A Generative Chatbot with Natural Language Processing

By
David Liebman

In Partial Fulfillment of the Requirement for the Degree of
MASTER OF SCIENCE

in
The Department of Computer Science
State University of New York
New Paltz, New York, 12561

December 2020
A Generative Chatbot with Natural Language Processing

David Liebman
State University Of New York at New Paltz

We the thesis committee for the above candidate for the
Master of Science degree, hereby recommend
acceptance of this thesis.

Professor Chirakkal Easwaran, Thesis Advisor
Computer Science Department, SUNY New Paltz

Professor Min Chen, Thesis Committee Member
Computer Science Department, SUNY New Paltz

Professor Hanh Pham, Thesis Committee Member
Computer Science Department, SUNY New Paltz

Submitted in partial fulfillment of the requirements
for the Master of Science degree in
Computer Science at the
State University of New York at New Paltz
Abstract

The goal in this thesis is to create a chatbot, a computer program that can respond verbally to a human in the course of simple day-to-day conversations.

A deep learning neural network model called the Transformer is used to develop the chatbot. A full description of a Transformer is provided. The use of a few different Transformer-based Natural Language Processing models to develop the chatbot, including Generative Pre-Training 2 (GPT2), are shown. For comparison a Gated Recurrent Unit (GRU) based model is included. Each of these are explained below.

The chatbot code is installed on a small device such as the Raspberry Pi with speech recognition and speech-to-text software. In this way a device that can carry out a verbal conversation with a human might be created. For the GRU-based model a Raspberry Pi 3B with 1GB RAM can be used. A Raspberry Pi 4B with 4GB of RAM is needed to run a chatbot with the GPT2.
Dedication

This is dedicated to my mother and father. Without them this paper could never be completed. Thanks to Professor Chirakkal Easwaran, Professor Hanh Pham, and Professor Min Chen. Thanks also to Leslie Masker and all the Professors in the Computer Science Department. Thanks to other family members and friends. Thanks to Bess Stewart, Joe Matos and others.
## Contents

1 Background/History of the Study ................................................................. 1
  1.1 Background .............................................................................................. 1
    1.1.1 Recurrent Neural Network and Transformer .................................. 1
    1.1.2 Pre Trained Language Models .......................................................... 1
  1.2 Outline ...................................................................................................... 2
  1.3 Goals For This Thesis .............................................................................. 2

2 Recurrent Neural Network ........................................................................... 4
  2.1 Recurrent Neural Network Components ............................................... 4
    2.1.1 Hidden Values .................................................................................. 4
    2.1.2 Simple RNN ................................................................................... 4
    2.1.3 Gated Recurrent Unit ...................................................................... 5
  2.2 Sequence-to-Sequence and Translation ................................................... 6
    2.2.1 Word Embeddings .......................................................................... 7
    2.2.2 Corpus ........................................................................................... 8
    2.2.3 Training and Evaluation ................................................................ 8
    2.2.4 Input Tokens .................................................................................. 9
    2.2.5 Encoder ........................................................................................ 9
    2.2.6 Decoder ......................................................................................... 10
    2.2.7 Output Tokens .............................................................................. 11
    2.2.8 Loss and Accuracy During Training ............................................. 11
    2.2.9 Attention Mechanism .................................................................... 12
    2.2.10 Batching Problems ....................................................................... 13
    2.2.11 Exploding or Vanishing Gradients ............................................... 13
    2.2.12 Sequence-to-Sequence Chatbot .................................................... 13

3 Transformers and the Generative Pre-Training ......................................... 15
  3.1 Transformer and Attention ...................................................................... 15
    3.1.1 Byte Pair Encoding ....................................................................... 15
    3.1.2 Attention ....................................................................................... 15
    3.1.3 Encoder - Scaled Dot-Product Attention ..................................... 16
3.1.4 Decoder Attention I - “Key” and “Value” ........................................... 21
3.1.5 Decoder Attention II - “Query” .......................................................... 23
3.1.6 Decoder Attention II - Masking ............................................................ 23
3.1.7 Input - Positional Encoding ................................................................. 24
3.1.8 Output - Feed Forward Network .......................................................... 25
3.1.9 Visualization - Transformer ................................................................. 25

3.2 The Generative Pre-Training 2 Model ....................................................... 25
3.2.1 Pre-Training ......................................................................................... 26
3.2.2 General ............................................................................................... 26
3.2.3 Training ............................................................................................... 27
3.2.4 Inference .............................................................................................. 27
3.2.5 Corpus ................................................................................................. 29
3.2.6 Releases ............................................................................................... 29
3.2.7 Application Details ................................................................................ 29
3.2.8 Visualization - GPT2 ........................................................................... 30

4 Experimental Design and Setup .................................................................. 31
4.1 Approach to the Study ............................................................................ 31
4.2 Hardware Installation Overview ............................................................. 32
4.3 Setup ....................................................................................................... 33
4.3.1 Graphical Processing Unit vs. Central Processing Unit ...................... 33
4.3.2 Raspberry Pi ......................................................................................... 34
4.3.3 Jetson Nano ......................................................................................... 34
4.3.4 Reply Time .......................................................................................... 35
4.3.5 Tensorflow vs. Pytorch ....................................................................... 36
4.3.6 Speech and Speech To Text ................................................................. 36
4.3.7 Corpus Considerations ........................................................................ 37
4.3.8 Chatbot vs. Smart Speaker ................................................................. 37
4.4 ARMv7 Build/Compile ............................................................................ 38
4.4.1 Pytorch “torch” Library 1.1.0 For ARMv7 ........................................... 38
4.4.2 Pytorch “torch” Library 1.4.0 For ARMv7 ........................................... 38
4.4.3 Docker Container “tensorflow-model-server” For ARMv7 ................. 39
5 Experimental Results - Raspberry Pi

5.1 Experiments - Installations

5.1.1 Questions

5.1.2 Checklist

5.2 Chatbot - Gated Recurrent Unit Model

5.2.1 Questions

5.2.2 Checklist

5.3 Smart Speaker - Gated Recurrent Unit Model

5.4 Chatbot - Transformer Model with Persona Corpus

5.4.1 Questions

5.4.2 Checklist

5.5 Smart Speaker - Transformer Model with Persona Corpus

5.6 Chatbot - Transformer Model with Movie Corpus

5.6.1 Questions

5.6.2 Checklist

5.7 Smart Speaker - Transformer Model with Movie Corpus

5.8 Chatbot - Generative Pre-Training 2 Model

5.8.1 Context Experiment

5.8.2 GPT2 Fact Sheet

5.8.3 History Experiment

5.8.4 Program Launching

5.8.5 Summary

5.8.6 Questions

5.8.7 Checklist

5.9 Smart Speaker - Generative Pre-Training 2 Model

6 Further Installations

6.1 Generative Pre-Training 2 - X Large Model

6.1.1 Questions

6.1.2 Checklist

6.1.3 Context Experiment

6.1.4 History Experiment

6.1.5 Artificial Intelligence Markup Language Experiment
6.1.6 User Name Experiment ................................. 56
6.1.7 Internet Search Experiment ............................ 56
6.2 Generative Pre-Training 2 - Jetson Nano .................. 58

7 Results and Observations ................................. 59
7.1 GRU vs. Transformer .................................. 59
7.2 Turing Test ............................................ 59
7.3 Word and Sentence Comparisons ......................... 60
   7.3.1 Word Usage ....................................... 60
   7.3.2 Sentence Usage .................................. 62
   7.3.3 Maximum Sentence Values ......................... 63
7.4 Training and Lists .................................... 64

8 Further Research and Readings ......................... 65
8.1 Questions and Further Research ....................... 65
8.2 Winograd Schema .................................... 65
8.3 Generative Pre-Training 3 ............................. 66
# List of Figures

