Machine Learning Engineer Nanodegree

Capstone Project

Jami Daly
February 13th, 2019

I. Definition

Project Overview

Content Marketing is the process of creating an organic growth channel for your Business content. Content can range from Cat Videos, blog talks or funny pictures, all created to attract new potential customers. Content Creation can be very time taxing so creating content efficiently would be very beneficial to a Business.

Chatbots' central purpose is to facilitate interactions between products or services and customers. They connect with customers, can provide content and even facilitate a purchase. Chatbots are very scalable and personable. Since Messaging apps have surpassed social networking apps in usage, Companies need to engage customers through this medium in a cost and sales effective way. Chatbots satisfy both of those requirements!

Problem Statement

Modern Machine Learning techniques have not been applied fully to the chatbot domain however. Currently used techniques are Retrieval and Generative Models. Retrieval models analyze a user's input and then tries to match pre-defined answers to the question or statement posed. Generative models take the users input and tries to generate a response. This allows Generative Models to come up with answers they have never seen before making them more robust.

Generative Adversarial Networks are a technique that has not been widely applied to chatbot creation. GANs take a generative model discussed above and pits it against a discriminator model. The generative model is given input and generates an output. This output is fed into a discriminator which compares all of the inputs and tries to reject outputs it thinks are fake. Both are then updated based on if they got caught or if they caught the generated message. I believe this technique has not been applied to chatbots because the process of training a discriminator and generator from scratch

takes quite a long time. A way to overcome this problem is to pre-train a discriminator so it can catch bad generative outputs much easily.

I will attempt to solve this problem by creating a discriminator to be released so it can be used in a GAN as a discriminator. This will speed up the time it takes to get a GAN up and running since the discriminator will have a very good starting point. This means that the Generator side of the GAN will also see benefits due to false positives or true negatives that would hurt its training early. I am going to try and achieve this through utilizing different deep learning architectures.

The Ubuntu Dialogue Corpus will be used for the dataset in the creation of my proposed chatbot. This dataset was developed and released to provide a large dataset for dialog systems to facilitate breakthroughs in AI dialog systems. It will be used to train the chatbot discriminator agent to keep the "best" responses and try to locate the false or generated responses.

Metrics

The evaluation metrics to be used will be Recall@K and Precision@k. Measures the models ability to find the relevant documents while Precision@K measures the models ability to reject non relevant documents. If the model can both choose the correct text given the context, and reject generated text given the context, then it is a great discriminator for our generator. Both measures will be used in determining the models usefulness since if only one is used, potentially a bias model can become the optimal choice. Using the F score I think is the optimal choice and combination of the two. The mathematical Representation is as follows:

$$\mathcal{F} = \frac{1}{\frac{1}{2}\left(\frac{1}{\mathcal{P}} + \frac{1}{\mathcal{R}}\right)} = \frac{2 \times \mathcal{P} \times \mathcal{R}}{\mathcal{P} + \mathcal{R}}$$

II. Analysis

Data Exploration

In this section, you will be expected to analyze the data you are using for the problem. This data can either be in the form of a dataset (or datasets), input data (or input files), or even an environment. The type of data should be thoroughly described and, if possible, have basic statistics and information presented (such as discussion of input

features or defining characteristics about the input or environment). Any abnormalities or interesting qualities about the data that may need to be addressed have been identified (such as features that need to be transformed or the possibility of outliers). Questions to ask yourself when writing this section:

The Ubuntu Corpus Dataset was created to help advance the AI chatbot space by providing a large robust dataset for public use. It's almost one million one on one conversations were collected from the Ubuntu help chat. This means it is not a general. The Ubuntu Dialogue Corpus will be used for the dataset in the creation of my proposed chatbot. This dataset was developed and released to provide a large dataset for dialog systems to facilitate breakthroughs in AI dialog systems. It will be used to train the chatbot agent to select the best answer to questions posed by users.

A black mark against the dataset is that it was extracted from a chat room, meaning that multiple conversations may have been going on at one time. This means that there will be some amount of noise but since the chats were moderated, the conversations follow a similar flow. A new user joins, asks a question and a more experienced user answers and mentions the new users name. This helped with the extraction of person to person dialogs in a multi person to multi person environment. Another concern I have is about how much long the utterance and context sentences are. If the dataset has very short sentences for context and utterances short as well this could be problematic for our machine learning model to learn. In our data visualization we will have to investigate the description of the length of both context and utterance.

