A Generative Chatbot with NLP

David Liebman david.c.liebman@gmail.com

January 11, 2020

Coordinator 1 Coordinator 2

Department of Computer Science

SUNY New Paltz

Abstract

We experiment with the Transformer Neural Network model and we try to explain in this paper how one works.

We chronicle a few experiments in Natural Language Processing. We try a GRU based chatbot. We try a transformer based chatbot. We also try a GPT2 based chatbot.

We are also interested in installing the chatbot code on a small computer like a Raspberry Pi with speech recognition and speech-to-text software. In this way we might be able to create a device which speaks and which you can speak to. For the GRU based model we can use a Raspberry Pi 3B. In the case of GPT2 the running chatbot model uses too much ram, so we may try to install it on a Raspberry Pi 4B. GPT2 does not fit on a Raspberry Pi 3B.

Contents

1 Background/History of the Study **1**

1.1 Background 2

2 GPT2 and Transformers **3**

3 Experiments **4**

3.1 Approach to the Study 5

3.2 Setup 5

3.3 Experiments 5

3.4 Chatbot - GRU Model 7

3.5 Smart Speaker - GRU Model 8

3.6 Chatbot - Transformer Model 8

3.7 Smart Speaker - Transformer Model 9

3.8 Chatbot - GPT2 Model 9

3.9 Smart Speaker - GPT2 Model 10

A Terminology **11**

A.1 Gated Recurrent Unit 11

A.2 Sequence to Sequence - Training 11

A.3 Sequence to Sequence - Architecture 12

A.4 Corpus Considerations 13

A.5 Word Embeddings 14

A.6 WordPiece - BPE 14

A.7 Transformer 15

A.8 GPT2 15
A.9 Raspberry Pi 16
A.10 Speech 17

B Tables **18**

B.1 Model Overview 18
B.2 Question Answering – babi 19

Chapter 1

Background/History of the Study

1.1 Background

In their paper Vinyals et al (2015)[1] discuss making a chatbot using a neural network configured for sequence to sequence neural machine translation. We code our own sequence to sequence chatbot, though our results are less than spectacular. As our hand coded model does not run sufficiently well, we use code authored by Matthew Inkawhich (2018)[2].

In their paper Vaswani et al (2017)[3] discuss using the Transformer architecture for solving machine learning tasks. We train a transformer model as a chatbot.

Also Radford et al (2019)[4] discuss the GPT2 neural network for NLP tasks. The GPT2 model is based largely on the Transformer architecture. GPT2 stands for 'Generative Pre-training Transformer 2'.

We implement a chatbot with a GPT2 model. We use a program library from Wolf et al (2019)[5] to run our model.

It is worth noting that with the appearance of the Transformer architecture and WordPiece vocabulary scheme, some technologies have become redundant or obsolete. This may be true of any model that uses RNN components and also the traditional word vector embeddings.

Chapter 2

GPT2 and Transformers

Chapter 3

Experiments

3.1 Approach to the Study

Several tasks are necessary for the project. One task is to implement the algorithms for different models, one the sequence to sequence model and the other the transformer model for a generative chatbot and finally the GPT2 model.

We will not try to rewrite the transformer or GPT2 model ourselves.

3.2 Setup

We use linux computers, sometimes with gpu hardware for parallel processing. We also use the Python programming language. Code from this project can be run with the 3.x version of Python.

When the project was started we did some programming with Keras using Tensorflow as a back-end. Keras was later discarded in favor of Pytorch. Pytorch as a library is still under development at the time of this writing.

Some of the GPT2 code uses Pytorch. Some of the Transformer and GPT2 code uses Tensorflow. There is a repository on Github that has the GPT2 trained model using Pytorch instead of Tensorflow.

We use github as a code repository. Code corresponding with this paper can be found at: https://github.com/radiodee1/awesome-chatbot .