2.1 Hidden Input ................................................................. 4  
2.2 Recurrent Neural Network ............................................. 5  
2.3 Word Embeddings ......................................................... 7  
2.4 Sequence-to-Sequence Architecture ................................ 10  
2.5 Loss and Accuracy ......................................................... 14  
3.1 Transformer Encoder and Decoder .................................... 16  
3.2 Lowering Dimensionality ............................................... 17  
3.3 Attention Output ......................................................... 18  
3.4 Matching Input and Output ........................................... 19  
3.5 Transformer Encoder and Decoder Flow ............................ 20  
3.6 Encoder Attention Detail ............................................... 21  
3.7 Decoder Flow ............................................................... 22  
3.8 Decoder Attention Detail ............................................... 23  
3.9 Decoder Mask ............................................................... 24  
3.10 Visualized Attention Transformer .................................. 25  
3.11 Generative Pre-Training 2 ............................................. 26  
3.12 Generative Pre-Training 2 Inference ............................... 28  
3.13 Visualized Attention GPT2 ............................................ 30  
4.1 Raspberry Pi ............................................................... 34  
4.2 Jetson Nano ............................................................... 35  
5.1 Loss - Larger Transformer Model ..................................... 46  
7.1 Word Usage ............................................................... 61  
7.2 Total Word Responses .................................................. 62  
7.3 Simple Sentence Usage ................................................ 63
List of Tables

GPT2 Size Overview ................................................................. 29
Hardware Overview .............................................................. 32
Time Comparison ................................................................. 36
GPT2 Fact Overview .............................................................. 49
Chapter 1

Background/History of the Study

1.1 Background

Transformer style models and also a Recurrent Neural Network model are used to allow for meaningful comparison, along with the Generative Pre-Training 2 Transformer. RNN models are explained in Chapter 2 and Transformers in are explained in Chapter 3.

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

1.1.1 Recurrent Neural Network and Transformer

In their paper Vinyals et al [1] discuss making a chatbot using a neural network configured for Sequence-to-Sequence Neural Machine Translation. An attempt to code our own Sequence-to-Sequence model was not very fruitful so instead the thesis uses code authored by Inkawhich et al [2].

In their paper Vaswani et al [3] discuss using the Transformer architecture for solving machine learning tasks. A transformer model is trained as a chatbot.

In both of these experiments a common factor is the Movie Dialog Corpus that the models train on. The corpus comes from Danescu-Niculescu-Mizil et al [4].

1.1.2 Pre Trained Language Models

Radford et al [5] discuss the “Generative Pre-Training 2” (GPT2) neural network for Natural Language Processing (NLP) tasks. The GPT2 model is based largely on the Transformer architecture.

This is essentially a Language Model. The model is given a large section of text during training and it is asked to generate a single word or token to add to the end of the sample. During inference it does this over and over to complete a passage. This kind of model shows up in GPT2 and the thesis uses that trained model and some traditional programming techniques to generate text that naturally answers questions. Among all the models tested, this is the model that was found to be most comprehensive in execution.

Several chatbots are implemented, one with a GPT2 model using a program library from Wolf et al [6] to run the model.
1.2 Outline

This thesis starts with explanations of how some Recurrent Neural Network components work, followed by a discussion of Transformers.

The thesis goes on to describe installation of chatbot models on Raspberry Pi 4B computers, and on a Jetson Nano computer.

After that the thesis describes some further installations on an X86_64 computer. Some graphs are given that show word and sentence usage of the Gated Recurrent Unit and Transformer based models. Some final thoughts are offered, and further reading is suggested for those interested.

1.3 Goals For This Thesis

This work focuses on providing a number of implementations of a Generative Chatbot, installed on small headless computers. Some overall goals are listed below. Checks in the check-boxes indicates the goal was achieved.

☐ Implement a generative chatbot using Recurrent Network components. This implementation uses GRU objects and the Tutorial found at Inkawhich et al [2].

☑ Implement a generative chatbot using a Transformer architecture and the Movie Dialog Corpus. This is discussed in Section 5.6.

☐ Implement a chatbot using the Generative Pre-Trained 2 transformer. This is a pre-trained model. This is discussed in Section 5.8.

☐ Subjectively compare the GRU, Transformer, and GPT2 models and the kind of output that each one produces. This is discussed throughout Section 5.

☐ Successfully install Google Speech Recognition and Text To Speech on the Raspberry Pi computer platform, as well as other small computers like the NVIDIA Jetson Nano. This is discussed in Section 4.3.6.

☐ Install a GRU, Transformer, and GPT2 model on individual Raspberry Pi computers. Also install Google Speech Recognition and Speech To Text on these computers so that they can operate without keyboard or monitor. Allow the chatbot model to interact with a human’s voice. This is discussed throughout Section 5.
Install a GPT2 model on a Jetson Nano, along with Google Speech libraries and allow a human to interact with the chatbot. Compare execution time to the same model on the Raspberry Pi. This is discussed in Section 6.2.

Compare the GRU and the Transformer model. Try to compare the word usage and sentence usage of the GRU and Transformer. This is discussed in Section 7.1.

Implement a generative chatbot using Recurrent Network components and an independent code base. This was not accomplished as the task was beyond the scope of this work.

Implement a generative chatbot using a Transformer architecture and the Persona chatbot corpus. This implementation did not work well. This is discussed in Section 5.4.
Chapter 2

Recurrent Neural Network

2.1 Recurrent Neural Network Components

The goal behind Recurrent Neural Network (RNN) components is to detect patterns. Since understanding RNNs is important to GPT2 and Transformer models, it is discussed here.

2.1.1 Hidden Values

The term “Hidden Values” refers to the input or output of the RNN that is not exposed to the input tokens and is not converted to an output token. Hidden values are passed from one RNN component to another but are not passed to the output of the neural network. Additionally they are generated by an RNN element itself. Conceptually the input token from the input sentence is not passed to the Hidden input of any given RNN. They generally have the same dimensions as the input.

![Figure 2.1: Hidden Input - Input and output connections for an RNN component.](image)

2.1.2 Simple RNN

Imagine using a string of words as input. This could be a sentence used for the input of a translation task with each word in the input called a token.

The simplest RNN unit has two inputs and two outputs. They can be arranged in patterns. In this example the input will be a sequence of data as stated above, and the Recurrent Neural Network will be a line of components of the same length as the data.
One input for each component is the hidden state output from the RNN to the left. Another input is the current token from the sequence that the component is encoding. One output is the generated hidden state, meant for the component to the right. The last output is the value that the Recurrent Neural Network outputs or surmises.

![Diagram of RNN component]

Figure 2.2: RNN - 1 and 2 are inputs, 3 is the output.

In Figure 2.2 the two inputs are labeled 1 and 2, and the single output does double duty as both the hidden state output and the value that the RNN outputs or surmises. This is labeled 3.

There are several designs for a RNN component. The inner workings of these components are what makes them different. In the example in the diagram the inner workings are very simple. Two paths, labeled as inputs, take data into the RNN. Their data is combined in the green circle. This combination is done with concatenation and simple feed forward neural network components. The output from the concatenation is passed through the other circle. This is a “tanh” activation operation that limits the output to values from -1 through 1. This “tanh” activation keeps the output within reasonable values. Finally the output is delivered outside the component to the program employing the RNN. In this diagram there is only one output. The single output would serve as both the hidden state output for that position in the encoder or decoder, and also the data output component for that position.

### 2.1.3 Gated Recurrent Unit

An implementation of a Recurrent Neural Network is the “Gated Recurrent Unit” (GRU). There are other varieties of RNN, most notably the Long Short Term Memory (LSTM) cell, not explored here.

