing, the size of the vocabulary and the number of slots were checked. Additionally, the length of training dataset was also calculated as below.

Vocabulary Size: 11422

Types of Slots: 76

Number of training examples: 13084

4.2 A simple example

For the given input: Add this track by Taylor Swift to Acoustic Souls;

The output looks something like this:

[O: Add] [O: this] [B-music_item: track] [O: by] [B-artist: Taylor] [I-artist: Swift] [O: to] [B-playlist: Acoustic] [I-playlist: Souls]

In slot filling, there are two major terms used - slots and labels. Slots are the slots to be filled – such as 'O', 'B', and 'I'. 'B' indicates beginning of an entity, 'I' represents Inside - i.e. it is the part of the entity that began before it. 'O' indicates no entity. Labels are the words with which the slots are filled; in the example above, they are music_item, artist, and playlist. Before the data loader we follow a few steps for data processing.

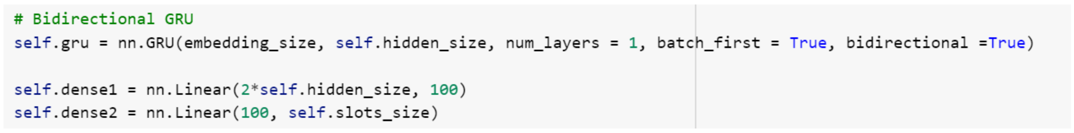
5. Semantic slot filling model:

We have one GRU and two linear layers in the model. The GRU is bidirectional, with embedding size and hidden size as 100. Vocabulary and slot sizes are fixed, and are the respective lengths of vocabulary and total number of slots. We utilized GRU to generate output and get hidden states in the forward layers, then we packaged the output as padded sequences and passed it through the linear layer, an activation function of ReLU, and then, another linear layer. So, the network model has Vocab size, Slot size, 100 Hidden size, and 100 embedding size.

GRU – Embedding, hidden size = 100, #layers = 1, bidirectional = True

First Linear Layer – #In = 2 * hidden size, #Out = 100

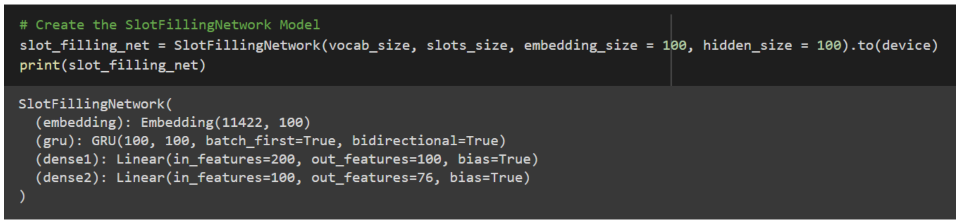
Second Linear layer – #Inputs = 100, #Output = Slot size = 76



```
# Bidirectional GRU
self.gru = nn.GRU(embedding_size, self.hidden_size, num_layers = 1, batch_first = True, bidirectional =True)

self.dense1 = nn.Linear(2*self.hidden_size, 100)
self.dense2 = nn.Linear(100, self.slots_size)
```

Figure 3: Model architecture

5.1 Creating the actual model



```
# Create the SlotFillingNetwork Model
slot_filling_net = SlotFillingNetwork(vocab_size, slots_size, embedding_size = 100, hidden_size = 100).to(device)
print(slot_filling_net)
```

```
SlotFillingNetwork(
  (embedding): Embedding(11422, 100)
  (gru): GRU(100, 100, batch_first=True, bidirectional=True)
  (dense1): Linear(in_features=200, out_features=100, bias=True)
  (dense2): Linear(in_features=100, out_features=76, bias=True)
)
```

Figure 4: Building the model

6. Training the model:

We now train the real model, having setup the optimizer and loss functions. The learning rate is 0.001 and the weight decay is 0.0005, and the loss function is cross entropy loss. We utilized 24 epochs and a batch size of 64 for the actual training. We also calculated the validation loss. The graph of training and validation loss over the epochs looks as below.

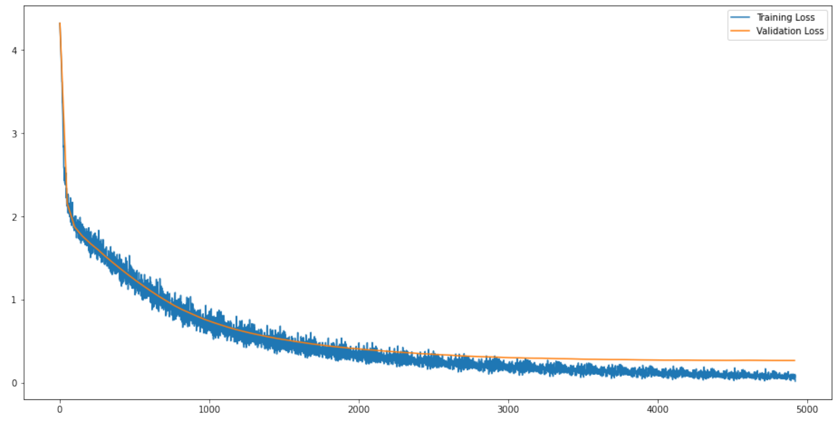


Figure 5: Plotting the validation and training losses

6.1 Saving the model parameters

For ease of use in the future, without needing to re-train the model, we also preserved the model parameter as a pretrained model. This can be utilized in the future without having to train it.

