Chatbot Using Trax

Uday Nayak, Leyton D’silva, Valentine Miranda and, Aaron Lobo

Department of Information Technology, Don Bosco Institute of Technology, Mumbai, India

Abstract- In certain situations when human interaction is not possible, having something that can substitute for human interaction is also complementary and vital. Therefore, our project aims to develop a multi-domain voice-enabled chatbot to answer queries in a more human-like manner while maintaining extended conversations. The technologies used in this project are trax, fast math, attention, and a reformer model to implement this system.

Index Terms- Artificial Intelligence, Natural language processing, chatbot, Neural network, Deep Learning, Trax.

I. INTRODUCTION

Chatbots can be classified using different parameters: the knowledge domain, the service provided, the goals, the input processing and response generation method, the human-aid, and the build method.

The users' expectations are rising, and they want everything available at the moment. Therefore, chatbot technology helps the users in almost all ways, from satisfying them to answering their queries and offering full assistance. Chatbots are no longer mere assistants, and their way of interacting brings them closer to users as friendly companions.

A chatbot is a program capable of communicating with users. It acts as a gateway to a service. Sometimes it is powered by machine learning. More commonly, driven using intelligent rules (i.e., if the person says this, respond with that). The services that a chatbot can deliver are diverse. From important life-saving health messages, checking the weather forecast or purchasing a new product, and anything else. The term chatbot is synonymous in text conversation but is proliferating through voice communication. E.g., "Alexa, what time is it?" The chatbot can talk to the user with the help of various channels, like Facebook Messenger, Siri, Slack, Skype, and many more. Consumers use messaging applications (more than social media applications). Therefore, there are popular ways for companies to deliver chatbot experiences to consumers.

II. AIM

The aim is to design a multi-domain voice-enabled chatbot to answer queries in a more human-like manner while maintaining extended conversations.

III. RELATED WORKS

A. TOWARDS UNIVERSAL DIALOGUE STATE TRACKING, BY LILIANG REN, KAIGE XIE, LU CHEN, KAI YU

In this paper, the dialogue system gathers all the information and keeps track of the user's goal at each dialogue turn. However, the current dialogue state trackers usually have several limitations, like not handling the situation where slot values are dynamically changing or having a higher number of model parameters with every new slot added.

It limits the ability of these systems to scale to large dialogue domains. Thus, the authors suggest a universal dialogue state tracker StateNet, which overcomes these limitations and significantly outperforms the state-of-the-art models but lacks multi-language support and cannot be customizable.

B. SEMANTIC PARSING FOR TASK ORIENTED DIALOG USING HIERARCHICAL REPRESENTATIONS, BY SONAL GUPTA, RUSHIN SHAH, MRINAL MOHIT, ANUJ KUMAR, MIKE LEWIS

This paper proposed a hierarchical annotation scheme for semantic parsing that allows the representation of compositional queries efficiently and accurately parsed by standard constituency parsing models. They released a dataset of 44k annotated queries and show that parsing models outperform sequence-to-sequence approaches on this dataset, but it lacks multi-lingual support and has higher latency.

IV. NEED FOR THE PROPOSED SYSTEM

Chatbots have potential. Even though increasingly used, the modern chatbot is still a young technology. With the continuing development of A.I., the potential for bots in business and personal lives is unlimited.

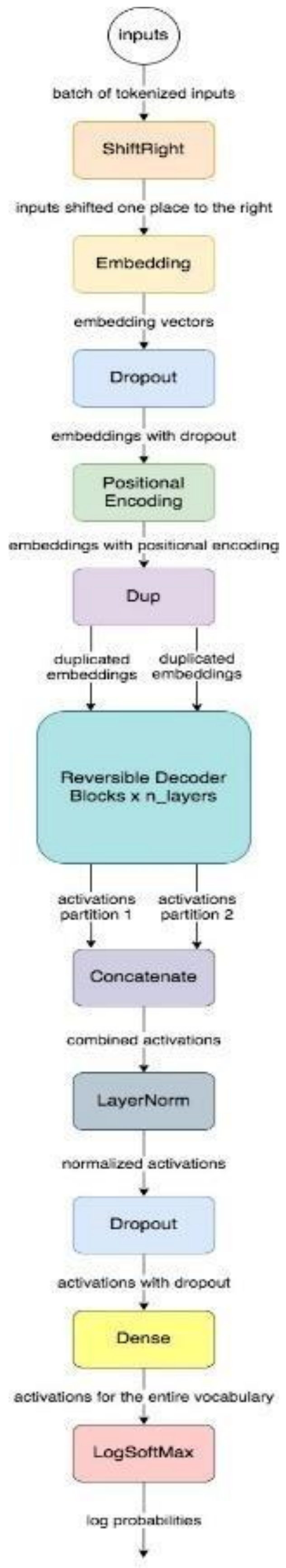
Chatbots emphasize the company's brand and image. The chatbot represents the company when it communicates with the customer, so it is a perfect embodiment of brand building from a marketing point of view.