A GRU has two inputs and two outputs. The formulas for a Gated Recurrent Unit, as outlined by Denny
Britz in the website WILDML (Britz et al) [7], are as follows.

\[ z = \sigma(x_t U^z + s_{t-1} W^z) \]

\[ r = \sigma(x_t U^r + s_{t-1} W^r) \]

\[ h = \text{tanh}(x_t U^h + (s_{t-1} \circ r) W^h) \]

\[ s_t = (1 - z) \circ h + z \circ s_{t-1} \]

The GRU has two inputs and two outputs. It also has two internal gates. One internal gate is the “reset” gate. This one determines how much of the previous input is combined with the new value calculated by the mechanism of the GRU. It is denoted as “r” above. Another internal gate is the “update” gate. The update gate decides how much new information is to be included in gate computation. It is denoted as “z”.

Here “s_t” is the symbol for the combined output. The two inputs are “x_t” and “s_{t-1}.” “x_t” is hidden state input. “s_{t-1}” is the regular input for the Recurrent Neural Network or Gated Recurrent Unit. Sigmoid activation is used on the two gates, using the symbol “\( \sigma(\ldots) \)” , while tanh activation is used to compute the hidden output.

The dimension of the \( U^z, U^r \) and \( U^h \) matrices is the hidden unit size by the hidden unit size. \( x_t \) is a vector the size of the hidden unit. The \( U \) values, along with \( W^z, W^r \) and \( W^h \) are all simple matrices that allow the GRU to operate.

In the last line, the regular output is determined using the “dot” product which is denoted with a circle, along with an addition operation. In the two gate formulas (the first and second) the output is determined as the sum of two matrix multiplication operations passed through sigmoid activation. This produces values in the range of 0 to 1.

Under most programming circumstances the GRU is not implemented by the average programmer. The programmer employs a language like Python and a programming library like Pytorch or Tensorflow. The library then implements the GRU and makes it easy for the programmer to use that implementation.

### 2.2 Sequence-to-Sequence and Translation

Translating text from one language to another has become a common task for computers. The Sequence-to-Sequence architecture is often used today for this purpose.

A naive approach to translation involves using a dictionary. Each key is encoded as a word from one language and the value for that key would be the translated word in the target language. Of course this
doesn’t work, because different languages not only have different words for the same thing, but they also have different sentence structures for what might be similar concepts.

A better approach for a computer is sequence-to-sequence translation. A description follows with a section at the end for how training works.

In this approach recurrent neural networks are used to obtain translation. Two recurrent neural network components are employed. One is responsible for the input language and the other for the output.

### 2.2.1 Word Embeddings

Also employed are two vocabulary sets. One vocabulary is for the source language and another is for the target language. A table of word vectors the size of the input vocabulary is created and a maximum sentence length is picked. There is also a “hidden size”, which is an important dimension in the model. In practice the hidden size could be 300 units and more for this kind of application.

Words are translated from strings to individual numbers from an input 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. For each word in the dictionary there is a vector whose size is equal to the hidden dimension.

![Figure 2.3: Embeddings - Each word from a dictionary is converted to a vector of numbers.](image)

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, they are not trained or changed in any way.

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


2.2.2 Corpus

A text must be prepared for training. A text corpus with source and target pairs is chosen. Sentences in the source corpus are paired with sentences with the same meaning in the target corpus. A sentence length restriction is observed and, for all sentences shorter than that length, a special “end-of-sequence” token is appended to all sequences. This restriction is applied in both languages.

2.2.3 Training and Evaluation

Our task is to train a Neural Network to translate a given input corpus. It is generally felt that the corpus should be divided into three parts. The largest part, possibly 80%, is held for training. Then 10% should go to what is called “validation” and 10% should go to “testing”.

The training portion, obviously, is used during the training phase. The model can become familiar with training data and provide target data that leads one to believe that the model is learning well. This is similar to memorization. The model may see the same training data repeatedly and thereby learn how to answer a source with a target without learning the task of why it is giving a particular answer.

For this reason, during training, the validation set is used for statistical purposes. The actual training mechanism is halted and the validation set is run through the model. Loss is measured and compared to the loss found when training is enabled. Loss during validation is usually greater than during training. This is done periodically during what would otherwise be called training.

This is useful for tuning hyper-parameters, which are parameters that occur in the model. Repeatedly one trains, and as one does, one assesses the validation scores for clues on what to change in the hyper-parameters.

When done one uses the final holdout set, the “test” set, to determine weather or not training has been successful. One must never train with the test or validation set.
2.2.4 Input Tokens

At this point, the model takes a word, translates it to an integer, and finds the vector in the word embedding table that represents that word. It does this for the first word and all subsequent words one by one. Then it gives the entire vector for a word to the GRU, one at a time. The GRU takes the word and passes it to inner components. It decides whether to return as output only the input or the input modified. This is what the Gated Recurrent Unit does internally.

The Gated Recurrent Unit takes as input two vectors. It processes the “input” vector and returns another vector. This could be exactly the same as the input but is usually somehow changed. The input vector and the output vector have the dimension of the “hidden size” mentioned above. Throughout the discussion of this model the hidden size will remain the same. The GRU also operates on two hidden states. One hidden state, a vector, is taken in and another hidden state, also a vector, is produced for output. The input components, using the source language, are the encoder and the output components, using the target language, are the decoder.

2.2.5 Encoder

The input segments, composed of Gated Recurrent Units, take two input vectors and return two output vectors. One input is the vector from the embedding table. Another input vector is the previous hidden state. The hidden state is the same size as the input from the embedding table, but typically it comes from the previous Gated Recurrent Unit. The output vector is a hidden value for the Recurrent Unit to the right.

The very first Gated Recurrent Unit in the input encoder ignores the fact that the first word has no hidden value. It consumes the first word vector. Then it passes its output to the next Gated Recurrent Unit in the encoder. This GRU uses the output of the previous GRU as the hidden value. It also uses the vector for the second word. It passes its important information to the Gated Recurrent Unit to its right. This is repeated if needed. Then the last Gated Recurrent Unit in the encoder passes its hidden state to the output decoder.

A complicating detail is that although many GRUs are called for in the encoder, they all use the same set of weights and biases. For this reason only a single input Gated Recurrent Unit is used for all of the words in practice.
Figure 2.4: Seq2seq: The left represents an input sequence and the right represents the corresponding output. “sol” stands for “start-of-line”. “eol” stands for “end-of-line”.

Figure 2.4 generalizes a sequence-to-sequence model. The left side of the diagram deals with the encoding of sentences. “Open the door” would be consecutive words in a sentence, and the rectangular blue nodes above those words are Recurrent Neural Network units. “Ouvre la porte” are outputs and in the right side of the diagram, the 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.

2.2.6 Decoder

The decoder is in charge of generating tokens that represent, in this case, the translation of the input to the output language. The output uses Gated Recurrent Unit segments also. The first hidden input for the first output cell is taken from the last hidden output of the last recursive unit of the input. It is important because it is the point where a great amount of data is passed from the encoder to the decoder. The connection at this point is said to carry the “thought vector.” Most of the information responsible for translating one language to another is passed at this point.

The hidden values from the input section are passed to the first output Gated Recurrent Unit. It outputs the values that are later converted to the first word of the translation. The first output Gated Recurrent Unit also has a hidden state. It passes the first word and the hidden state on to the second Gated Recurrent Unit to its right.

The second Gated Recurrent Unit generates the second word and also its own hidden state. The second word is recorded and the word and the hidden state are passed on to the next Gated Recurrent Unit. This is repeated until a special “end-of-sequence” token is generated or until the number of tokens equals the maximum number allowed.