7. Model evaluation

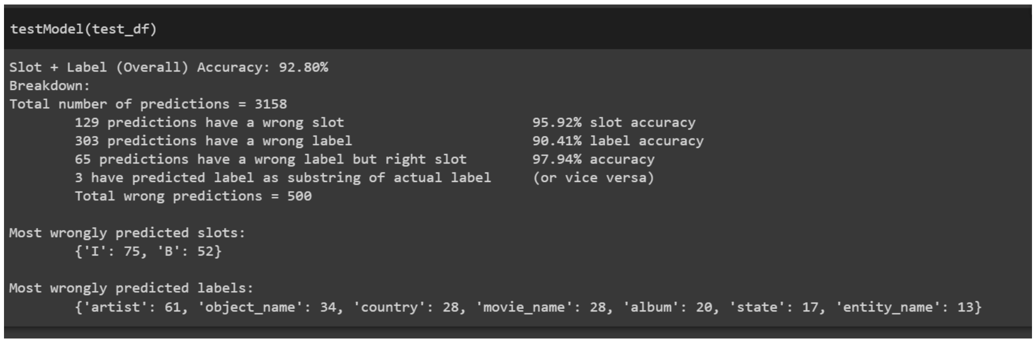
To evaluate the performance of the model, we came up with different form of metrices. They are as follows – overall accuracy, wrong slots, spurious slots, wrong label, and wrong boundary. Each of these terms can be explained as below:

- Accuracy – NumberOfExcatMatches / TotalNumberOfPredictions
- Missing Slot – predicted slot doesn't exactly match actual slot
- Spurious Slot – label of predicted slot doesn't match actual labels
- Wrong Boundary – predicted label is a substring of actual (or vice versa)
- Wrong Label – the label predicted is wrong but the slot matches

We'll now look at the model's assessment and resulting outcomes are as below:

Table 1: Evaluation of the model

Metric	**Value in evaluation**
Accuracy	96.75%
Missing slots	108
Spurious slots	271
Wrong boundary	49
Wrong label	1
Total predictions	3197



```
testModel(test_df)

Slot + Label (Overall) Accuracy: 92.80%
Breakdown:
Total number of predictions = 3158
        129 predictions have a wrong slot                    95.92% slot accuracy
        303 predictions have a wrong label                   90.41% label accuracy
        65 predictions have a wrong label but right slot     97.94% accuracy
        3 have predicted label as substring of actual label  (or vice versa)
        Total wrong predictions = 500

Most wrongly predicted slots:
        {'I': 75, 'B': 52}

Most wrongly predicted labels:
        {'artist': 61, 'object_name': 34, 'country': 28, 'movie_name': 28, 'album': 20, 'state': 17, 'entity_name': 13}
```

Figure 6: Evaluation of the model

8. Comparison with existing models

Table 2: Comparison with existing models

Model name / type	**Accuracy**
SlotRefine + BERT	99.05%
SlotRefine	97.44%
Stack-Propagation + BERT	99%
Stack-Propagation	98%
SF-ID (BLSTM) network	97.43%
Capsule-NLU	97.70%
Slot-Gated BLSTM with Attention	97.00%
BiGRU (Our approach)	**96.75%**

9. Conclusion

Although the accuracy of our approach is not as good as the existing models, simple changes in our model (even simply finetuning the hyperparameters!) could improve the model accuracy. Additionally, it is important to note that other metrics are equally important and should not be ignored while comparing all the models. As a result, we can infer that our approach along with the train time and computation time consumed by our model is more efficient than other highly complicated models. As a result, simple adjustments to the model may enhance accuracy and might even allow it to match existing models.

References:

- https://aclanthology.org/2021.acl-long.340.pdf
- https://medium.com/koderunners/semantic-slot-filling-part-1-7982d786928e#:~:text=One%20way%20of%20making%20sense,known%20as%20Semantic%20Slot%20Filling.
- http://nlpprogress.com/english/intent_detection_slot_filling.html
- https://pytext.readthedocs.io/en/master/hierarchical_intent_slot_tutorial.html
- https://media.neurips.cc/Conferences/NeurIPS2020/Styles/neurips_2020.pdf