CSE244 Homework 2 Report: Slot Tagging

Molly Can Zhang, Feb 25 2020

Introduction

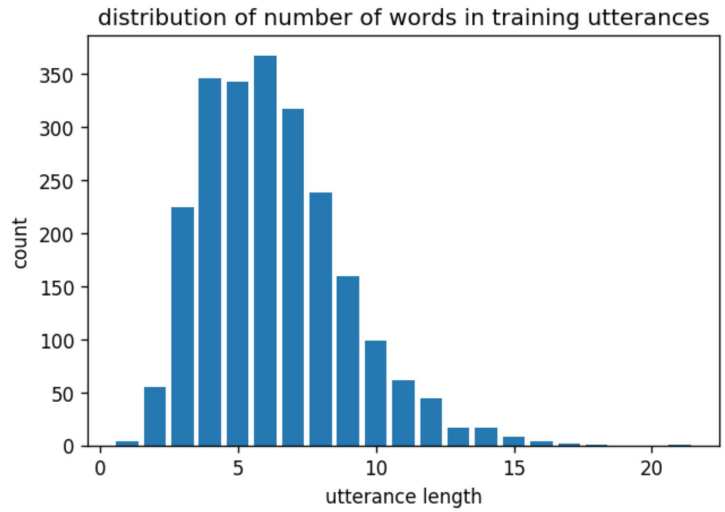
1.1 Data and Task Description

	utterances	IOB Slot tags
2263	list movie ratings for us movie looper	O O O O B_country O B_movie
309	origin of spanglish	O O B_movie
1144	ed harris films	B_person I_person O
1626	show me scifi fantasy movies	O O B_genre I_genre O
2058	show my spielberg info	O O B_person O

The table above shows 5 examples of the data. The feature of the data is an utterance: a sequence of words. The label is a "slot tag" for each word in the utterance. Label O means that word does not belong to a slot, label B_x means that this is the beginning of slot x, and label I_x means that this is the continuation of the slot B_x. I_x tags can only happen after B_x tags. The total number of tags is equal to the total number of words in an utterance.

1.2 Utterances Stats

There are 2321 training samples, 294 validation samples and 687 test utterances. We are given both utterances and slot tags for training and validation, and we don't have any label for test data. In training data utterances, the following plot shows the distribution of utterances length (number of words in an utterance):



1.3 Label Stats

Training data has 26 possible labels. (There are 27 labels in the original training data, however, this is one typo in a label I-movie which double counts as I_movie, after correcting that typo by replacing dash "-" with underscore "_", I ended up with 26 unique labels). These 26 labels and the number of times they occur in training data are shown below:

O	10519
I_movie	1138
B_movie	1018
B_person	195
B_director	185
I_person	176
I_director	168
B_producer	164
B_country	153
B_mpaa_rating	141
B_language	119
I_producer	114
B_cast	106
I_cast	105
B_subject	95
B_genre	71
I_subject	33
I_language	17
B_char	15
I_mpaa_rating	12
I_country	12
I_genre	5
I_char	5
B_release_year	5
I_release_year	3
B_location	2

However, in validation data, there are four additional labels not found in training data, they are: 'B_gross_rev', 'B_org', 'I_gross_rev', 'I_location'. I therefore created one additional label <UNK> in training, and I map all unknown labels to <UNK> during evaluation. This is a compromise I made in this homework, and it's not necessarily a good choice in real life situations, because I basically give up on predicting these labels.

Preprocessing

2.1 Train, validation and test split

I split the 2321 training data into 90% train, 10% holdout test randomly. I reserved the 10% holdout test data only to be used at the very last step when I am deciding on which model to use for final submission, and I use the validation data titled “./hw2_utterance_dev.txt” and “hw2_tags_dev.text” for development during my training.

2.2 Process Labels

The 27 unique slot tags, including “<UNK>”, are converted from string to one-hot encoding.