All of the information that the decoder uses for its output is present in the thought vector and is passed along the corridor from the encoder. Aside from Attention Mechanisms, there are no other places where
information is passed from the encoder to the decoder.

Making this vector larger by increasing the size of the hidden dimension 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 an exceedingly large hidden dimension does not improve the sequence-to-sequence model.

Again, in the Decoder, many GRU units are called for but they all share the same weights and biases. For this reason, a single GRU is employed for the entire decoder. The encoder and decoder are never the same unit.

2.2.7 Output Tokens

Each output is currently in the form of a vector. These vectors are long strings of floating point numbers, each one the dimensions of the “hidden size” mentioned above. They are converted to the dimensions of the output vocabulary, through matrix multiplication. Then they are processed in what is called a “softmax” function. This processing, along with an “arg-max” function, determines the index of the maximum value in the new vocabulary-sized vector. This index allows the program to find the corresponding word in the output vocabulary. This word is then used as the model output at that point in the output decoder.

Some computer models do language translation this way. Using ‘arg-max’ is an example of a greedy approach to decoding. Another approach is ‘Beam Selection,’ not discussed here.

2.2.8 Loss and Accuracy During Training

At first the prediction from a model is not close to the desired output. The output is compared to the prediction and a “loss” is generated. “Loss” measures the difference between the predicted output and the target. A larger loss represents two values, prediction and target, that are further apart.

Another metric is Accuracy, a numerical representation of the difference between the desired output and the generated prediction. It is a percentage of the time that the output is exactly correct.

Getting a prediction, running input data through a neural network, is forward-propagation. Training is a mathematical process involving back-propagation. Back-propagation identifies areas of the model weights that need to be modified in order to get the proper prediction in the future.

The derivative of the loss function is taken in order to back-propagate. The derivative is manipulated with the learning rate. A learning rate is required in these calculations. It is a small positive fraction.

The original weight value is changed minutely. The amount changed with every backward propagation is dependent on the learning rate. The result is a set of adjusted weight matrices and a new loss. When these
matrices are used later they allow for better predictions.

To train the model this is done repeatedly with every source/target sentence pair. The model is changed and predictions start to match the target. The loss should decrease over time and the accuracy should increase.

2.2.9 Attention Mechanism

Attention Mechanisms are used by sequence-to-sequence models to transfer more information from the encoder to the decoder. The encoder imparts information to the decoder only at the “thought vector.” Attention helps the encoder tell the decoder which word is more important. This stressing of a vector by the model is the attention mechanism attending to one output.

A simple attention mechanism is used in the Sequence-to-Sequence model by Inkawhich et al. [2] The concept for this attention comes from Luong et al. [10]

Luong et al [10] are interested in three kinds of calculation for their attention mechanisms. These methods use slowly increasing levels of complication. First they propose a method that uses the dot-product. Then they propose a method that uses a field of weights. Finally they use a method that uses concatenation, along with a field of weights and a pass through a “tanh” activation layer.

\[
\text{score}(h_t \bar{h}_s) = \begin{cases} 
    h_t^\top \bar{h}_s & \text{dot} \\
    h_t^\top W_a \bar{h}_s & \text{general} \\
    v_a^\top \text{tanh}(W_a [h_t; \bar{h}_s]) & \text{concat}
\end{cases}
\]

Here \(h_t\) is the symbol for the output of the current decoder and \(\bar{h}_s\) is the symbol for another output taken from the input encoder. This is the entire set of encoder states. \(h_t^\top\) stands for \(h_t\) transposed. Inkawhich et al [2] uses the “dot” variety.

The formula is used after the decoder Gated Recurrent Unit calculates it’s hidden state. It is below.

\[
\text{score} = h_t^\top \bar{h}_s
\]

The output of the current decoder is transposed. It is multiplied by the hidden value from the entire set of encoder states. Later it is multiplied by the Gated Recurrent Unit decoder output, and passed through a “tanh” activation layer. It becomes the decoder output.
2.2.10 Batching Problems

There are some general batching problems with using Recurrent Neural Network components. If a batch of data is to be cycled through a recurrent component, it should go through all the components at once.

In Neural Machine Translation the data has to be processed one batch at a time. The data is passed to the first RNN. Only then it can be passed to the next RNN. A single GRU is used in this example for all words in the encoder, while another GRU is used in the decoder. These GRUs can not process the subsequent batches all at once. It must be done sequentially creating a bottleneck. These sorts of bottlenecks are found in Neural Machine Translation based on recurrent elements.

2.2.11 Exploding or Vanishing Gradients

There are some specific problems that affect gradients in Recurrent Neural Network components. Data that begins with a floating point value above 1.0, when multiplied with weights with values above 1.0, will return values above 1.0. The same RNN component is used for every position in the encoder or decoder. Some values will tend to infinity. This is called an exploding gradient.

The reverse is true. When a number starts out as a floating point below 1.0, and is multiplied with RNN weights below 1.0, it will tend to a value of zero. This is called a vanishing gradient.

Gradients in an RNN can be unstable. RNN input and output lengths need to be limited to a small number. In our examples a limit of 10 is used.

2.2.12 Sequence-to-Sequence Chatbot

Vinyals et al [1] make an interesting proposition. It is possible to make a Neural Conversational Model by constructing a Sequence-to-Sequence model. Instead of using two corpus from different languages, a single language is used for both source and target.

Chatbots have for a long time been constructed using AIML. AIML, (Artificial Intelligence Markup Language) requires hand coding individual rules for every input and response. A Neural Conversational Model would not use those rules.

To create such a model, more is required than a single input and output language. There must be a relationship between the source and the target permitting a question and answer paradigm. Finding a corpus like this can be difficult.

It would be easy to duplicate the input source material in the target corpus, producing auto-encoding. The model would learn to repeat everything that was given to it in its output. Though the model learns a task, it is not the preferred task. Conversations, on the other hand, supply that relationship.
Vinyals et al [1] use a movie transcript data set. Essentially they take movie dialogue and break it into sentences. Consecutive sentences can be divided into source and target. Each sentence is a source and the sentence that follows it would be the target for that pair.

Training for this model is relatively straightforward, however the accuracy does not increase. Although the loss decreases, the accuracy is unimportant.

![Loss and Accuracy](image)

Figure 2.5: Loss and Accuracy: Loss is above and accuracy is below.

This is because the source and target do not have the same meaning. The model does learn the task at hand but during training accuracy must be ignored. Success is usually measured by the accuracy of the holdout test set. Here success is measured with the subjective examination of the trained model.

If the answers are considered satisfactory the training was a success.
Chapter 3

Transformers and the Generative Pre-Training 2

3.1 Transformer and Attention

The Transformer is a mechanism that is based entirely on attention. Strictly speaking, this is not the attention explored in the sequence-to-sequence model in Section 2.2.9, though there are some similarities. It is a model that uses no recurrent components.

Recurrent components have some negative qualities. They are hard to run with batch input data. In addition they do not work with very long data strings.

Transformers use no Recurrent Neural Network components. Their operations can be parallelized so that large batches of data can be processed at once during the same time step. Longer sequences can be considered as well, so Transformer input can contain longer English language sentences and even paragraphs.

Transformers are usually constructed on eight or more of these layers in the decoder and the encoder. One layer is discussed below.

3.1.1 Byte Pair Encoding

BPE stands for “Byte Pair Encoding.” WordPiece is a particular implementation of BPE. WordPiece is used by some Transformer systems 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. Byte Pair Encoding doesn’t use pre-trained word embeddings as do Word2Vec and GloVe.

For the Generative Pre-Training 2 transformer, a version of BPE is used instead of a vocabulary system like Word2Vec or GloVe. Some form of BPE is included in almost every Transformer type Neural Network model, so no decision needs to be made about what type of word embeddings to use.

