Intent Classification with BERT

Ayan Gangopadhyay

July 21, 2020

Abstract

In this project our goal is to build a state-of-the-art text classifier using transfer learning methods. This text classifier will be used as a part of a chatbot which will help in intent classification. Our data consists of sentence-label pairs which was collected from SeeClickFix[1]. We have tried to use transfer learning to achieve this. We have used a BERT model to generate sentence level embeddings and the used a Classifier network to classify these into our categories. The whole model was implemented using Nvidia NeMo[2] which is a framework for creating state of the art conversational AI. On the face of it this problem sounds simple, and conceptually it is, the difficulty lies in the fact that there are no available datasets for our specific problem. The data preparation phase took considerable manual effort and we did not have much time to fine tune the model further. Another difficulty is that this model will eventually be integrated into a chatbot, due to this popular frameworks like Pytorch and Tensorflow cannot be used. NeMo is a relatively newer framework and has a steeper learning curve than the frameworks mentioned earlier, with the upside of lending itself more readily to chatbot development.There are still some obvious ways to improve our model which have not been explored yet. In spite of all this, we have been able to achieve around an overall accuracy of 75% on our task. We hope to keep fine-tuning our model to improve our overall accuracy.

Problem Statement

Essentially we are trying to build an intent classifier which will take sentences and output labels which correspond to categories. Our model will learn the relationship between sentences and labels which are present in the training data. For this we inted to build a model using transfer learning.

Methodology

BERT

BERT Stands for Bidirectional Encoder Representations from Transformers and is a large scale language model which is trained on unlabelled data from large corpora such as the Wikipedia data set, it essentially learns word vector embeddings by jointly conditioning on both right and left context in all layers. As a result the pre-trained BERT model can be fine tuned with just an additional output layer. BERT is described in [3] and was first proposed by Google.

For our purpose we used BERT-base-uncased pre-trained model available at Nvidia NGC[4]. This model was trained on the Wikipedia and BookCorpus uncased datasets with the sequence length of 512.

Nvidia NeMo

Is a toolkit for developing AI applications. It was chosen to because it helps in streamlining the process for developing conversational AI.

Programming Model

A typical application using NeMo consists of 3 logical stages:

1. Creation of `NeuralModuleFactory` and necessary `NeuralModule`
2. Defining a Directed Acyclic Graph (DAG) of `NeuralModule`
3. Call to "action" such as `train`

NeMo follows *lazy* execution model - actual computation happen only after training or inference is called.

`NeuralModule` is an abstraction between a layer and a neural network, for example: encoder, decoder, language model, acoustic model, etc. Each `NeuralModule` computes a set of outputs from a set of inputs. Every `NmTensor` has `NeuralType` . `NeuralType` describes tensor semantics, axis' order and dimensions. These types are used to determine how modules should be connected together.

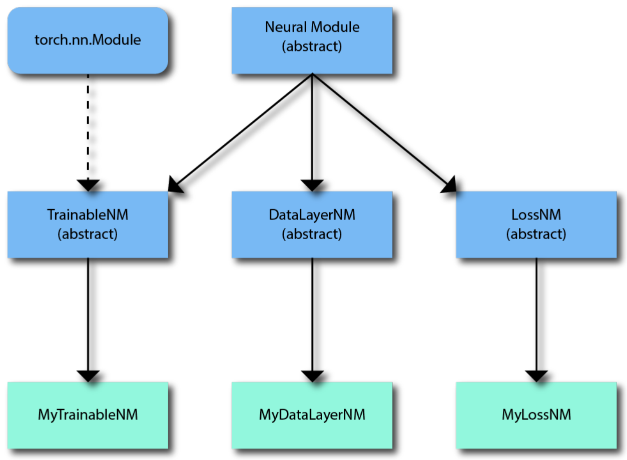
Neural Types

Neural Types are basically data types which are used to check compatibility between different `NeuralModules` when they are connected together.

