Intent Detection Using Transformers

Bhaskar Seshadri
Supreeth Bannur Sudhakara
Madhurima Madiraju

Purdue University

Introduction

Intent detection is a text classification task used in chatbots and intelligent NLP systems. Its goal is to capture the semantics behind users' messages and assign it to the right label. Such a system must be given only a few examples, learn to recognize new texts that belong to the same category as the ones it was trained on. This can often be a tough task as users tend to formulate their requests in ambiguous ways.

Benefits of intent Classification:

1. xConversational AI apps employ intent classification to give consumers individualized conversational experiences. Sales are boosted, and overall customer satisfaction is raised. It enables companies to comprehend the intentions of their clients and respond to them more precisely.
2. Businesses may be able to use it to automate communications between interested customers and salespeople. It organizes the demands of the client and their specific attention before analyzing what the customer wants to accomplish and directing them to the appropriate representative by organizing the customer's intentions.

Problem Statement

The detection of intent is a critical component of any task-oriented conversational system. In order to understand the user's current goal, the system must leverage its intent detector to classify the user's input (provided in varied natural language) into one of several predefined classes, that is, intents. However, the traditional intent system uses static-based encoders (like word2vec), and we intend to explore attention-based NLP algorithms to effectively identify user intents in few-shot learning has become popular.

Dataset

The dataset used for this project is the CLINIC150. Each of the 150 in-domain intent classes have 100 training samples, 20 validation samples, and 30 test samples. The out-of-domain class has 100 training samples, 100 validation samples, and 1,000 test samples. Note that the out-of-domain class does not necessarily need to be used during training time. This is the main version of the dataset.

This dataset is used to assess the performance of intent classification algorithms when "out-of-scope" queries are present. Out-of-scope queries are those that do not fall into one of the system-supported intent classes.

Most datasets include only data that is "in-scope". Our dataset includes both in-scope and out-of-scope data. You might also know the term "out-of-scope" by other terms, including "out-of-domain" or "out-of-distribution".

Model Architecture

For the given problem statement, we have 2 main parts when it comes to implementation:

1. Choosing the correct embedding mechanism:

There are many texts embedding models that can be used to convert tokens to vectors. After careful observation based on the dataset we went ahead with the transformer based word embeddings as the length of the sentences are less in the dataset used. The maximum sentence length used is 33. Hence quality of the embeddings is key. BERT performed best.

When implementing Bert we adopt the feature of adding special tokens like CLS and SEP into the tokens which helps in MLM (Masked language modelling). By default, we use 15% of the sentence is masked to generate a more complex sentence to generate boosting the confidence of the model for complex input queries.

Masked language modeling and picture modeling are like autoencoding modeling in that they generate results from disorganized or contaminated input. Masking, as the name implies, is used with these modeling processes to mask words from a series of input or sentences, and the constructed model must anticipate the masked words to finish the sentence. This type of modeling approach is analogous to filling in the blanks on an exam paper.

MLM entails feeding a sentence to BERT and optimizing the weights within BERT to output the identical sentence on the other side. So, we input a sentence and ask BERT to return the same sentence. However, before passing the input sentence to BERT, we mask a few tokens.

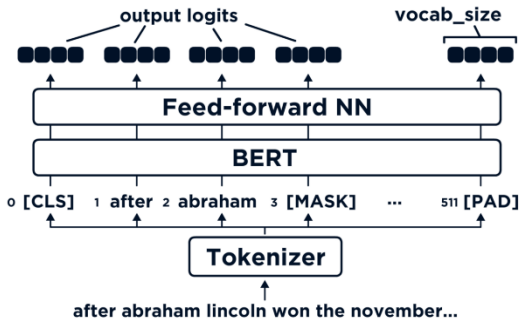


Fig. Before passing our tokens into BERT- we have masked the Lincoln token, replacing it with [MASK].

So, we're inputting an incomplete sentence and asking BERT to complete it for us. We use the following steps:

I. Our text is tokenized. As we would with transformers, we begin with text tokenization.

II. Make a tensor of labels. Because we're training our model here, we'll need a labels tensor to compute loss — and optimize towards.

III. Tokens in input ids are masked. Now that we've generated a copy of input ids for labels, we can mask a random selection of tokens.

IV. Determine your loss. We run the input ids and labels tensors through our BERT model and compute the difference between them.

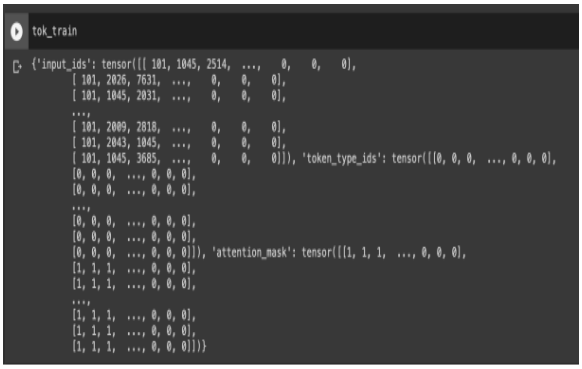


Fig. Tensor after masking

After a given sentence is passed through the steps, the output tensor is generated as shown in the figure above.
The difference between the output probability distributions for each output 'token' and the genuine one-hot encoded labels is used to determine the loss.