3.1.2 Attention

Attention mechanisms are used in a similar way in three places in the Transformer model. The first implementation of Self Attention is discussed below. Each of these attention mechanisms is contained in a
layer. There are typically the same number of layers in the encoder as in the decoder.

Input to the Transformer is composed of strings of words from a desired input language. Output is composed of words in a given language. Input words are treated very much the way that they are in sequence-to-sequence models. A word is translated to a number and that number indexes an entry in a word-to-vector table. From that time forward, a word is represented by a vector of floating point numbers. In a Transformer this word vector can be large. In the original paper, Vaswani et al [3] use a vector size of 512. The discussion of Generative Pre-Training 2 will see vector sizes of 768 and 1280.

Figure 3.1: Transformer Encoder and Decoder. - $N_x$ shows number of layers. (Data travels from the bottom to the top.) - Vaswani et al [3]

### 3.1.3 Encoder - Scaled Dot-Product Attention

Each layer of the Transformer’s encoder has a signature self-attention mechanism. This is possibly one third of the entire Transformer mechanism, but a variety shows up in the other two-thirds.

Initially the input word vectors are converted to three other values. These vectors are similar to the input vector but they have a smaller dimensionality. Converting the word vectors in this way is accomplished by...
three simple matrix multiplication operations.

Figure 3.2 shows a simple conversion of this type. In the diagram a conversion of a vector with dimension of 1x3 to a dimension of 1x2 is shown. In a real world example matrices are converted from a vector of 1x512 to 1x64, a division of 8.

![Diagram of matrix multiplication](image)

Figure 3.2: Lowering Dimensionality

In the Transformer, this conversion operation is probably the reason for the model’s name. The input is transformed to a lower dimension.

The dimension of the starting vector must be preserved. The starting vector is of floating point numbers sized 512. After some processing the vector is converted to smaller 64 sized floating point numbers. The final output of the attention mechanism must be sized 512.

In this self-attention scheme three vectors are actually required. All three vectors are sized 64, and all three are converted by separate matrix multiplication operations. The weights to convert each of the three vectors are different. For this reason the new smaller vectors are all different.

The smaller vectors individually are called q, k, and v. They can also be referred to as larger matrices. The new vector matrices are denoted as Q, K, and V. Q stands for “Query.” K stands for “Key.” V stands for “Value.” The lower-case names refer to single vectors and the upper-case refer to matrices. These are essentially batches of input.

The Query value is multiplied by the Key values from all vectors in the input. This multiplication is “dot-product” multiplication. When it is done, all keys will have low output values, except those that are closest to the Query. Then the results are passed through a softmax function. When this is complete, there will be a single vector that is close to 1 and another group of vectors that are all close to 0.

The vector produced by multiplying the softmax with the V values of every word produces a single word
vector that is close to its original value, and many others that are near zero. This formula from Vaswani et al [3] shows the process.

\[
\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V
\]

The value of \(\sqrt{d_k}\) is used to limit the size of the \(QK^T\) output. The \(d_k\) is the dimension 512. Without this, the softmax function has to deal with much larger numbers. Smaller numbers for the softmax are preferred. \(K^T\) is notation for the \(K\) vector transposed.

The function can actually perform this on large matrices with high dimensionality, in parallel. This parallel matrix operation increases the speed of training.

In the triangle in the Figure 3.3 the multiplication and selection that was just described is performed.

![Diagram](image)

Figure 3.3: Attention Output. (Data travels from the top to the bottom.)

Finally the output calculated above must be returned somehow to the input dimensionality. This is accomplished by duplicating the procedure described eight times with eight separate weights. When this is done the output of the eight separate attention mechanisms is concatenated together, returning the output to the proper size. This multi-headed approach allows different heads to learn different types of relationships and, then when they are grouped together, the learned relations are recovered and contribute to the output.
Later the output is passed through a feed forward network. It is also combined with the original input again through addition. Then the output is normalized. This ensures that the values are all within reasonable ranges. This recombination of the attention output with the original output is done throughout each Transformer layer.

This describes the encoder section. There are two other attention segments. Together these two sections combine to form the decoder section. This is repeated for each layer.
In the flow diagram, Figure 3.5, most sub-segments of the entire transformer are not described. The focus is the flow of data through the three encoder segments and the different flow of data through the decoder segments. The encoder is largely serial, while the decoder is serial and parallel. In fact each decoder segment includes a feed forward part, and all decoder and encoder parts include a residual connection where the input is added back to the output of the attention and feed forward segments.

The output of the encoder is a group of vectors the same size as the input sequence. They become the “Key” and “Value” batches below. The encoder section iterates once for every input data sequence.
3.1.4 Decoder Attention I - “Key” and “Value”

The decoder is composed of two attention mechanisms and a feed forward segment at each layer. The result of the encoder’s work is passed to the decoder and remains applied to one of the decoder’s attention mechanisms in each decoder layer. In one attention mechanism of the decoder, the “Key” and “Value” matrices are imported from the encoder.

While the encoder takes in the entire input and attends to whatever portion of that input it finds to be important, the decoder is interested in producing one output token at a time. The decoder section iterates once for every output word. Throughout this process the information from the encoder does not change. In the flow diagram one layer of the decoder is illustrated.
Illustrated in the sequence-to-sequence discussion was the importance of the single “thought vector.” The Transformer can be seen as having a thought vector also. There is a corridor of data from encoder to decoder. It is composed of a sequence of vectors the size of the input sequence or sentence. It is larger, strictly speaking, than a single vector.

Two important smaller vector-sized inputs from the encoder are ultimately required in all layers of the decoder. They represent the “Key” and “Value” matrices from the thought vector. The matrices required are the size of the smaller reduced vector. The full sized vectors are transported from the encoder and are reduced dimensionally in the decoder layers to a sequence of two smaller matrices.

These full sized vectors come from the last encoder layer’s output. Typically there will be as many decoder layers as there are encoder layers. The output from the last encoder layer is applied to the “Key” and “Value” inputs of one of the attention mechanisms in all the decoder layers.
3.1.5 Decoder Attention II - “Query”

There is another attention mechanism in each decoder layer. It works solely on data from the decoder itself. It works very much like the attention mechanism from the encoder, but it attends to every word of output as opposed to the entire input sequence. It passes its output to the attention mechanism described above. This data is lowered in dimensionality and becomes the “Query” matrix for that mechanism.

The “Key” and “Value” sequences from the encoder are a group of vectors the size of the input sequence. The “Query” matrix is the size of a single vector. This is because the decoder is interested in predicting one word at a time. This section produces a single vector as well.

3.1.6 Decoder Attention II - Masking

Input for the second decoder attention section is a group of vectors from the output generated thus far. During inference this output grows by one token with every pass through the decoder. This is how text is generated.

During training the second decoder section is masked. The mask prohibits the decoder from seeing parts of the target. This mimics the inference setup. In inference the decoder can only see up to the most recent
word it has produced.

During inference the decoder produces a sentence one token at a time. Then it adds to that sentence, one token at a time, until the decoding is finished and something like English is produced. It can attend to any part of the output it has already produced. It is concerned with producing a single token at a time. These tokens strung together are the output of the Transformer. This output should be readable.

More information about how this part of the decoder works can be found in Section 3.2. Attention in this part of the Transformer is very similar to the GPT2.

![Figure 3.9: Mask. - Decoder uses masked input during training.](image)

### 3.1.7 Input - Positional Encoding

The input of the Transformer encoder and decoder layers employ not only a word vector table, but also a positional encoding scheme. The model adds sine and cosine patterns to the input vector that it can then use to learn the position of words in a sentence.