As a coding experiment we rewrite the code for the sequence-to-sequence model. We have varying amounts of success with these experiments. We do not rewrite the GPT2 code from the Tensorflow or Pytorch repository.

3.3 Experiments

We have several basic neural network models. One is the basic sequence to sequence model typically used for neural machine translation. We also have the transformer and the GPT2.

The GPT2 code is versatile. We use it for the chatbot problem. The GPT2 model is pretrained. We experiment with transfer learning and further training of the GPT2 model, but in our case it does not improve the model's performance.

When we talk about the chatbot problem on GPT2 we are talking about a PyTorch version of the GPT2 code.

We found that the chatbot with the hand coded Neural Machine Translation did not work very well. We found that the GPT2 chatbot worked well enough with the 'zero-shot' setup so that fine-tuning was not necessary.

Fine tuning on the GPT2 problem actually had a negative effect. To fine-tune GPT2 also involved training the model on Tensorflow and then translating the model to PyTorch when that was done.

We use speech recognition and speech to text libraries to allow a user to give the chatbot model auditory input.

We also train the transformer to do the chatbot task. This works better than the sequence to sequence but not as well as the GPT2.

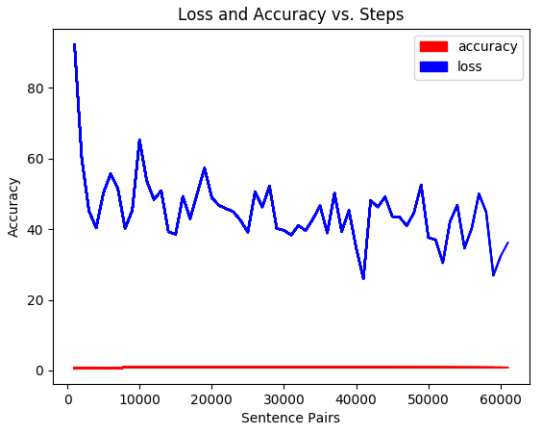
Finally we describe installing the chatbot in small computing platforms, the Raspberry Pi and the O-Droid, in an attempt to create a smart speaker.

All of the three mentioned models work on a laptop computer.

It should be noted that at some point we would like to describe some of our subjective results with the different models. It is difficult to measure our results objectively because we are using code most closely associated with a translation task.

If we were measuring progress with an actual translation task accuracy could be measured as a score that improves when the output matches the meaning of the input, albeit in a different language. We don't use an actual translation task, we use something close to one. The problem is that though the input and the output are in the same language, the input and output have different meanings. We do not dictate what the correct output would be for a given input. The output just has to make sense as a reply in the English language. We would actually prefer that the output not have the same meaning as the input. This makes it difficult to calculate an objective accuracy score. This is touched on in the paper from Vinyals et al(2015)[1].

Loss can still be monitored during training. When the loss stops decreasing you know you should stop training as the model is probably overfitting.



Loss and Accuracy: Red is accuracy and blue is loss.

3.4 Chatbot - GRU Model

We trained the sequence to sequence model on a large english corpus in an attempt to produce a chatbot.

For the sequence to sequence model we want to use text found in a movie dialog corpus (Danescu-Niculescu-Mizil et al, 2011)[6].

In our experience with coding our own chatbot we find that the model learns a single english sentence that can be used for most replies.

Subjectively the GRU chatbot is unusable.

For example our chatbot frequently replies to input with the phrase "I don't know". This is the most common output from our model. In this respect our original chatbot responds poorly.

A large portion of the time the bot's response to a question is "I don't know". This is interesting in that the chatbot has identified that this phrase is a suitable answer to many questions, but it is disappointing that there isn't more variety to the output.

Occasionally the sequence-to-sequence model will reply with the phrase "I'm sorry." This happens very infrequently and it is not clear why the model chooses to reply this way.

3.5 Smart Speaker - GRU Model

The GRU model was installed on a Raspberry Pi. This allowed us to test out speech-to-text and text-to-speech libraries. It also allowed us to compile the Pytorch library for Raspberry Pi.