Neural Modules

Neural Modules can be conceptually classified into 4 potentially overlapping categories:

- Trainable Modules - modules that contain trainable weights. Inherit from `TrainableNM` class.
- Data Layers - modules that perform (extract, transform, load, feed) ETLF of the data. Inherit from `DataLayerNM` class.
- Loss Modules - modules that compute loss functions. Inherit from `LossNM` class.
- Non Trainable Modules - non-trainable module, for example, table lookup, data augmentation, greedy decoder, etc. Inherit from `NonTrainableNM` class.



Inheritance class diagram. Provided API's classes are in Blue. Green classes are to be implemented by user.

Data

We have trained two different models on the two different datasets that we have gathered.

These datasets were gathered using from the APIs provided by SeeClickFix. Information on the *places* API can be be found *here* and information about the *issues* API can be found *here*.

See Click Fix

SeeClickFix is a public request management company which publishes a tool for android and iOS devices for quick reporting of civic issues. It provides an easy interface to select labels corresponding to civic issues and then provide information about the issue through a text field. They also have kept their collected raw data to be accessible through the two APIs mentioned above. Since our intent classification chatbot is supposed to work in a similar domain as that of SeeClickFix, we have used their data to build our model.

SeeClickFix *Places* API

Some statistics about the data collected from the *Places* API are given:

- Examples collected - 41390
- Number of categories (Raw) - ~2.1k
- Reduced categories - 31
- Number of examples in "Reduced Set" - 24089

The following 31 categories were identified for this dataset:

Integer Map	Label
0	Other
1	Pothole
2	Code Enforcement
3	Illegal Dumping
4	Graffiti
5	Camera Outage
6	Trash/Bulk Pick-ups
7	Traffic Signal Issue
8	Sidewalk Repair
9	Street Light
10	Report Code Violation - Other
11	Mosquito Control
12	Abandoned Vehicle
13	Parks
14	Road Issue
15	Signs
16	Tree Issue
17	Parking Enforcement
18	Public Space, Streets and Drains
19	Sanitation Department
20	Flooding/Erosion
21	Environmental
22	Encampment
23	Animal Issues
24	HART Bus Stop Maintenance
25	Property Maintenance (Overgrowth/Grass & Weed Mowing)
26	Yard and/or Structure ... Garages)
27	Snow and Ice Removal
28	HART Arrival Times/Schedules
29	HART Employee Behavior
30	Water Issues

Table 1: Categories in *Places* API

SeeClickFix *Issues* API

Some statistics about the data collected from the *Issues* API are given:

- Examples collected - 280360
- Number of categories (Raw) - >5k
- Reduced categories - 20
- Number of examples in “Reduced Set” - 74084

The following 20 categories were identified in the *Issues* API:

Integer Map	Labels
0	Yard and Structure Concern
1	Drainage Issues
2	Noise Concern
3	Pothole
4	Traffic Signs
5	Trees Issues
6	Street Light
7	Animal Issues
8	Illegal Dumping
9	Parking - Enforcement
10	Nuisance On Property
11	Traffic Signals
12	Sidewalk
13	Abandoned Vehicle
14	Illegal Construction
15	Homeless Concerns
16	Graffiti
17	Health Issues
18	Park Maintenance
19	Pests

Table 2: Categories in *Issues* API

Here we see that for both the datasets a lot of examples have been discarded due to being either incomplete or unfit for the labels we have chosen, further work to reduce the dataset is being carried out.

Model

For both the datasets similar models were used, this model is described in the following illustrations.

First the Transformer is illustrated which form the building blocks of BERT, then the BERT architecture is illustrated and finally we show a pictorial description of our final model.