Words that are early in the sentence have a certain appearance and words later on appear differently. The encoder and decoder use the sine and cosine waves to impart this information onto the sentence sequence.
3.1.8 Output - Feed Forward Network

At the output of the last layer of the decoder, the output vectors are processed through a linear matrix which increases the vector’s dimensionality, so that the output vector is the size as the output vocabulary dimensionality. After the linear matrix the vector is processed by a softmax function. Then the highest floating point value in the new larger vector is the index of the chosen output word.

3.1.9 Visualization - Transformer

A colorful chart is used to visualize what is happening during inference. This chart illustrates how each word attends to all the other words in the input text.

Figure 3.10: Visualized Attention – “favorite” shows attention to some but not all words in the sentence.

It is significant that words like “what” and “your” do not have strong attention to other words in the text. In a chart like this one they would show no colors on the left and light colored lines connecting the right to the left. This diagram is from the Transformer with the larger hyper-parameter set that is described in Chapter 4, trained on the movie dialog corpus.

3.2 The Generative Pre-Training 2 Model

“Generative Pre-Training 2” is a large model. It is based on the Transformer from Vaswani et al [3] but there are some major changes. The model uses the decoder portion of the Transformer without the encoder. There are other changes to the output layers. Furthermore it is pre-trained and downloadable. The GPT2 model is used to create some of our chatbots.
3.2.1 Pre-Training

In Pre-Training the authors of a model train an instance and then make the model available to the user on-line. This is helpful for the average programmer interested in Neural Networks. Training an instance of the Transformer model can use computational resources for days, and require hardware that is costly. Usually the cost of producing a trained model is prohibitively expensive.

After acquiring a trained model, programmers go on to adjust the model to their task. Adjusting a pre-trained model to a given task is called “Transfer Learning.” Many tasks lend themselves to Transfer Learning. Conceptually a model can be fine-tuned to many problems.

3.2.2 General


![Figure 3.11: GPT2 - Radford et al [11]. (Data travels from the bottom to the top.)](image)

There are several sizes of pre-trained GPT2 models, all rather large. The smallest model has 12 layers, while the Transformer model in the example from Vaswani et al [3] uses 8 layers. This model also has a hidden dimension of 768, not 512. With 8 heads this leaves a smaller dimensionality of 96 at each attention head.
3.2.3 Training

The GPT2 model is trained on text from the web, specifically Reddit. The goal for training is to show the model part of a large piece of text and then to have the model predict the next word. A mask is used in the Self Attention segment of the model during training.

Training could present the model with the text in complete form and have the model look at it through a mask. A mask is visualized in Diagram 3.9.

Each word would have an opportunity to be focused on as the “next” word. A boundary is formed between the last word and the masked area to its right. The boundary between each word and the one that follows it is examined. The model can still be trained on large batches in parallel. Words to the right of the last word and the particular boundary being examined are not available to the model.

Training is done by the developers of the model and the authors of the paper. The model is too big for individuals to train from scratch.

3.2.4 Inference

In this example creating “conditional samples” are discussed, in contrast to creating “unconditional samples.” Conditional samples rely on an input sequence for generating output. Unconditional samples have no input specified.

First a series of input tokens must be selected for the example. This series of tokens is generated from an English sentence. The sequence used will be “Good day” for this example. The words in the sequence translate into single tokens in the corpus. An input word may be made up of several tokens, but that should not be the case in this simple example.

The input context for GPT2 models is 1024 tokens. Here the input tokens, “good” and “day,” take up two spots in the input area. They are followed by an end-of-sequence token. Together they take up three spots. At that time there are 1021 spots left in the input area.

The first words are converted to tokens and are passed through the embedding matrix where they are converted to vectors. Positional encoding patterns are generated for each of the three vectors. These positional encoding patterns are created from sine and cosine waves that are concatenated together. They are added to the input tokens.

The model starts at the fourth location and attempts to generate the next token. The entire model is at this moment addressing the task of generating the next token.

One of the important processes that the input goes through is the Scaled Dot-Product Attention. This is performed at each layer. There are 12 layers in the 117M model.
All three tokens are converted to smaller vectors for each layer. Then the third vector, for the end-of-sequence token, is treated as the “Query.” The matrices for the “Key” and “Value” are assembled from the words in the input. This is done for all of the heads of each individual layer.

At this time, the model is making the transition from the third spot to the fourth spot.

The third word “Query” vector, the end-of-sequence, is compared to each other previous word “Key” vectors using dot-product multiplication. Then the result is Softmaxed, producing a single vector that is close to 1 and a group of all other vectors that are closer to 0. The result of that is multiplied by all of the three “Value” vectors. A single result is found in this way.

The output is concatenated together at each layer across all the heads at that layer. The output is recombined with the input. There are also components in each layer that do normalization. Ultimately an output vector is produced that represents what the next token should be.

The new token is placed in position four. The first three tokens are left as they are and GPT2 goes back to the start and now looks at the first four tokens as input. It will try to generate the fifth token.

The model will continue to try to generate tokens until the input area is filled and there are 1024 tokens, or a special “end-of-sequence” token is generated. The output could be anywhere from 1 to 1021 tokens. This is because the input area starts with a dimension of 1024, and there are three tokens in the original input sentence.

---

**Input Sequence:**
'good day'

**STEP 1**

1 2 3 4  
good day <eol> 2  ... 1024

**STEP 2**

1 2 3 4 5  
good day <eol> hi 2  ... 1024

**STEP 3**

1 2 3 4 5 6  
good day <eol> hi there 2  ... 1024

**Output So-Far:**
'hi there'

---

Figure 3.12: GPT2 - Inference Progress
3.2.5 Corpus

The GPT2 models are trained on a corpus called WebText, a 40GB corpus taken from the Reddit web site. All the material comes from before 2017 and all has a “karma” rating of three or better. “Karma” is a rating system used internally on Reddit.

3.2.6 Releases

In their paper, Radford et al [5] showed that their model could generate text from a seed sentence or paragraph. At that time the case was made that the largest “Generative Pre-Training 2” model should not be released because of its ability to generate text that might fool humans into believing that another person was responsible for the text. Later the larger model was released to the public.

<table>
<thead>
<tr>
<th>Size</th>
<th>Parameters</th>
<th>Layers</th>
<th>$d_{model}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>small</td>
<td>117M</td>
<td>12</td>
<td>768</td>
</tr>
<tr>
<td>medium</td>
<td>355M</td>
<td>24</td>
<td>1024</td>
</tr>
<tr>
<td>large</td>
<td>774M</td>
<td>36</td>
<td>1280</td>
</tr>
<tr>
<td>x-large</td>
<td>1.5B</td>
<td>48</td>
<td>1600</td>
</tr>
<tr>
<td>xx-large</td>
<td>8.3B</td>
<td>72</td>
<td>3072</td>
</tr>
</tbody>
</table>

When the first “Generative Pre-Training 2” models were released there were three of them. Later two more were trained. The final xx-large model was trained by NVIDIA Applied Deep Learning Research [12] and was not released to the public.

The “Generative Pre-Training 2” models also work in many circumstances in “zero-shot” mode, with no extra training to make the model suit the task. It is used “as is.”

For the chatbot the model using 117 million parameters was successful. Some programming was required to make the model output look like chatbot output, but the model itself was not modified.

Tests were done on both the small and large models. When the larger 774M model was released it was used as a substitution for the 117M model. The test worked, and returned answers that were more well formed than the small model. The larger model does not fit on a Raspberry Pi and so it was not employed here on a permanent basis.