3.6 Chatbot - Transformer Model

Using the Persona corpus we trained a transformer model to use as a chatbot. This transformer was not pre-trained with any large corpus, so this example did not use transfer learning.

Subjectively this model did not perform as well as the GPT2 model, but it did perform better than the GRU model.

The memory footprint of the model while it was running was below 1 Gigabyte. It is conceivable that the model could be installed on a Raspberry Pi board but it requires a python package called 'tensorflow-model-server' and this package would have to be built from source for the Raspberry Pi.

Training of the model followed a certain pattern. First the model was trained on the persona corpus until a familiar pattern emerged. When the model began to answer all questions with the phrase "I don't know" training was stopped.

At that time the corpus was modified to include no sentences that have the word "don't" in them. Training was started again until the output contained nothing but the phrase "I'm sorry."

At that time the corpus was modified to include no sentences that have the word "sorry" in them. Training was started again and was continued for some period. Training was stopped. A further segment of training was not attempted.

At this point, after looking at the change in loss, further training was not thought of as helpful. Loss stopped improving at some point in this process, and this lack of improvement was taken as a sign that progress was not likely.

Subjectively the transformer model is better than the GRU model. It can respond to something like four sentences. When it comes upon a question that it doesn't expect it defaults to a certain sentence. It can answer questions that you might ask in a rudimentary conversation. It has answers to prompts like 'hi', 'How are you?' and 'What do you do?'. If you tell it your name it will tell you that its name is 'Sarah'. It doesn't answer arbitrary questions. It cannot answer 'What is your favorite color?'. It cannot tell you the time. The default reply sentence for unknown prompts is 'Hi, how are you today?'

3.7 Smart Speaker - Transformer Model

Later we look to install the transformer model on the Raspberry Pi. This model is more dynamic than the sequence-to-sequence model.

The transformer model takes about two minutes to boot on the Raspberry Pi. After that the time between responses is slow. The time between the first two or three responses is uncomfortably slow. After those first responses the time between answers gets to be more natural.

3.8 Chatbot - GPT2 Model

We used a pre-trained GPT2 model with the large english corpus to produce a chatbot and ascertain if this model works better than the sequence-to-sequence model. In our tests this worked well. The corpus is called 'WebText'.

For our experiments GPT2 was used for the chatbot model in 'zero-shot' mode. This means we did no special fine-tuning of the GPT2 model in the application.

We did do some special coding for the input and output of the GPT2 code in order to operate it as a chatbot. Input and output was limited to about 25 tokens.

Input to the model was prepended with the character string "Q:" by our code. Output was observed to have the character string "A:" prepended to it. We assume therefore that the GPT2 model was at some point exposed to the "Question/Answer" paradigm in written passages during its training. This was helpful.

Output from the GPT2 model was usually larger in size than we needed. Also, output had the character of having some sensible utterance followed by some output that was only a partial sentence.

It was necessary to 'scrape' the output. First the output was checked for the "A:" character string at the start. If it was there it was removed. Then the first complete sentence was used as output, while words and phrases after that were discarded.

Lastly we decided that we would attempt to give the model some details that it could draw on during normal execution. We had two choices here. One choice was to train the model using fine-tuning and transfer learning to recognize certain questions and to supply answers. The other choice was to simply show the model the list of facts that we thought were important before every input sequence. This information would be summarized with each reply.

The second choice was more interesting. The text that the model was shown always included the name of the model (picked somewhat arbitrarily) along with information about the location of the model and the occupation. The time was also included.

Subjectively the model was the best of the three tested. The model would answer questions about it's location, it's name, and the time, faithfully most of the time. Interestingly there where times when it did not do so. Some times it used alternative answers. For example, it would answer with the time but not the correct time. This was odd.

Under almost all circumstances the output was sensible English. There were few if any times where the model replied with gibberish.