The dataset is appropriate for the given problem because it includes good and bad responses given the context allowing a pre-trained model to be created. It includes "noise" responses so the model has to select the correct response out of the noise (2). The input format of the data is utterance, context, and flag. The utterance is the input to be evaluated, the context is the previous statements and responses in the conversation and the flag is a training label 1 for it is a correct response and 0 for an incorrect response. This is our prediction variable.

A sample of the data is provided below:

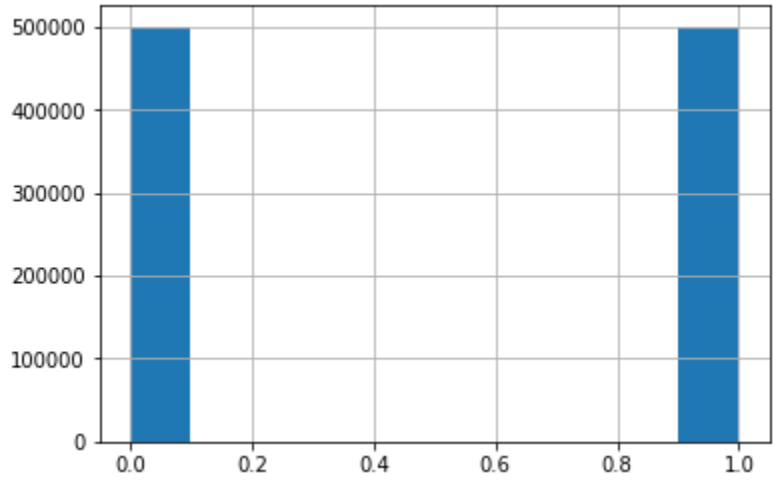
```
In [12]: pd.options.display.max_colwidth= 1000
         data.head()
```

Out[12]:

	Context	Utterance	Label
0	i think we could import the old comment via rsync , but from there we need to go via email . i think it be easier than cach the status on each bug and than import bite here and there __eou__ __eot__ it would be veri easi to keep a hash db of message-id __eou__ sound good __eou__ __eot__ ok __eou__ perhap we can ship an ad-hoc apt_preferec __eou__ __eot__ version ? __eou__ __eot__ thank __eou__ __eot__ not yet __eou__ it be cover by your insur ? __eou__ __eot__ yes __eou__ but it 's realli not the right time : / __eou__ with a chang hous upcom in 3 week __eou__ __eot__ you will be move into your hous soon ? __eou__ post a messag recent which explain what to do if the autoconfigur doe not do what you expect __eou__ __eot__ how urgent be # 896 ? __eou__ __eot__ not particular urgent , but a polici violat __eou__ __eot__ i agre that we should kill the -novtswitch __eou__ __eot__ ok __eou__ __eot__ would you consid a packag split a featur ? __eou__ __eot__ context ? __eou__ __eot__ spli...	basic each xfree86 upload will not forc user to upgrad 100mb of font for noth __eou__ no someth i do in my spare time . __eou__	1
1	i 'm not suggest all - onli the one you modifi . __eou__ __eot__ ok , it sound like you re agre with me , then __eou__ though rather than `` the one we modifi '' , my idea be `` the one we need to merg '' __eou__ __eot__	sorri __eou__ i think it be ubuntu relat . __eou__	0
2	afternoon all __eou__ not entir relat to warti , but if grub-instal take 5 minut to instal , be this a sign that i should just retri the instal :) __eou__ __eot__ here __eou__ __eot__ you might want to know that thinic in warti be buggi compar to that in sid __eou__ __eot__ and appar gnome be suddent almost perfect (out of the thinic problem) , nobodi report bug : -p __eou__ i do n't get your question , where do you want to past ? __eou__ __eot__ can i file the panel not link to ed ? :) __eou__ __eot__ be you use alt ? or the window key ? __eou__ wait for the gnome-them , compon will be ad __eou__ __eot__ i just restart x and now nautilus wo n't show the desktop : (__eou__ hal be n't start : (__eou__ __eot__ do you think we have ani interest to have hal support turn on in gnome-vf at this point ? it increas the sourc of problem for no real benefit imho ... __eou__ __eot__ be it a know bug that g-s-t doe n't know what distribut it run on ? __eou__ be there ani chang to desktop...	yep . __eou__ oh , okay . i wonder what happen to you __eou__ what distro do you need ? __eou__ yes __eou__	0
3	interest __eou__ grub-instal work with / be ext3 , fail when it be xfs __eou__ i think d-i instal the relev kernel for your machin . i have a p4 and it instal the 386 kernel __eou__ holi crap a lot of stuff get instal by default :) __eou__ you be instal vim on a box of mine __eou__ ;) __eou__ __eot__ more like osx than debian ;) __eou__ we have a select of python modul avail for great justic (and python develop) __eou__ __eot__ 2.8 be fix them iirc __eou__ __eot__ pong __eou__ vino will be in __eou__ enjoy ubuntu ? __eou__ __eot__ tell me to come here __eou__ suggest thursday as a good day to come __eou__ __eot__ we freeze version a while back :) __eou__ you come today or thursday ? __eou__ we re consid shift it __eou__ yay __eou__ enjoy ubuntu ? __eou__ usplash ! __eou__ __eot__	that the one __eou__	1