They offer specific services. A well-optimized chatbot communicates only the essentials and does not overwhelm the user.

Chatbots automate processes. Bots can take on human work for, generally speaking, mundane or basic analytic tasks.

V. SYSTEM ARCHITECTURE

The Reformer architecture is a Transformer model designed to process longer data sequences efficiently (up to 1 million words in a language processing). The execution of Reformer requires lower memory consumption for achieving impressive performance even when running on a single GPU. It addresses three primary sources of memory consumption in the Transformer while improving how the Reformer model can handle up to one million words of context windows efficiently, all on a single accelerator and using only 16 Gigabytes of memory. In a nutshell, the model combines two techniques to solve attention and memory allocation problems: locality-sensitive-hashing (LSH) to reduce the complexity of attending over long sequences and reversible residual layers to more efficiently use the memory available.



5.1 Dot-product Attention

Scaled dot-product attention is the standard attention used in the Transformer, where the input consists of keys and queries of dimension dk, and values of dimension dv. The dot products are computed, scaled by √ dk, and a softmax function is applied to get the values' weights. The attention function is computed simultaneously on queries, packed together into a matrix Q in practice. Assuming the keys and values are packed together into matrices K and V, the matrix of outputs is defined as:

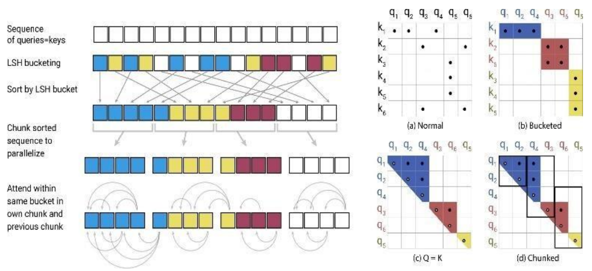
$$Attention(Q,V,K) = (\frac{QK^T}{\sqrt{dk}})V$$

The self-attention dot product grows as the size of the input is squared. For example, if one wished to have an input size of 1024, that would result in 1024^2or over a million dot products for each head! Therefore, it has resulted in significant research related to reducing the compute requirements. One such approach is Locality Sensitive Hashing (L.S.H.) Self Attention.

5.2 Locality-Sensitive Hashing Attention

L.S.H. Self-attention uses Queries only, no Keys. Attention then generates a metric of the similarity of each value of Q relative to all the other values in Q. Further. Multiple random hashes can improve the chances of finding similar entries. It is the approach taken here, though the hash is implemented a bit differently. The values of Q are hashed into buckets using a randomly generated set of hash vectors. Multiple sets of hash vectors are used, generating multiple hash tables. In the figure above, we have three hash tables with four buckets in each table. Notionally, following the hash, Q's values have been replicated three times and distributed to their appropriate bucket in each of the three tables. To find similarity, then, one generates dot-products only between members of the buckets. The result of this operation provides information on which entries are similar. As the operation is distributed over multiple hash tables, the results need to be combined to form a complete picture used to generate a reduced dot-product attention array. It is clear that because we do not compare every value vs. every other value, Dots' size is reduced.

The challenge in this approach is getting it to operate efficiently. It will operate poorly on a vector processing machine such as a GPU or T.P.U. Ideally, operations are done in large blocks with uniform sizes. While it is straightforward to implement the hash algorithm this way, it is challenging to manage buckets and variable-sized dot- products.