The subject matter of the prompts did not need to be the same as the simple introductory conversation of the transformer model. In fact any subject matter could be chosen and the model would answer. The model did not remember its own answers but it was consistent. Questions it answered include 'What is your favorite color?' and 'Do you like lollipops?'.

3.9 Smart Speaker - GPT2 Model

Tests showed that the GPT2 chatbot worked well. We wanted to continue and allow the chatbot to have more of the abilities of a smart speaker. We constructed a simple corpus that contained key phrases that we wanted the chatbot to recognize and act upon. We did some transfer learning with this new corpus.

We found that one of two things would happen. The chatbot would either learn the new phrases and forget all it's pre-training, or it would not learn the new phrases and it would retain all it's pre-training. For our examples there seemed to be no middle ground. Comparisons were made with all available models and a version without the transfer learning was settled on.

Code was added that uses Text To Speech and Speech To Text libraries. In this way the model could interact with a subject using auditory cues and commands.

We did some programming that allowed the GPT2 model to launch programs when directed to by the user. In this way we have tried to move our project closer to the smart-speakers that are produced commercially. The programming did not rely on the neural-network aspects of the model. Instead the code used heuristics and simple word recognition. This code can be disabled when the model is run from the command line.

Appendix A

Terminology

A.1 Gated Recurrent Unit

A Gated Recurrent Unit is a relatively simple recurrent network component. It is a component where the output of the model is fed, after some modification, to the input of the model.

At each time step the model sees the new input and a version of previous inputs. In this way it remembers previous inputs. Several inputs in a series are fed to a GRU in sequence and the GRU is responsible for deciding which of the inputs is most important.

GRU elements, like all recurrent network components, suffer from information loss when the input segments are very long.

A.2 Sequence to Sequence - Training

The sequence to sequence model we use is based on the 'Neural Machine Translation' model, or NMT. Originally designed for translating one language to another, NMT can be used to create a chatbot.

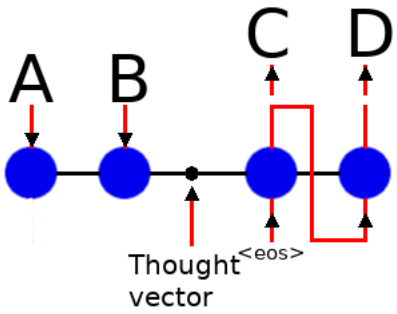
In NMT two languages are given to the sequence to sequence model. For a chatbot NMT is used with two identical language sets. The input language is the same as the output language.

After establishing this language relationship all that is left is to find a suitable training set. Vinyals et al (2015)[1] use a set of movie subtitles. For each sentence in the data set the network is trained to reply with the contents of the next sentence in the set.

It is pointed out in the paper (Vinyals et al, 2015)[1] that the NMT network is typically designed to answer questions directed at the computer with an answer that has the same meaning - though in a different language. Training for the chatbot consists of sentence pairs that are not necessarily related in that way.

During training accuracy improvements are not necessarily reflective of progress, where loss improvements are.

We use a GRU arrangement for a Sequence to Sequence chatbot.



Seq2seq: A and B represent an input sequence and C and D represent the corresponding output.

A.3 Sequence to Sequence - Architecture

In Fig. 1 we generalize a sequence to sequence model. The idea is that A and B, on the left side of the diagram, deal with the encoding of sentences. A and B would be consecutive words in a sentence, and the round blue nodes below A and B are RNN units. They are Recurrent Neural Network elements. C and D are outputs and in the right side of the diagram the blue nodes represent the output RNN units. Between the input and the output there is a corridor of information exactly the size of the RNN hidden vector.

All of the information that the decoder uses for it's output is present in this corridor and is passed along the corridor from the encoder. For this reason we refer to it as the thought vector.