The Transformer

The transformer architecture was introduced by Google in the paper [5] by Vaswani *et. al.*. Previously LSTMs and GRUs dominated the field of Natural Language Processing (NLP), but they contained time-recurrences and it was a well known problem that these variants of RNN, though powerful, were hard to train. Transformers did away with the need for recurrences altogether hence making training parallelizable using a mechanism called attention, which were used with RNNs before. Although this meant that we lost the advantage of having variable-length input which made RNNs particularly attractive, in practice this did not pose a problem. For example our pre-trained BERT model was trained on sequence lengths of 512 words, which means it could process sentences which were as long as 512 words. Since, in practice we rarely write such long sentences this did not pose much of a problem, especially since it was found to work much faster than LSTMs and GRUs.

Performance metrics for the transformer can be found in the paper and have revolutionized the NLP landscape. A brief illustration of the transformer is given here.

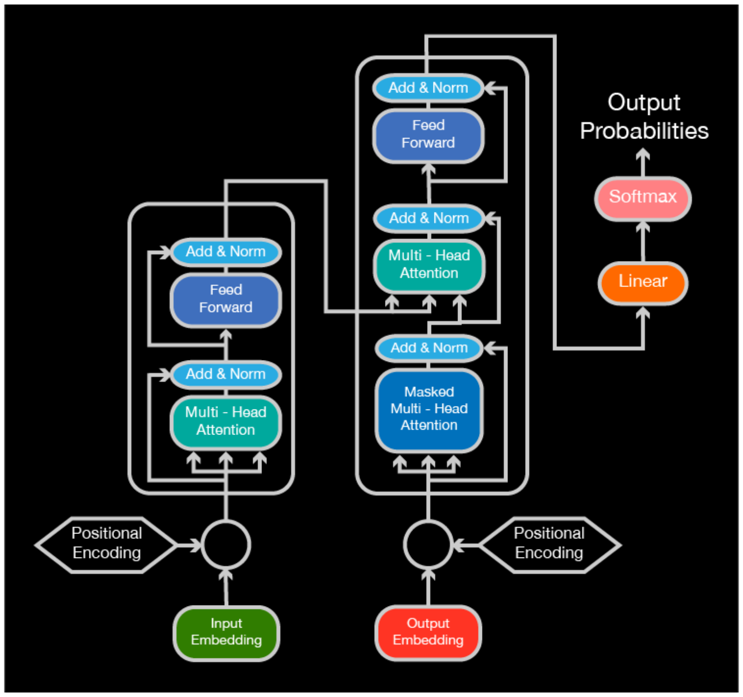


Figure 1: The Transformer Architecture

BERT

At its core BERT has a simple architecture, it consists of a stack of transformer encoders where instead of the final softmax output (in the diagram above) we connect the output to the input of the next encoder in the stack. The BERT-base model that we have used has 12 such units of the transformer encoder. It is important to note also that this model uses a feedforward network with 768 hidden units and 12 attention heads which give us a total of 110 million parameters for the BERT model.

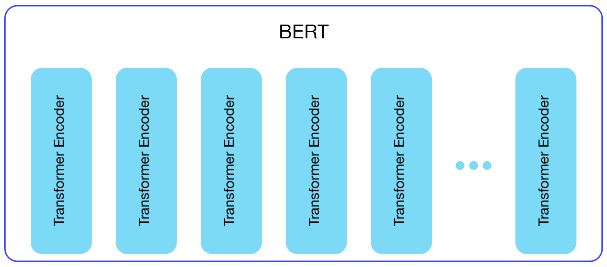


Figure 2: The BERT Architecture

This model takes in as input the text that we want to classify with a maximum sequence length of 512 among which a special [CLS] (classification) token is supplied. Each encoder layer outputs a vector of length 768 for each of the 512 positions, which is the number of units in the feed-forward layer. These outputs become input to the next encoder layer, this information flows forward through all the 12 layers. At the final layer only the vector of length 768 from the position where the special [CLS] token was passed to is used and passed on to the classification layer. Even using a single classification layer gives quite striking results that have been demonstrated in [3].

To fine tune BERT it is recommended to use a pre-trained model and add on a simple dense classifier which will provide the required outputs.