For a single query position i at a time:

$$o_i = \sum_{j \in P_i} \exp\left(q_i \cdot k_j - z(i, P_i)\right) v_j$$

$$\text{where } P_i = \{j : i \geq j\}$$

We introduce the notation Pi to represent the set that the query at position i attends to, and z to denote the partition function (i.e., the normalizing term in the softmax). For clarity, we also omit to scale $\sqrt{dk}$.

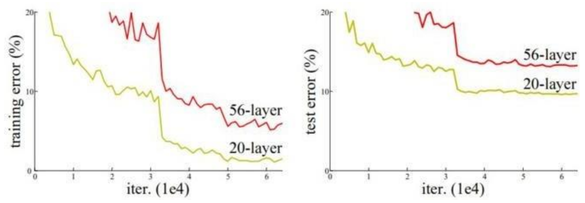
For batching purposes, we typically perform attention over a larger set $\tilde{P}i = \{0, 1, \ldots, l\} \supseteq Pi$ while masking out elements, not in Pi:

$$o_i = \sum_{j \in \tilde{P}_i} \exp\left(q_i \cdot k_j - m(j, P_i) - z(i, P_i)\right) v_j$$

$$\text{where } m(j, P_i) = \begin{cases} \infty & if\ j \notin Pi \\ 0 & otherwise \end{cases}$$

5.3 ResNet

For solving a complex problem, we stack some additional layers in Deep Neural Networks. It results in improved accuracy and performance. The intuition behind adding them is that they progressively learn more complex features. For example, for recognizing images, the first layer learns to detect edges. The second learn to identify textures. Finally, the third layer will learn to detect objects. Nevertheless, there exists a maximum threshold for depth with the traditional Convolutional neural network model. Let us look at a plot that describes error% on training and testing data for a 20-layer Network and 56 layers Network.



We can see that the error percentage for 56-layer is more than a 20-layer network in both cases of training data and testing data. It suggests that with adding more layers on top of a network, its performance degrades. It could be blamed on the optimization function, initialization of the Network, and vanishing gradient problem. One might be thinking that it could be a result of overfitting too. However, the error percentage for a 56-layer network is worst on both training and testing data which is not the case for an overfitting model.

The problem of training deep networks has been alleviated with ResNet or residual networks' introduction, and these ResNets are made up of Residual Blocks.

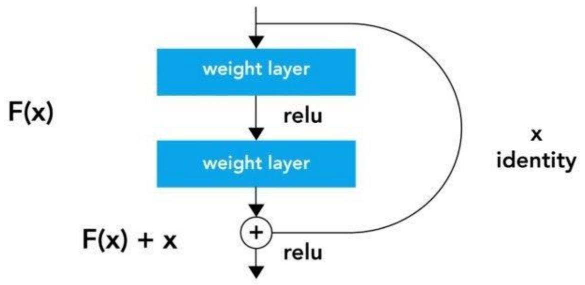


Figure 2. Residual learning: a building block.

The first thing to notice is a direct connection that skips some layers (may vary in different models) in between. Such a connection is called 'skip connection' and is the core of residual blocks, due to which the output of the layer is different. Without using the skip connection, the input 'x' is multiplied by the layer's weights, followed by a bias term.

Next, it goes through the activation function, f(), and we get our output as H(x).

$$H(x) = f(wx + b)$$

Or

$$H(x) = f(x)$$

Now, after introducing skip connection, the output is changed to

$$H(x) = f(x) + x$$

A slight problem with this approach is when the input dimensions vary from that of the output, resulting in convolutional and pooling layers. However, when dimensions of f(x) are different from x, two approaches can be considered:

A skip connection is padded with extra zero entries, resulting in the dimensions increase.

The projection method used to match the dimension is done by adding a 1×1 convolutional layer to input.

The output is:

$$H(x) = f(x) + w1.\, x$$

Here, an additional parameter w1 is added with no additional parameter for the first approach.

Here we add parameter w1, whereas no additional parameter is required when using the first approach.

Skip connections in ResNet help solve vanishing gradient in deep neural networks, allowing the gradient's alternate shortcut path to flow through. These connections help by allowing the model to learn the identity functions, ensuring that the higher layer will perform similarly to the lower layer.