Making this vector larger by increasing the size of the hidden unit size allows for more information in the thought vector. Size also increases the time to train the network. The size must also match the dimension of the vectors in the GloVe or Word2Vec download if one of those is used.

Ultimately exceedingly large hidden dimension does not improve the sequence to sequence model.

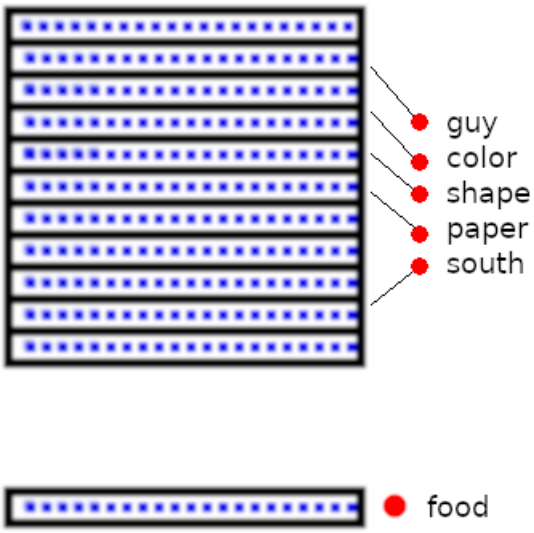
A.4 Corpus Considerations

We have collected several data sets for the training of a chatbot model. Firstly we have a corpus of movie subtitles. Secondly we have the 'JSON' dump from Reddit that is downloadable. Finally we have the corpus described by Mazaré et al(2018)[7]. This final corpus is designed for training the chatbot task specifically. This is referred to as the Persona corpus.

The movie corpus is medium sized and the Reddit 'JSON' download is large and filled with hyperlinks and sentence fragments.

At the time of this writing we are using the movie subtitles corpus and the Persona corpus. We use the movie corpus because it is smaller. Both the movie corpus and the Reddit corpus are described as noise filled, so it is likely that neither one is perfect for the training. The movie corpus is easier to deal with if we are training on a single processor. In the future if we can train in a faster environment the Reddit corpus might be superior.

For the Persona corpus the text is organized into 'JSON' objects. There are several different repeated labels. Some of the text is meant to be used in question and answer pairs. There is also some very specific information there that is not organized in this kind of pattern. When we take apart the Persona corpus we find that the sentences labeled with the 'history' tag are most suited to our task. We record these values only and discard other labels.



Embeddings: Each word from a dictionary is converted to a vector of numbers.

A.5 Word Embeddings

There are several basic building blocks of sequence to sequence models. They are regular neural network cells, recurrent network components, and word embedding components.

Regular neural network cells, are arranged in layers and are pretty straight forward in most programming environments. To use them you define their dimensions in width and height. They have weights and biases that must be initialized before use.

Recurrent networks have internal parts that are constructed of regular network cells, so they have weights and biases too. They have several internal dimensions that need to be set. One of these is the RNN hidden unit. This hidden unit is a dimension.

Word embedding components are the third item we want to describe. What happens is words are translated from strings to individual numbers from a vocabulary dictionary. The dictionary only contains a single unique number for every word. Then the number is passed through an embedding structure. This turns the single number into a vector of numbers that is the same size as the RNN hidden dimension. Then, from that time on the model uses the vector instead of words.

The contents of the embedding unit is a table of numbers, all of the size of the RNN hidden dimension. The vectors are usually, but don't have to be, unique values. There is one complete hidden dimension sized vector for each word in the vocabulary.

The vectors can be initialized randomly or they can be filled with predetermined values. As the network trains the embedding values can either be modified or frozen in place. Typically if the contents were initialized randomly the values would be trained. If the contents were filled with predetermined values you don't want to train them or change them in any way.

There are at this writing two main types of pretrained word embeddings. One is called 'Word2Vec' and one is called 'GloVe'.

Word2Vec is short for 'Word to Vector.' (Mikolov et al, 2013)[8] GloVe is short for 'Global Vector.' (Pennington et al, 2014)[9] .