Our Model

Our model for both dataset is illustrated here, the only difference is that the dataset with 31 labels have an output space of the integers {0, 1, ..., 30} and the one with 20 labels has an output space of the integers {0, 1, ..., 19}. The

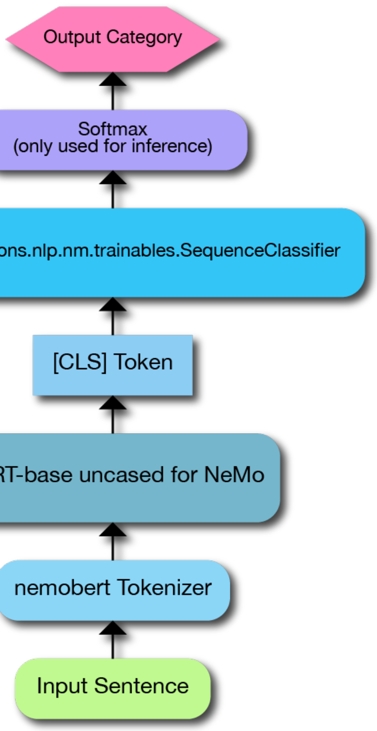

Output Category
Softmax
(only used for inference)
nemo.collections.nlp.nm.trainables.SequenceClassifier
[CLS] Token
BERT-base uncased for NeMo
nemobert Tokenizer
Input Sentence

Figure 3: Our Model

correspondence of the integers and the labels are given in tables 1 and 2 for the *places* and *issues* APIs respectively.

Results

Dataset from the *Places* API

On splitting the data into *train*, *test* and *dev* sets we had 17321, 4830 and 1941 examples respectively in each of these sets.

We also had 31 categories many of which overlap semantically as can be seen from table 1.

At once it is clear that since our training dataset is so small, there is a chance of overfitting, which was found to be the case. The cross validation accuracy was reported to be ~93% whereas the test accuracy was ~73%.

On using the model for single sentence classification a very poor performance was observed. The model failed to give correct outputs even when the label was put as an input sentence. This suggests that our model was too complex for the dataset.

All the required learning curves to show from tensorboard couldn't be fetched because all the training was done on remotely on the GPU machine.

I was only able to fetch the accuracy and loss curves for the training phase only which I produce here.