Exploratory Visualization

In the following visualization, I am displaying the distribution of correct utterances to noise utterances. I decided to use the default 50/50 split between the two. This distribution seemed to be the best for training.



Next let's look at the context length for the entire dataset. This will tell us how long the context

```
count    1000000.000000
mean          86.339195
std           74.929713
min            5.000000
25%           37.000000
50%           63.000000
75%          108.000000
max         1879.000000
Name: Context, dtype: float64
```

From these results we can observe that the context lengths has a mean of 86. This means that most examples have a context of around 86 words. This is a very good indicator that our model will have enough information to approximately categorize the utterance given the context.

Now let's take a look at Utterance Length.

```
(data.Utterance.str.split(" ").apply(len)).describe()
```

```
count    1000000.000000
mean          17.246392
std           16.422901
min            1.000000
25%            7.000000
50%           13.000000
75%           22.000000
max          653.000000
Name: Utterance, dtype: float64
```

From these results, we can gather that the mean utterance length is around 17 words. Again this is a good indicator that the utterances aren't too short to do analysis on. If the mean length was 2 we would have difficulty applying this model anywhere but here and it would be prone to over fitting on such a small input space.

Both of these visuals put our earlier worries at ease and address our concerns.

Algorithms and Techniques

The main algorithm I am going to leverage is Neural Networks. I believe Neural Networks, specifically recurrent Neural Networks will be a great choice for categorizing text generation. Since Recurrent Neural Networks take into account previous context, they are a perfect choice for conversation processing since conversation utterances are based on previous statements. For example, if I say “Hey when do you want to go to the movies?”, “Today” is a perfectly good response. If however I said “How’s the weather?” and the response was “Today” that should be flagged as illegitimate. Since RNNs take context into account, they are a perfect choice of a model.

There are problems with basic RNNs though, mainly with gradient. A gradient measures the change is all of the weights with regard to the change in error. What this basically means is that if the gradient is higher(above zero) the model is still learning but when the gradient approaches zero, weights remain unchanged meaning that the model doesn’t learn anything. RNNs are known for this gradient vanishing problem, but to solve this we will utilize LTSM networks, which are Long Term Short Term Networks. LTSMs utilize “gates” to decide whether or not to keep, forget or use new input. This keeps the gradient high enough for the model to train fast and have high accuracy.

Benchmark

Term Frequency Inverse Document Frequency has been used widely to determine how important a word is in a document and information retrieval. It takes the words in the input and compares them to the words in the outputs to score how important each word is. Google used to use it in their search algorithm to match searches to their related documents. I propose using this model as a benchmark to determine if my own is “successful”. To test the successfulness of this model I will be using Recall@K.

The following results indicate the results of applying a tdif vectorizer on our results. We hope that these results will be a little better than if we chose an answer at random.

```
(deeplearning) C:\Users\JD550T\Documents\MachineLeaningDegree\Capstone Project\SubmissionFolder\codebase>python TDIFExample.py
Recall @ (1, 10): 0.495032
Recall @ (2, 10): 0.596882
Recall @ (5, 10): 0.766121
Recall @ (10, 10): 1
```

As you can see the TDIF vectorizer does better than choosing an answer at random(10% recall@10) but still not where a good model should be.

III. Methodology

Data Preprocessing

The Ubuntu Corpus Dataset comes with the ability to lemmatize, stem and tokenize the dataset which will be taken advantage of. "The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form" (1). This means that words such as organize, organizes and organizing is treated as one word as well as democracy, democratic and democratization will be treated as one word as well. A tokenizer will also be applied to the data to separate the words so the input can have lemmanization and stemming applied to it.

A dictionary will also be created and saved to map words to integers, making the sentences a sequence of ids. This turns a sentence into a vector which will be easier to for the model to train on. I will be utilizing the GLOVE which is the global vectors for word representations. I chose Glove over wordToVec since it is easier to parallelize the implementation which means it's easier to train over more date. Both the dictionary and the glove representation of the words will be saved in text files to be referenced and used later in the implementation.