A.6 WordPiece - BPE

BPE stands for 'Byte Pair Encoding.' WordPiece is a particular implementation of Bite Pair Encoding.

WordPiece is used by the BERT system to encode words much the way that Word2Vec does. Like Word2Vec, WordPiece has a vocabulary list and a table of embeddings that maps one word or token to a vector of a given size.

WordPiece, though, handles Out Of Vocabulary (OOV) words gracefully. It breaks large words into smaller pieces that are in the vocabulary, and has a special notation so that these parts can easily be recombined in order to create the input word again.

A.7 Transformer

In this section we discuss the 'Transformer' model. In their paper, Vaswani et al (2018)[10] describe use of the python project Tensorflow and the Transformer model which is part of it.

The Transformer uses no recurrent elements. It is in a sense a group of attention mechanisms. The Transformer in this case is a single structure that can be used to solve many machine learning problems. It is used for Neural Machine Translation, Sentiment Analysis, and others. It is a single model capable of solving many machine learning questions.

We use the translation models for the chatbot problem by feeding the model the English language on both input and output.

The Transformer itself can be configured for sentence long output but it is not a pre-trained model. There are pre-trained versions of the transformer, one of which is called BERT. BERT is described in Devlin et al (2018)[11] and the acronym stands for Bidirectional Encoder Representations from Transformers. Unfortunately BERT output is as a classifier. Full sentence-length output is not supported.

A.8 GPT2

Another pre-trained model that uses the Transformer is GPT2. GPT2 stands for 'Generative Pre-training Transformer 2.'

GPT2 is a model that takes as input a seed sentence or topic and returns as output text in the same language that is auto-generated. GPT2 also is very capable at summarizing the input or seed statement. We use both of these capabilities in our experiments.

There are two GPT2 models. One is larger than the other. The smaller model has been released

and the larger model has not.

GPT2 is discussed in Radford et al (2019)[4] and on the blog associated with the parent company, OpenAI.com. (https://openai.com/blog/better-language-models/)

GPT2 comes with its own vocabulary encoding and decoding functionality. This system is closer to WordPiece and BPE than it is to Glove or Word2Vec.

It is pre-trained on a corpus that is developed from Reddit posts called WebText. WebText is a 40 Gigabyte corpus that takes high powered computers to train.

The version of GPT2 which has been released is similar in size to the largest currently released BERT transformer model. GPT2 has been released in Tensorflow format, and more recently in a converted PyTorch format. It is possible to download the trained model and then fine-tune the model for your own task. This kind of fine-tuning is called 'transfer learning'.

GPT2 works so well in certain conditions that it is appropriate to use without fine tuning. This type of implementation is called 'zero-shot' implementation. We use GPT2 in a zero-shot implementation with the chatbot problem.

A.9 Raspberry Pi

A Raspberry Pi is a small single board computer with an 'arm' processor. There are several versions on the market, the most recent of which sports built-in wifi and on-board graphics and sound. The memory for a Raspberry Pi 3B computer is 1Gig of RAM. Recently available, the Raspberry Pi 4B computer can sport 4Gig of RAM.

It has always been the intention that at some time some chatbot of those examined will be seen as superior and will be installed and operated on a Raspberry Pi computer. If more than one model is available then possibly several models could be installed on Pi computers.

For this to work several resources need to be made available. Pytorch needs to be compiled for the Pi. Speech Recognition (SR) and Text To Speech (TTS) need to work on the Pi.

For the transformer model to work Tensorflow needs to work on the Pi.

All the files that are trained in the chosen model need to be small enough in terms of their file size to fit on the Pi. Also it must be determined that the memory footprint of the running model is small enough to run on the Pi.

In the github repository files and scripts for the Raspberry Pi are to be found in the 'bot' folder.

Early tests using Google's SR and TTS services show that the Pi can support that type of functionality.