2.3 Clean labeling error in data

In training data, 14 samples have different number of words in utterances vs number of labels in slot tags. Here are few examples:

	utterances	IOB Slot tags
8	find the guy who plays charlie on charlie 's a...	O O O O O B_char O B_movie I_movie
21	what 's the cast for life is beautiful	O O O O B_movie I_movie I_movie
25	who were roberto benigni 's costars on life is...	O O O O O O B_movie I_movie I_movie
62	i can 't remember the lead guy in the avengers	O O O B_movie O O O B_movie I_movie
612	what is the director 's name of lord of the rings	O O I_movie O O I_movie I_movie I_movie I_movi...
614	rocky director 's name	B_movie O O

Additionally, there are 6 samples in which I_x tags appear right after an O tag, which shouldn’t be possible. Here are some examples:

	utterances	IOB Slot tags
428	when was the first hunger games shown	O O B_movie O I_movie I_movie O
470	who directed the movie	O O I_movie O
612	what is the director 's name of lord of the rings	O O I_movie O O I_movie I_movie I_movie I_movi...
672	language of sound of music	O I_movie I_movie I_movie I_movie

All of these 20 examples are corrected manually. This correction leads to much improved prediction (results not shown).

2.4 fasttext and glove word embedding

I downloaded pretrained fasttext and glove word embedding. Then I filtered the fasttext or glove word vocabulary down to the 1098-word vocabulary from this dataset, these word embeddings are both used later to test their efficacy.

Model Descriptions

All models used in this homework are in model_utils.py.

3.1 Loss Function

All models share the same loss function, CrossEntropyLoss. In all models, the output is 27 logits, each representing a label. The softmax over the 27 output represents the probability distribution of underlying label.

3.2 BaseModel

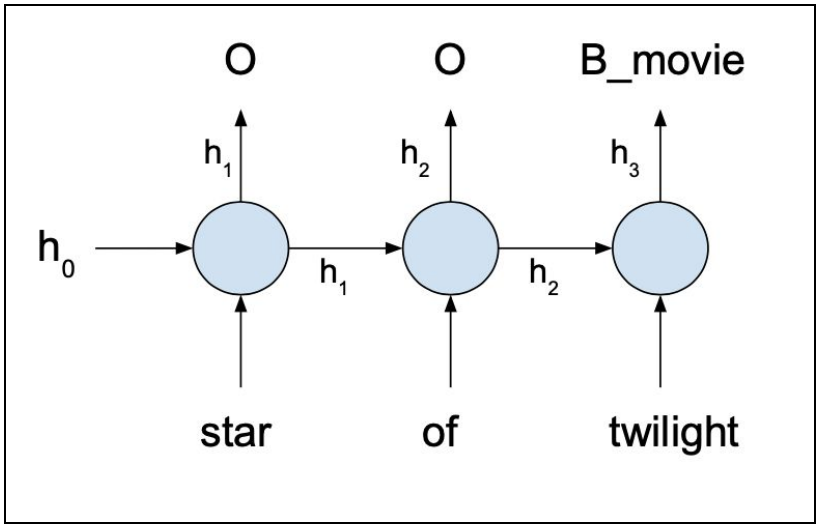
This model is the most basic one. It assumes that each word in an utterance is independent of each other. BaseModel takes word embedding as input, the input dimension is the same as word embedding dimension, which is 300 in my model. It then directly outputs the 27 label logits. It's essentially a logistic regression model predicting slot tags straight from word embedding. The performance of this model (see results section) is surprisingly good enough to serve as a simple baseline, and this model is used to make basic decisions about data preprocessing, debugging etc.