3.2.7 Application Details

The 774M model is described in Radford et al [5] and the accompanying blog post. The model is trained on English without a stated problem, however large neural network models are usually trained for a stated
problem. Rather famously this model is used after training to generate English language text. The model takes input from the user, a premise or summary of what is to be generated. The model also takes as input a number called the “temperature.” Then the model generates output. As the “temperature” is set higher, the output is more fanciful. There is also a tune-able parameter for the output length.

For the chatbot the temperature is set to a low number. The length of the output is set to a sentence-length number of tokens. Then as input the output from the speech-to-text translator is used.

The output is not immediately useful. Traditional programming and string manipulation are employed to clean the output and render a short single sentence.

Because the input is meant to be a number of sentences and, because the architecture is Transformer-based, more information can be added to the input string. In this respect the model acts to summarize the input.

A set of three or four sentences is included with every input string. They suggest the time, the bot’s name, and the bot location and occupation. The chatbot summarizes the input. If the information is relevant then it is used by the model as output.

Surprisingly the chatbot answers most of the questions in the first person. It is felt that WebText, the Reddit corpus, has many examples of sentences in the first person.

3.2.8 Visualization - GPT2

During inference the Scaled Dot Product Attention in the GPT2 focuses on certain words as it processes input text. Here, the word “favorite” shows a relationship to many of the other words in the text.

![Visualized Attention GPT2](image)

Figure 3.13: Visualized Attention GPT2 – “favorite” shows attention to some but not all words in the sentence.

The phrase “What is your favorite color?” is often answered with “I love the colors of the rainbow.” This answer does not mention a specific color, as one might expect it should. Figure 3.13 might support this observation because “color” on the left is not heavily highlighted. Words like “what” and “your” are barely considered at this head at all.
Chapter 4

Experimental Design and Setup

4.1 Approach to the Study

Several neural network models are used in the project. One is the GRU sequence-to-sequence model, another the Transformer model for a generative chatbot and finally, the Generative Pre-Training 2 model. A rewrite for the Transformer or GPT2 model is not attempted.

In this project the chatbot code is loaded onto small single board computers as much as possible. The Raspberry Pi and the Jetson Nano are considered to be small single board computers.

Models are trained using the Pytorch and Tensorflow libraries. These models are responsible for taking in English sentences and producing English output. Another part of the typical Raspberry Pi setup includes another neural network component. Speech to text models, which this application requires, rely on large neural network resources. Speech to text resources, supplied by Google on the Google Cloud are used. To include speech to text libraries locally on the Raspberry Pi would be too costly in computation time and resources like RAM. It would also be complicated to implement technically. It could easily comprise an entire project on its own.

The speech to text service used here and the Raspberry Pi memory limitations are problematic. The chatbot function could be served from some faster machine located somewhere on the Internet. It would entail two calls from the Raspberry Pi for every sentence.

Another problem is the models use both Pytorch and Tensorflow. Tensorflow has “tensorflow-model-server” for serving models, but Pytorch has no equivalent.

It is important to note that the large GPT2 model specifically could be served from a remote computer and would operate faster. Currently on the Raspberry Pi, decoding a single sentence with the 117M parameter GPT2 takes approximately 13 seconds. However, it is preferred to install these trained models on the Raspberry Pi directly.

The 117M parameter GPT2 model fits on a Jetson Nano and can produce the response to a question in two or three seconds. The Nano is not much bigger than a Raspberry Pi. If every Raspberry Pi computer in this project was replaced with Jetson Nano computers, all of them would respond quickly and with only a natural pause before each reply.
4.2 Hardware Installation Overview

<table>
<thead>
<tr>
<th>Model Name</th>
<th>File Size</th>
<th>RAM Train</th>
<th>RAM Inference</th>
<th>Hand Trained</th>
<th>Board Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq-2-Seq/Tutorial</td>
<td>230 M</td>
<td>1.1 G</td>
<td>324 M</td>
<td>YES</td>
<td>Pi 3B</td>
</tr>
<tr>
<td>Transformer/Persona</td>
<td>25 M</td>
<td>556 M</td>
<td>360 M</td>
<td>YES</td>
<td>NONE</td>
</tr>
<tr>
<td>Transformer/Movie ††</td>
<td>550 M</td>
<td>6.5 G</td>
<td>500 M</td>
<td>YES</td>
<td>Pi 4B</td>
</tr>
<tr>
<td>GPT2 small - 117M †</td>
<td>523 M</td>
<td>-</td>
<td>2.0 G</td>
<td>NO</td>
<td>Pi 4B/Nano</td>
</tr>
</tbody>
</table>

† a large GPT2 model exists, but it is not small enough to fit on a Raspberry Pi or Jetson Nano.
†† This is referred to as the “larger” Transformer model, though it is smaller than the smallest GPT2 model.

A short description for each row is provided below.

- **Sequence-to-Sequence - Tutorial** This model uses the sequence-to-sequence architecture and the Gated Recurrent Unit component. Here the slightly modified version of the sequence-to-sequence model is based on the tutorial from Inkawhich et al [2], using the Movie Dialog corpus with a hidden size of 500 units. This model was used in one Raspberry Pi installation.

- **Transformer - Persona** This model uses a Tensorflow Transformer architecture. Coding is used to interface with the text-to-speech and speech-to-text libraries. Additional coding loads project corpus data during this training. The model parameters describe a rather small model and uses the Persona Dialog corpus. It uses 2 layers, 4 heads, and a hidden size of 128. This model was not permanently installed. It is not on a Raspberry Pi board.

- **Transformer - Movie** This model is based on the Transformer model above but uses the Movie Dialog corpus and a larger parameter set. In many ways this model is bigger than the one that uses the Transformer and the Persona corpus. It uses 6 layers, 8 heads, and a hidden size of 512. This was used in one Raspberry Pi installation.

- **GPT2 small** This model was downloaded from the internet, and fits on a Raspberry Pi 4B with the 4GB RAM option. It also fits on a Jetson Nano. Some modification was made so that model output was suitable. It uses 117M parameters, with 12 layers and a hidden size of 768. This model was used in one Raspberry Pi installation and one Jetson Nano installation.
4.3 Setup

Linux computers are used, sometimes with GPU hardware for parallel processing. The Python programming language is used. Code from this project can be run with the 3.6 or 3.7 version of Python.

When the project was started, some programming with Keras was done using Tensorflow as a backend. Keras was later discarded in favor of Pytorch and Tensorflow, although Tensorflow and Pytorch do not work together. Pytorch as a library is still under development at the time of this writing.

Some of the Generative Pre-Training 2 code uses Pytorch, and some uses Tensorflow. A repository on Github uses the GPT2 trained model with Pytorch instead of Tensorflow.

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

As an experiment, a rewrite for the sequence-to-sequence GRU was coded with little success. There is no rewrite for the GPT2 code from the Tensorflow or Pytorch repository.

4.3.1 Graphical Processing Unit vs. Central Processing Unit

A CPU has a number of cores, usually between 2 and 16. A CPU is designed to execute one command at a time, and has limitations when it comes to executing matrix multiplication.

GPUs, Graphical Processing Units, have the ability to address tasks like matrix multiplication with many more processing units at once. They speed up parallel processing and are a benefit to neural networking training.

Unfortunately, state of the art neural network models are larger than the capacity of a single GPU. Some models are trained on many GPUs simultaneously. It is not uncommon for a model to train on a computer with eight GPU cards for many days. It is possible to rent time on Amazon Web Services or Google cloud to do this training, but it can be costly.

This is addressed partially by the Transfer Learning scheme, where an already trained version is accessible to the public. The programmer downloads the model and fine tunes it to suit the task.

Here, the GRU based sequence-to-sequence model and the Tensorflow based Transformer model were trained from scratch on a CPU laptop. In the case of the Transformer, several days were required for training. In the GRU example the model trained in less than an hour.
4.3.2 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 has 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 has 4Gig of RAM.