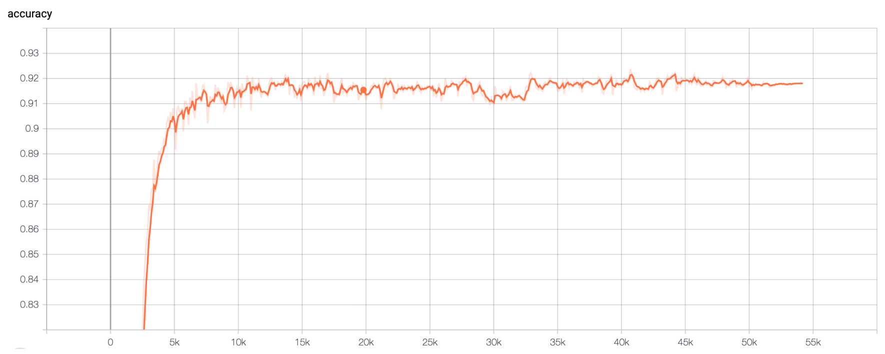


Figure 4: Accuracy curve for Training

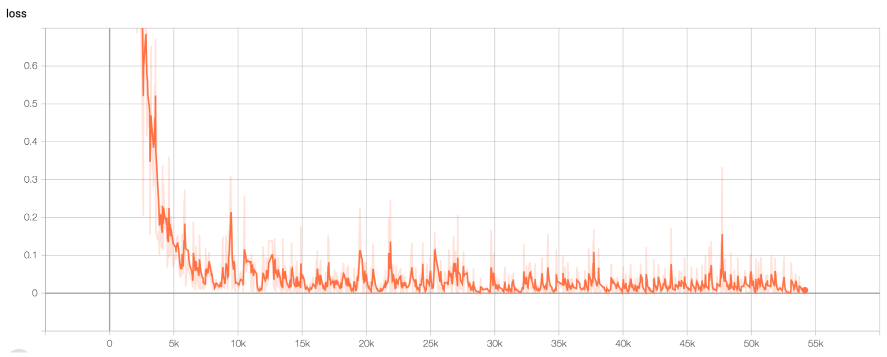


Figure 5: Loss Curve for Training

Dataset from the *Issues* API

On splitting the data into *train* and *dev* sets we had 59258 and 14828 examples respectively in each of these sets.

We had a smaller number of categories which did not have significant semantic overlap. We also have a large number of examples in the training and cross-validation (dev) sets.

On training our model gave us around 75% cross-validation accuracy.

Due to time constraints a proper train, test and cross-validation was not done, but single sentence classification was carried out with multiple queries for each category which mostly provided correct results. 15 our of 20 manually given examples were classified correctly, and even though this is too small of a sample size, this is much better than the previous model where only about 2 of 10 queries were classified correctly.

The final confusion matrix for 20 categories was found.

Label	Precision	Recall	F1-Score	Support
Yard and Structure Concern	0.72	0.69	0.71	1154
Drainage Issues	0.68	0.79	0.73	212
Noise Concern	0.67	0.65	0.66	142
Pothole	0.94	0.92	0.93	2858
Traffic Signs	0.80	0.79	0.80	350
Trees Issues	0.67	0.79	0.73	348
Street Light	0.78	0.86	0.82	292
Animal Issues	0.69	0.82	0.75	349
Illegal Dumping	0.78	0.77	0.77	2762
Parking - Enforcement	0.71	0.75	0.73	1531
Nuisance On Property	0.64	0.56	0.60	1035
Traffic Signals	0.86	0.87	0.86	320
Sidewalk	0.58	0.56	0.57	517
Abandoned Vehicle	0.53	0.56	0.55	669
Illegal Construction	0.54	0.65	0.59	150
Homeless Concerns	0.52	0.56	0.54	212
Graffiti	0.85	0.85	0.85	1469
Health Issues	0.47	0.32	0.38	112
Park Maintenance	0.55	0.46	0.50	287
Pests	0.71	0.81	0.76	58

Table 3: Confusion Matrix after Training for *Issues* API

Future Work

The most glaring problem right now is that we are discarding a lot of data. We might be able to use the data that we have discarded, currently we are searching for a way to do so. If we are able to boost the number of examples for the categories for which our model is performing poorly, we might be able improve the overall accuracy.

We also have only used a simple sequence classifier with only one hidden layer, we might experiment with more layers using cross validation.

There are other modern variants of BERT such as RoBERTa and ALBERT which might perform better, we will carry out these experiments.

All in all the results of our work look promising, and might perform at state of the art levels using a bit of data engineering and following careful software engineering principles.

Acknowledgement and References

[1] SeeClickFix and its numerous APIs through which its valuable data
[2] The folks at Nvidia NeMo for keeping its priceless libraries open sourced
[3] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding *by* Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova
[4] Nvidia NGC which provides such large scale pre-trained models without any cost
[5] Attention Is All You Need *by* Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin

I am also thankful for the excellent blog posts made by Jay Alammar and Andrej Karpathy the excellent course material that is freely available on YouTube from the folks at Stanford NLP : they made the path through NLP exciting and the field extremely welcoming.

I am extremely grateful to Purnendu Mukherjee for letting me be part of the work he is doing, I owe almost everything I have learnt about NLP in some way or other to him. Even when he does not directly show me the way he deals with me with patience and understanding, knowing just how much to push me so that I can get out of my comfort zone and find my own way.

And finally I want to thank Dripta Maharaj for pushing us so hard to do a hands on project which broadens our horizons and lets us learn things that are well outside the scope of a classroom.