The test data will have to be altered since the original intent of this dataset was to be used in context for selecting the most appropriate response. Instead of this, we will be trying to distinguish which response does not match the context. To do this we will be splitting the Train data into a 70, 30 split and creating our own test.csv based on the original test.csv. This will contain 10 context, response pairs with on being a "distractor" or a response that does not match the context.

Implementation

The initial solution was derived from the boilerplate code and used to create a Dual Encoder RNN. The boiler plate code was implemented in Tensorflow which we did not cover much so I implemented it in Keras. The boiler plate code helped guide me thought an implementation of tensorflow Dual Encoder RNN. This helped me get familiar with terminology and implementation of a Dual Encoder RNN.

I then moved onto implementing the dual encoder in keras. It is in these days that I truly appreciated how easy Keras is to use vs Tensorflow. After looking back at old deeplearning projects, and discovered an imdb project demo. The implementation was based on this project and altered to fit my needs. Keras documentation lead me to use 2 merged sequential models with LSTM cells. This allowed me to have both the response and context as inputs. We then have a Dense output layer with a sigmoid activation function. The most applicable loss function is binary cross entropy. This loss function

measures the performance of a model whose output is a probability. Since we are scoring the probability a response matches to a given context, this is a perfect loss function to score our model.

My coding is very straightforward so I do not believe that much documentation is needed besides what is already provided. The main program is keras_dual_encoder.py, with the benchmark model, is located in TDIFExample.py. The evaluation metric evaluation is located in evaluation.py and calculates recall@k (1, 2, 5, 10).

Refinement

My initial solution was a very poor initial run of tensorflow which failed miserably. The results were so bad that I had to switch to Keras using backend because there was something very wrong about my initial implementation. My model was incredibly slow in training and had poor results to match.

I then moved onto my current and final keras implementation. After running a couple of rounds with the model I saved it and tested to make sure I was headed in the right direction. The results are as follows:

```
10: {1: 0.2720401691331924, 2: 0.46881606765327694, 5: 0.8005285412262156}}
```

I then tried to increase my hidden units, and runs to improve on this result. After letting my computer run all night on a higher hidden layer count for my LSTM Cell, it did not finish ½ of a run. This led to the compromise with keeping 20 hidden layers for my LSTM Cells but increasing the run count. Here there was also a decision point to either cut down on the amount of data used for training, or keep the data. After discussing this with peers, the data was kept so the model would not memorize the data, or overfit on a subset, and decrease the performance on testing data it had not seen before. This did not perform much better than the benchmark so I had to go back to the drawing board.

I eventually settled on a model I ran for 2 days straight and due to time constraints, I had to stop it early. After increasing the hidden size to 50, the accuracy was much better after just 5 epochs. The final results are below. Notice the model performing better on recall@5 showing high potential to be trained more.

```
recall@1 0.3884249471458774
recall@2 0.5853594080338267
recall@5 0.8723572938689218
```

IV. Results

Model Evaluation and Validation

The model does align with the solutions expectations. This model is a good starting "pre-trained" discriminator, which will greatly improve the training time and results of a GAN network. The final parameters of the model are appropriate for the scope of the problem. Although model accuracy may have improved if the number of hidden layers of the LTSM was increased, the training time tradeoff would have not been worth it. Instead, providing a pre-trained model that can be further trained during the GAN process was the goal.

The test set was evaluated on over 200,000 never before seen examples. This means that the model has been tested on various inputs to make sure that it will generalize well to different input. Small changes in the training data are also not noticeable since the training data consists of 1,000,000 samples. Let's say a small change would be to alter 1000 examples or 10,000 examples. This would be less that .1% or 1% respectively. This extensive training and testing means that these results from the model can be trusted.

Justification

Final Solution Results:

```
recall@1 0.3884249471458774
recall@2 0.5853594080338267
recall@5 0.8723572938689218
```

Benchmark Results:

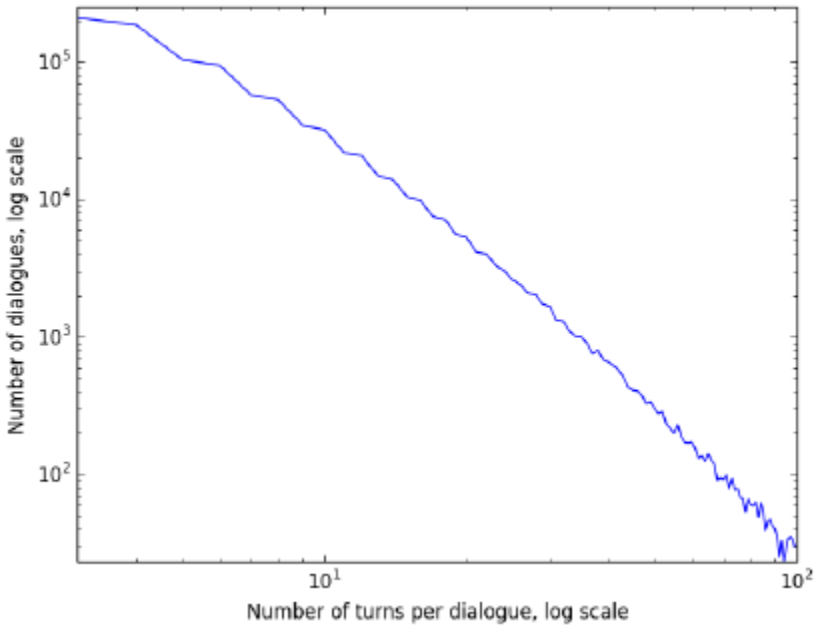
```
(deeplearning) C:\Users\JD550T\Documents\MachineLeaningDegree\Capstone Project\SubmissionFolder\codebase>python TDIFExample.py
Recall @ (1, 10): 0.495032
Recall @ (2, 10): 0.596882
Recall @ (5, 10): 0.766121
Recall @ (10, 10): 1
```

As you can see, the model performs better than the TDIF benchmark in the recall@5 case. I believe this final solution has improved on the reacall@k above the naïve model enough to be a great starting point for a GAN and therefore solved the problem. The benchmark cannot be retrained in the GAN network and therefore my solution will be better for retraining and robustness.

V. Conclusion

Free-Form Visualization

An important quality of the dataset that I did not discuss is the number of turns present in the conversation data.



This was taken from the Ubuntu Dataset Corpus Paper (2).

Turn Number is the amount of "turns" taken in a conversation. Unless you are talking to someone shy or overtalkative, each person in a conversation takes a turn saying something. This is important because what is desired from a generative text ML algorithm is a text generator who is able to generate a lot of turns. For example, we want our generative algorithm trained on this dataset encouraged to take atleast 3 turns. If the bot is trained on 1 or 2 turn conversations(discarded from this dataset), then it will learn these "habits" and not able to hold a conversation. Our discriminator also needs to handle a lot of turns(or context) so it can discriminate against generators who can't hold a conversation.

Reflection

The process started with research into what problems existed in the chatbot space. After researching different techniques extensively, I recognized was that GAN networks worked very well when applied to generative problems. GAN networks involve a discriminator and a generator who play cat and mouse, the generator trying to fool the discriminator. I thought this was a very interesting approach to chatbot speech generation. After researching more into GAN networks, it takes a very long time for them to produce anything very valuable. To solve this problem, a technique of pre-

training the discriminator has been used to speed up the process. This has been applied to different spaces but not chatbots. This inspired me to set out to create a publically available pre-trained discriminator using the largest dataset available for creating generative text models.

I set out to create a dual encoder from scratch, but this seemed to be out of the scope of this project. I spent several weeks researching how to create a RNN and trying to implement one. I ran into several roadblocks, since these algorithms were not covered in the course. Eventually I found some boiler plate code to base my program on so from there the project went more smoothly. I believe my solution fits the expectation for the problem with was to provide the public with a pre-trained model to begin their text generation GAN. This will speed up the GAN process.

The limitations of the equipment available was another difficulty. Due to the small memory of my computer and the complexity of this problem, running a single epoch of my model took around 10 hours. I did not foresee this problem so it was very difficult to change a variable and start another run without the process taking a week.

Improvement

The current implementation could be abstracted and generalized much better than the current implementation. There are a lot of path files, and while trying to get a better understanding of keras in general, I saw many ways I could generalize my code. For example, instead of having path variables for train and test specific to my machine, I could use os.join.path() functions so users of this can pass in their own datasets in. This would make the code reusable to train on another dataset set up the same way.

Along the same generalization improvements, flag variables could have been created. Instead I have a lot of “magic numbers” denoting layers, input size, and other random aspects. Instead flag variables could have been setup to make the code more readable and give the user the ability to change these variables from the command line.

If I used my final solution as a new benchmark, I do believe there is an even better solution. It would require a more powerful computer, and more time for training to complete. As stated before, if the architecture could include more LSTM cells in the encoders, I believe it would produce better recall@k results. Also adding more hidden layers to these LSTM cells might improve results as well.

Project References

(1) https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html
(2) https://arxiv.org/pdf/1506.08909.pdf
(3) http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/