3.3 GRU

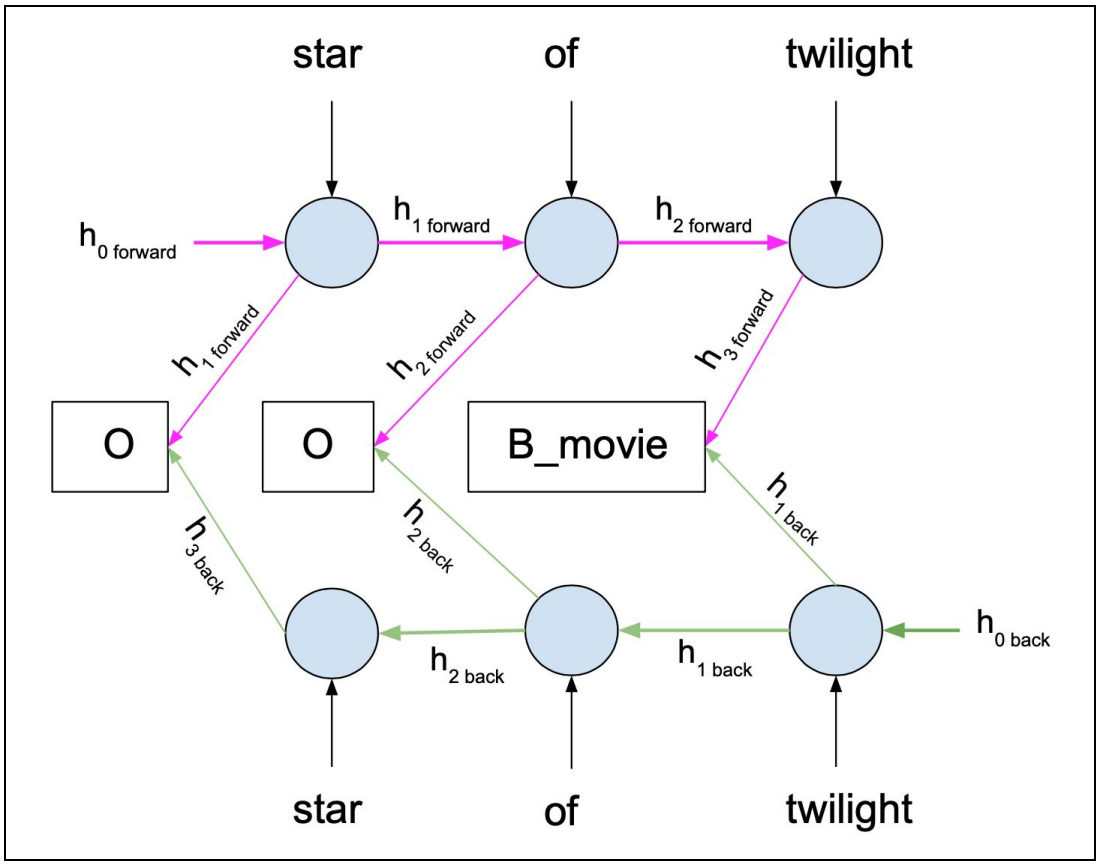
I chose to use nn.GRUCell in pytorch to build GRU models. In all my models, I make sure that at each batch, the utterances all have the same length. This can be possible by doing one of these two things: 1) Having a batch size of one. 2) Group together utterances of the same length in one batch. I tried both methods, and in particular the second method works very well, and it's described in more detail in section 4.1. By making sure that the utterances all have the same length L in a batch, I can stack L number of GRUCell together in one batch, and therefore achieves dynamically adapting the number of the GRUCell based on input sequence length, without needing to use padding.

I used 300 input dimensions and 200 hidden units in the GRUCell. In uni-directional GRU model, if the sequence length is L, it produces an output matrix of size L * 200, which is then fed into a linear layer of shape (200, 27), where 27 is the output dimension. In bi-directioan GRU model, a separate backward GRUCell weights track the backward direction, and I concatenate the forward and backward output, which then produces a L * 400 matrix, followed by a (400, 27) linear layer that connects the hidden states of each step in the GRU to an output. In this model, a sequence of length L always produces a output sequence of the same length, which is exactly what we needed for this task.