Let us explain this further.

Say we have an external network and a deep network that maps an input 'x' to output 'y' using H(x). The deep Network must perform as well as the shallow Network without degrading the performance observed in plain neural networks (without residual blocks). It can be achieved if the additional layers in a deep network learn the identity function. Thus, their output equals inputs which eliminates them to degrade the performance even with extra layers.

Moreover, the residual blocks make it exceptionally easy for layers to learn identity functions. From the formulas above, it is evident that in plain networks, the output is

$$H(x) = f(x) + x$$

So, in order to learn an identity function, f(x) must be equal to x, which is grader to attain, whereas in-case of ResNet, which has output:

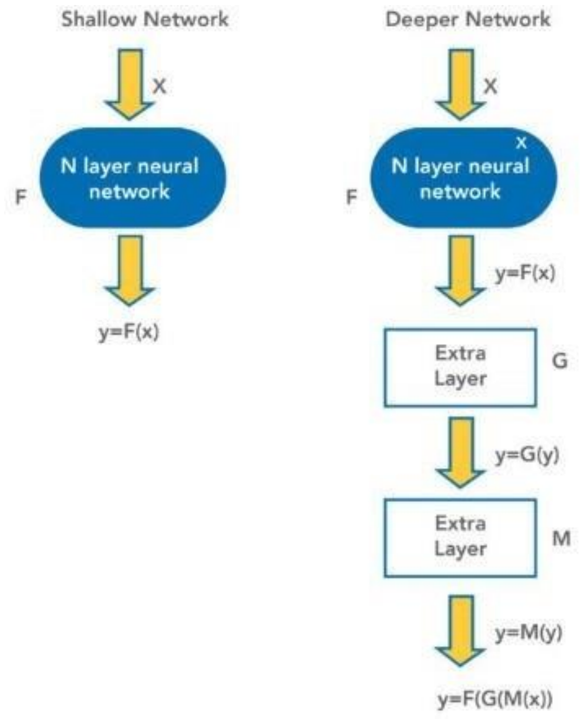
$$H(x) = f(x) + x,$$

$$f(x) = 0$$

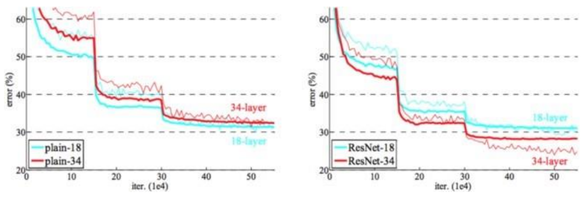
$$H(x) = x$$

All that is needed is to make f(x)=0, which is more manageable, which leads to x as output which is also the input.

In a best-case scenario, additional layers of the deep neural Network can better approximate the mapping of 'x' to output 'y' than the shallower counterpart and reduce the error by a significant margin. Furthermore, ResNet is expected to perform equally or better than plain deep neural networks. Using ResNet has resulted in a significant enhancement in the performance of neural networks with more layers.



G and M act as Identity Functions. Both the Network Give same output



The difference is vast in the networks with 34 layers where ResNet-34 has a much lower error percentage when compared to plain-34. Also, on further observation, the error percentage for plain-18 and ResNet-18 is almost the same.

5.4 RevNet

Reversible Residual Network (RevNet) is a ResNet variant where each layer's activations can be reconstructed exactly from the subsequent layers. Therefore, activations for most layers need not be stored in memory during backpropagation. It results in a network architecture whose activation storage requirements are independent of depth and typically an order of magnitude smaller than equally sized ResNets

RevNet consists of a series of reversible blocks. Units in each layer are divided into two groups, denoted $x1$ and $x2$; the authors find what works best is partitioning the channels. Each reversible block takes inputs ($x1$,