2. Choosing appropriate weight updating algorithm

Since the input length of the vectors for the model is not extremely complex, a basic gradient descent approach is used to determine the right weights for the words in the training vocabulary. We used the stochastic gradient descent approach because it trains in batches and incorporates unpredictability, making the training more robust. A simple training for 10-15 epochs yields the greatest results.

The term 'stochastic' refers to a system or process with a random probability. As a result, in Stochastic Gradient Descent, a few samples are chosen at random rather than the entire data set for each iteration. Gradient Descent uses the term "batch" to refer to the total number of samples from a dataset that are utilized to calculate the gradient for each iteration. In standard Gradient Descent optimization, such as Batch Gradient Descent, the batch represents the whole dataset.

Although utilizing the entire dataset is quite valuable for locating minima in a less noisy and random manner, the challenge occurs as our dataset becomes large. If you have a million samples in your dataset, and you use a typical Gradient Descent optimization technique, you will have to use all of the one million samples to complete one iteration while performing the Gradient Descent, and you will have to do this for each iteration until the minima are reached. As a result, doing it becomes computationally highly costly. This problem is addressed using Stochastic Gradient Descent. SGD does each iteration with a single sample, resulting in a batch size of one. The sample is jumbled at random and chosen for iteration.

$$\text{for } i \text{ in range } (m):$$
$$\theta_j = \theta_j - \alpha \left(\hat{y}^i - y^i\right) x_j^i$$

So, in SGD, we find out the gradient of the cost function of a single example at each iteration instead of the sum of the gradient of the cost function of all the examples.

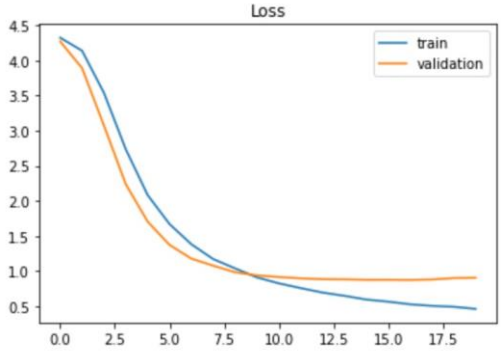
Because only one sample from the dataset is chosen at random for each iteration in SGD, the path travelled by the algorithm to reach the minima is typically noisier than in a traditional Gradient Descent algorithm. However, this is unimportant because the path taken by the algorithm is irrelevant as long as we reach the minima in a substantially shorter training time.

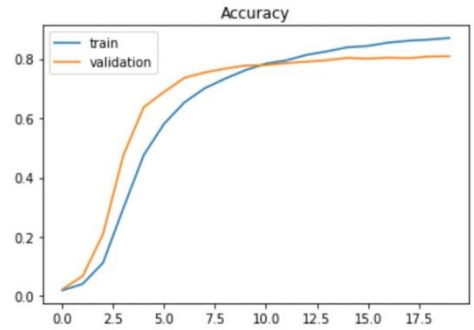
One thing to keep in mind is that because SGD is typically noisier than normal Gradient Descent, it usually takes a longer number of iterations to reach the minima. Even though it takes more iterations to reach the minima than conventional Gradient Descent, it is computationally significantly less expensive than usual Gradient Descent. As a result, SGD is recommended over Batch Gradient Descent for improving a learning algorithm in most cases.

Model evaluation

One of the most common methods for determining whether a model is overfitting or underfitting is to examine its convergence with validation data. We may conclude that the model does not overfit because the loss curve has a smooth slope against the training and validation datasets.

We conducted a test dataset that included 30 samples for each class of intent class. Totally the test set had 5500 entries.





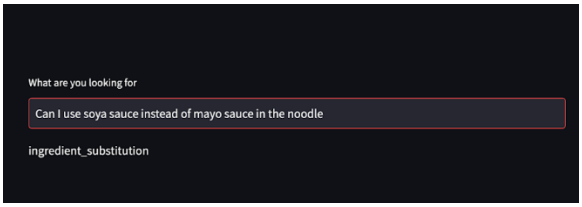
Results

After generating the classification report we can conclude that most of the classes have the confidence score in the range of 0.8-0.95 which states that the model is able to differentiate between the various classes in the dataset.

accuracy			0.81	5500
macro avg	0.82	0.92	0.85	5500
weighted avg	0.84	0.81	0.78	5500

To make the model easy for end user testing we developed a small dashboard using streamlit

where the input is the query from the user and the output is the intent displayed.



References

[1] https://arxiv.org/pdf/1909.02188.pdf

[2] https://github.com/clinc/oos-eval

[3]https://towardsdatascience.com/masked-language-modelling-with-bert-7d49793e5d2c

[4]https://link.springer.com/chapter/10.1007/978-3-642-35289-8_25

[5]https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/