Unidirectional GRU model:



Bidirectional GRU model:



3.4 BertSlot

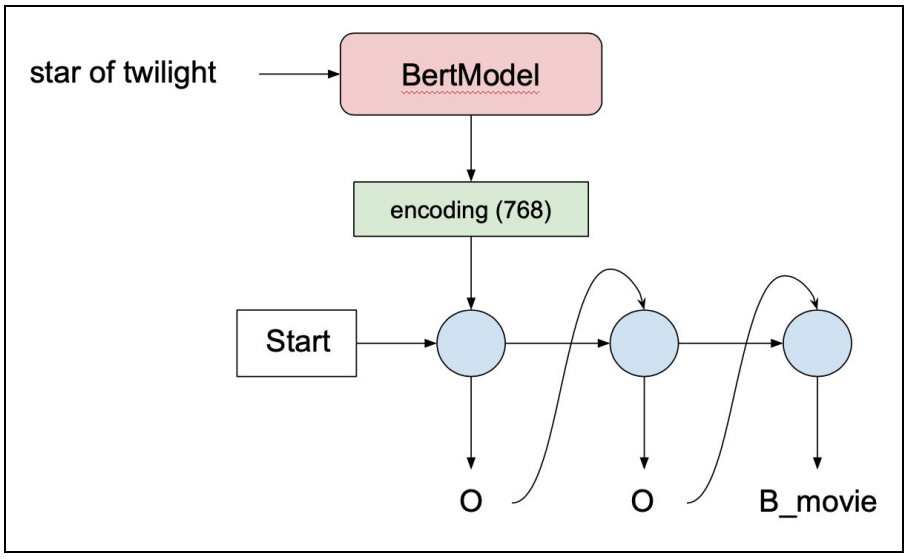
BertSlot is built on top of BertModel from huggingface transformer. It's essentially an encoder-decoder seq2seq model using pre trained bert model to extract embedding. The BertModel would take an utterance as input, and then output an "embedding" of dimension (L' * 768), where L' is the output sequence length and 768 is the embedding dimension.

Bert has its own data tokenization method, called "BertTokenizer", it would split a word into subwords if this word is not found in the existing dictionary. Due to this behavior of BertTokenizer, the length of sequence L' after tokenization is usually not the same as the original length of utterance L. Therefore, I cannot simply add another GRU layer (as described in 3.3) directly on top of bert output.

Instead, I take an average of the (L' * 768) output from bert along the first dimension, to get a "context vector" of size 768, this part can be considered to be an "encoder", and then the context vector is fed into a GRU of length L that serves as "decoder", and it would output a sequence of slot tags of length L.

There are two crucial parts that I did NOT do: 1) Attention mechanism. It would definitely have improved the performance of this model by paying attention to particular positions in the input. 2) Teacher-forcing in the decoder, such that I use true labels in training to "force" the models to learn a correlation between the sequence of labels. As a result, my model doesn't end up performing well. However, I would like to invest time in project rather than homework so I am just reporting a sub-optimal Bert model here.