Google's SR service costs money to operate. Details for setting up Google's SR and TTS functions is beyond the scope of this document. Some info about setting this up can be found in the README file of this project's github repository.

The pytorch model that is chosen as best will be trained on the desktop computer and then the saved weights and biases will be transferred to the Raspberry Pi platform. The Pi will not need to do any training, only inference.

A.10 Speech

Google has python packages that translate text to speech and speech to text. In the case of text to speech the library is called 'gTTS'. In the case of speech to text the library is called 'google-cloud-speech'.

The gTTS package is simple to use and can be run locally without connection to the internet. The google-cloud-speech package uses a google cloud server to take input from the microphone and return text. For this reason it requires an internet connection and an account with Google that enables google cloud api use. Google charges the user a small amount for every word that they translate into text.

Appendix B

Tables

B.1 Model Overview

Model Name	File Size	RAM Train	RAM Interactive	Pretrained Weights	Hand Coded
Seq-2-Seq/GRU	230 M	1.1 G	324 M	NO	YES
Transformer	25 M	556 M	360 M	NO	NO
GPT2 small*	523 M	5 G	1.5 G	YES	NO

* a large GPT2 model exists, but it has not been released to the public.

B.2 Question Answering – babi

	DMN plus	hand coded	GPT2 - small
QA1: Single Supporting Fact	100	100	100
QA2: Two Supporting Facts	99.7	xx	96.0
QA3: Three Supporting Facts	98.2	xx	38.18
QA4: Two Argument Relations	100	100	100
QA5: Three Argument Relations	99.5	99.4	97.8
QA6: Yes/No Questions	100	100	98.4
QA7: Counting	97.6	97.8	98.6
QA8: Lists/Sets	100	99.4	98.8
QA9: Simple Negation	100	98.2	97.0
QA10: Indefinite Knowledge	100	99.4	96.6
QA11: Basic Coreference	100	100	97.6
QA12: Conjunction	100	100	99.4
QA13: Compound Coreference	100	99.8	95.8
QA14: Time Reasoning	99.8	97.2	87.0
QA15: Basic Deduction	100	100	64.4
QA16: Basic Induction	54.7	48.2	96.39
QA17: Positional Reasoning	95.8	59.2	99.0
QA18: Size Reasoning	97.9	91.6	100
QA19: Path Finding	100	xx	97.3
QA20: Agents Motivation	100	100	100

Bibliography

[1] O. Vinyals and Q. V. Le, "A neural conversational model," *CoRR*, vol. abs/1506.05869, 2015.

[2] Matthew Inkawhich, "pytorch-chatbot," 2018. https://github.com/pytorch/tutorials/blob/master/beginner_source/chatbot_tutorial.py.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017.

[4] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.

[5] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew, "Huggingface's transformers: State-of-the-art natural language processing," *ArXiv*, vol. abs/1910.03771, 2019.

[6] C. Danescu-Niculescu-Mizil and L. Lee, "Chameleons in imagined conversations: A new approach to understanding coordination of linguistic style in dialogs.," in *Proceedings of the Workshop on Cognitive Modeling and Computational Linguistics, ACL 2011*, 2011.

[7] P. Mazaré, S. Humeau, M. Raison, and A. Bordes, "Training millions of personalized dialogue agents," *CoRR*, vol. abs/1809.01984, 2018.

[8] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[9] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*

(EMNLP), (Doha, Qatar), pp. 1532–1543, Association for Computational Linguistics, Oct. 2014.

[10] A. Vaswani, S. Bengio, E. Brevdo, F. Chollet, A. N. Gomez, S. Gouws, L. Jones, L. Kaiser, N. Kalchbrenner, N. Parmar, R. Sepassi, N. Shazeer, and J. Uszkoreit, "Tensor2tensor for neural machine translation," *CoRR*, vol. abs/1803.07416, 2018.

[11] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018.