$x2$) and produces outputs ($y1$, $y2$) according to the following additive coupling rules – inspired the transformation in NICE (nonlinear independent components estimation) – and residual functions F and G analogous to those in standard ResNets:

$$y1 = x1 + F(x2)$$

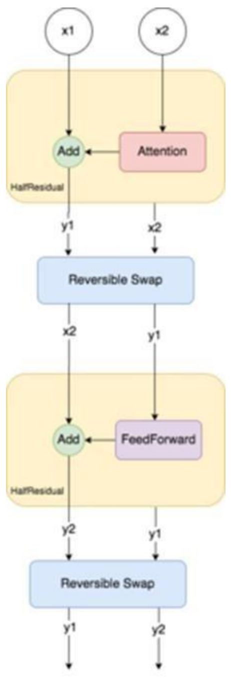
$$y2 = x2 + G(x1)$$

It is possible to reconstruct each layer's activations from the next layer's activations as follows:

$$x2 = y2 - G(y1)$$

$$x1 = y1 - F(y2)$$

Unlike residual blocks, reversible blocks must have a stride of 1; otherwise, the layer discards information and cannot be reversible. Standard ResNet architectures have a handful of layers with more significant stride. When defining a RevNet architecture analogously, the activations must be stored explicitly for all non-reversible layers.



The above image refers to the reversible decoder blocks. These reversible decoder blocks are used in the Reformer to improve memory efficiency.

VI. CONCLUSION

Trax is a deep-learning library focused on clear code and speed and is actively used and maintained by the Google Brain team. It includes basic models (like ResNet, LSTM, Transformer, and R.L. algorithms (like REINFORCE, A2C, P.P.O.). It is actively used for research and includes new models like the Reformer, and new R.L. algorithms like A.W.R. Trax have bindings to many deep learning datasets, including Tensor2Tensor TensorFlow datasets. On training the chatbot using the multiwoz Cambridge dataset, we obtained steady results.

VII. ACKNOWLEDGEMENT

We want to thank our Principal, Dr. Prasanna Nambiar, and the H.O.D Prof. Janhavi Baikerikar of Don Bosco Institute of Technology for letting us carry out this project.

VIII. REFERENCES

[1]. Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. Character-level language modeling with deeper self-attention. CoRR, abs/1808.04444, 2018. URL http://arxiv.org/abs/1808.04444

[2]. Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya P. Razenshteyn, and Ludwig Schmidt. Practical and optimal L.S.H. for angular distance. CoRR, abs/1509.02897, 2015. URL http://arxiv.org/abs/1509.02897

[3]. Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. arXiv preprint arXiv:1607.06450, 2016. URL http://arxiv.org/abs/1607.06450

[4]. Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. Large-scale simple question answering with memory networks. CoRR, abs/1506.02075, 2015. URL http://arxiv.org/abs/1506.02075.

[5]. Sarath Chandar, Sungjin Ahn, Hugo Larochelle, Pascal Vincent, Gerald Tesauro, and Yoshua Bengio. Hierarchical memory networks. arXiv preprint arXiv:1605.07427, 2016.

[6]. Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. URL https://openai.com/blog/sparse-transformers, 2019.

[7]. Felix Hill, Antoine Bordes, Sumit Chopra, and Jason Weston. The goldilocks principle: Reading children's books with explicit memory representations. CoRR, abs/1511.02301, 2015. URL http://arxiv.org/abs/1511.02301 for Computational Linguistics. URL https://www.aclweb.org/anthology/ W18-6319.

[8]. Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Herve´ Jegou. Large memory layers with product keys. ´ CoRR, abs/1907.05242, 2019. URL http: //arxiv.org/abs/1907.05242.

[9]. Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating Wikipedia by summarizing long sequences. CoRR, abs/1801.10198, 2018. URL http://arxiv.org/abs/1801.10198.

[10]. Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. Scaling neural machine translation. In Proceedings of the Third Conference on Machine Translation: Research Papers, pp. 1–9, Brussels, Belgium, October 2018. Association for Computational Linguistics. DOI: 10.18653/v1/W18-6301. URL https://www.aclweb.org/anthology/W1 8-6301.

[11]. Matt Post. A call for clarity in reporting BLEU scores. In Proceedings of the Third Conference on Machine Translation: Research Papers, pp. 186– 191, Belgium, Brussels, October 2018. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/ W18-6319.

[12]. Matt Post. A call for clarity in reporting BLEU scores. In Proceedings of the Third Conference on Machine Translation: Research Papers, pp. 186–191, Belgium, Brussels, October 2018. Association