BertSlot Encoder Decoder Model

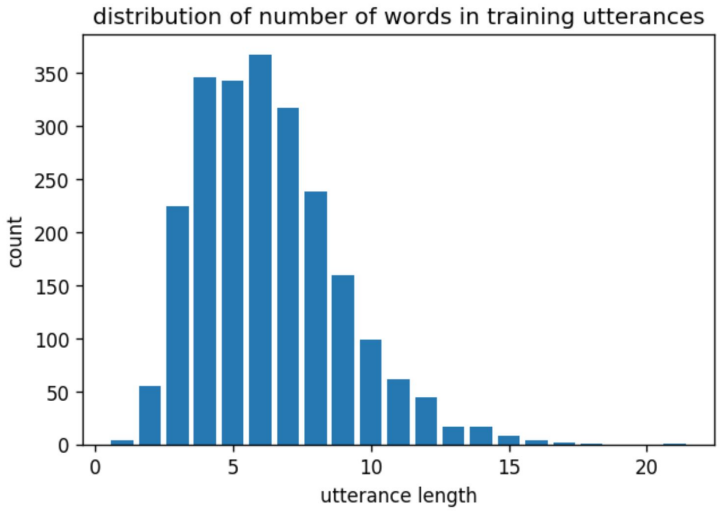


Training Setup

For training the data batch sizes are either 1 or variable, such that the GRU models can have a dynamic length based on input sequence length. (see 3.3 for more description, and see 4.1 for description of variable batch size) . After each epoch, I calculate the training loss, validation loss and validation f1 score. For batch size of 1, I set the “patience” to 5 and stopped training after validation f1 score hasn’t improved for patience number of epochs. For variable batch size, the patience is set to 10. The number of epochs it takes to reach patience depends on the model, but generally it ranges from 10-80 epochs. Learning rate also varies between models, with the simpler models using 1e-2 as learning rate and more complex models using learning rate of 1e-5. The learning rate is reduced by a factor of 10 when validation loss hasn’t improved for patience number of epochs. I used Adam optimizer. I trained all models with two 1080ti GPUs.

4.1 Variable Batch Size

In order to make sure that all utterances in each batch have the same length, I created a class called VarBatch in data_utils.py. This allows grouping utterances of the same length together in one batch. Because some utterance length only occurs 1 or 2 times, and some other utterances length occurs hundreds of times (see plot bellow), I allow the batch size to vary. The maximum



batch size is set to 32, and if there are only 5 utterances in the training data, then that 5 utterances then form a batch. This method allows for dynamically changing the length of GRU layer based on batch, it eliminates the need for padding during training, and it’s much faster than simply using a batch size of 1. The speedup will be shown in the result section.

Results

All results are reported as validation f1 score at two decimal points because there are variations in f1 score between runs at third decimal points due to random initialization. and I just want to report the f1 score to minimize the variations.

5.1 Baseline

Predicting every word to be the most frequent label “B_movie” gives a **0.11** f1 score.

5.2 BaseModel

Predicting output directly from word embedding using one linear layer gives a surprisingly good **0.59** validation f1 score. This same BaseModel is used for exploring using pretrained fasttext embedding v.s pretrained glove embedding v.s. random initialized embedding, results shown below:

Word Embedding	Validation f1
random	0.59
fasttext	**0.62**
glove	0.55

As can be seen from this table, pretrained fasttext embedding outperforms random initialized embedding which then outperforms glove embedding. Subsequently, fasttext embeddings are used.

5.3 GRU

Clearly bidirectional GRU works better than unidirectional. However, I also tried to use fasttext pretrained embedding for GRU, surprisingly, fasttext pretrained embedding performs very poorly with GRU model, giving only 0.66 f1 score.

Num Direction	validation f1 score
1	0.74
2	**0.80**

5.4 BertSlot

BertSlot model using bert-base reaches a **0.83** validation f1 score, after 120 seconds per epoch and 32 epochs (total training time: 70min). It outperforms GRU significantly. I believe this can be easily improved by using the larger model “bert-large”, as well as adding attention and teacher forcing, as explained in the model section, but due to the long training time I didn’t try that.

5.4 Speedup with varying batch size

Using variable batch size speeds up a more complex model, where as it does slightly worse for the simple BaseModel, as shown below:

BaseModel

Batchsize	Time Per Epoch	Number of Epoch	Total Time	Speedup
1	3.9s	10	**38s**	x 0.8
varying	**1.4s**	33	48s	

Unidirectional GRU

Batchsize	Time Per Epoch	Number of Epoch	Total Time	Speedup
1	12.5s	11	139s	x 3.0
varying	**2.1s**	22	**47s**	

bidirectional GRU

Batchsize	Time Per Epoch	Number of Epoch	Total Time	Speedup
1	22.5s	10	232s	x 3.6
varying	**2.6s**	24	**64s**	

5.5 Prediction for test

./prediction.txt has 687 lines and is the prediction result by BertSlot.

Links and References

- https://github.com/MollyZhang/CSE244_ML_for_NLP_HW2. **Github repo for hw2**
- https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html encoder-decoder tutorial.