It is the intention of this project to install several chatbot models on Raspberry Pi computers. All chosen models must be small enough in file size to fit on the Pi.

Pytorch needs to be compiled for the Pi. Speech Recognition and Text To Speech must be available. Early tests using Google’s SR and TTS services show that the Pi can support them. Details for setting up Google’s SR and TTS functions are found in the README file of this project’s Github repository.

The chosen model is trained on the desktop computer and then the saved weights and biases are transferred to the Raspberry Pi platform. The Pi is not required to do any training, only inference.

4.3.3 Jetson Nano

The Jetson Nano is another small single board computer. Though it has no WiFi, it is outfitted with an NVIDIA Graphical Processing Unit.

While the development computer uses x86_64 and the Raspberry Pi uses ARMv7, the Nano uses aarch64.
It is based on the ARM architecture with 64 bit memory units. ARMv7 uses 32 bit memory units.

It is not much bigger than a Raspberry Pi, has four Gigabytes of memory, and an Ubuntu based operating system. The operating system has Python 3.6 and special NVIDIA GPU libraries pre-installed. Pytorch can be installed.

In normal mode the board has an HDMI connector providing both video and audio output. For the project, the board was outfitted in headless mode, requiring USB connectors to provide audio input and output. Because it does not have on-board WiFi, a WiFi board must be installed, or a WiFi USB dongle used.

Google’s Speech Recognition and Text-To-Speech software works on this platform. The Nano did not need to be employed for any training. It was used solely for inference.

Figure 4.2: Jetson Nano without case.

4.3.4 Reply Time

The Google Text-To-Speech software takes negligible time to execute. The Speech Recognition software takes one or two seconds to execute, as each audio sample must be sent to a Google server for translation
into text. Because performance relies on network speeds, execution time is uncertain.

The actual model is responsible for taking question text, and returning an appropriate answer in text form to the calling function. Each model is, in turn, connected to the same SR and gTTS software.

On the development computer most of the models tested take little or no time to execute. On the Raspberry Pi only the GRU model executes instantaneously. Each of the other models requires more.

On the Jetson Nano, the largest installed model, the 117M GPT2 model requires a few seconds to execute. Reply time is determined by inference time and time for general SR server calls.

All models’ first reply is longer. This is normal and is explained by the program’s need to load a large set of matrices for the first reply. This loading overhead is not needed in the subsequent program replies.

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Board Type</th>
<th>SR Time</th>
<th>Reply Time</th>
<th>Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq-2-Seq/GRU Tutorial</td>
<td>Pi 3B</td>
<td>2s</td>
<td>0s</td>
<td>2s</td>
</tr>
<tr>
<td>Transformer/Movie Corpus</td>
<td>Pi 4B</td>
<td>2s</td>
<td>5s</td>
<td>7s</td>
</tr>
<tr>
<td>GPT2 small - 117M</td>
<td>Pi 4B</td>
<td>2s</td>
<td>13s</td>
<td>15s</td>
</tr>
<tr>
<td>GPT2 small - 117M</td>
<td>Jetson Nano</td>
<td>2s</td>
<td>3s</td>
<td>5s</td>
</tr>
</tbody>
</table>

4.3.5 Tensorflow vs. Pytorch

Tensorflow is a Google library. Pytorch has it’s roots with Facebook. Both run in a Python environment. While Pytorch offers Python objects that can be combined to create a neural network, Tensorflow has different pieces, but they cannot be examined as easily at run time. Tensorflow has a placeholder concept for inputting data and their results. Placeholders are set up at design time and are used to access data at run time.

Pytorch objects interact with Python more easily, allowing print statements in the code to show data transferred from one object to another. This is possible at run time.

Tensorflow has a tool, called Tensorboard, good for visualization which can print out graphs of data while the model trains.

4.3.6 Speech and Speech To Text

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.

Both of these resources work well on the Raspberry Pi, but configuring STT for the Pi is not trivial. The user must register a billing account with Google cloud services. In return for this registration, the user is able to download a JSON authentication file that must be copied to the Raspberry Pi.

Furthermore, an environment variable, called “GOOGLE_APPLICATION_CREDENTIALS” must be set that points to the authentication file. It must be set along with the launching of the neural network model.

The operating system on the Raspberry Pi is based on Debian Linux. In this system a file is run immediately after the basic system starts up, called “/etc/rc.local.” It is sufficient to put the environment variable there and follow it with the launching of the model. The setting of the environment variable is combined with the launching of the program in a single line of code.

### 4.3.7 Corpus Considerations

Several data sets have been collected for the training of a chatbot model. First is a corpus of movie subtitles, then the corpus described by Mazaré et al [13]. This corpus is specifically designed to train the chatbot task. This is referred to as the Persona corpus.

For the Persona corpus, the text is organized into “JSON” objects with several different repeated labels. Some of the text is meant to be used in question and answer pairs. Sentences labeled as “history” are most suited to this task.

The WebText corpus, culled from Reddit, is used for the training the GPT2 model. This material is not used in this paper directly, but it is a part of the success of the GPT2. This corpus is 40Gig of data.

### 4.3.8 Chatbot vs. Smart Speaker

A chatbot is a computer program that takes input from the user and replies as if it were a human. A chatbot is most often encountered on web sites and html pages. Input is usually in the form of typed text. Output is usually in the form of text on the screen.

A smart speaker is a term referring to a physical object that has a speaker and a microphone and some kind of computer that takes spoken input. Often the smart speaker controls lights or other appliances in a home. It will, on occasion, reply with a human voice.

For this paper the term smart speaker is used for a small computer that speaks and can be spoken to. The smart speakers in this paper do not control household appliances.
4.4 ARMv7 Build/Compile

4.4.1 Pytorch “torch” Library 1.1.0 For ARMv7

The Pytorch library is compiled for the Raspberry Pi. Several virtualization techniques are used to do this compilation. The result of those efforts is a Pytorch Python 3.7 library.

On their web site Milosevic et al [14] compile Pytorch 1.1.0 for the Raspberry Pi. The instructions called for constructing a change-root environment where a Fedora Core 30 linux system was set up. Then the ARMv7 system was used to compile the Pytorch library.

The production laptop used for development ran Ubuntu Linux. For this reason a Virtualbox emulation was set up with Fedora Core 30, and the change-root environment.

There are two problems with the resulting Python package. First there is a Python error when importing the torch library, that reads “ImportError: No module named _C.”

The build process for ARMv7 creates misnamed shared object files. To correct this, the misnamed files must be found, copied, and suitably renamed. The same outcome could be assured by making symbolic links to the misnamed files with proper names.

Three misnamed files can be found at “/usr/lib/python3.7/site-packages/torch/.” Named with the same convention, all have the ending “.cpython-37m-armhf-linux-gnu.so.” They must be renamed with the shorter “.so.” The files should be named “_C.so,” “_dl.so,” and “_thnn.so.”

The second problem occurs because, the GLIBC in the change-root environment does not match the GLIBC library in the Raspberry Pi Raspbian distribution. This produces the following error: “ImportError: /usr/lib/x86_64-linux-gnu/libstdc++.so.6: version “GLIBCXX_3.4.26” not found.” This is solved by rebuilding the package with Fedora Core 29 instead of 30.

4.4.2 Pytorch “torch” Library 1.4.0 For ARMv7

The Pytorch library is recompiled for the Raspberry Pi using Debian virtualization techniques. Because Ubuntu is a Debian derivative it is not necessary to run the process in a Virtualbox container. In addition, the files are properly named and there is no need to change anything in the directory “/usr/lib/python3.7/site-packages/torch/”.

Compiling took five hours. Time spent with the Virtualbox container was more than twice that. The time spent on the Raspberry Pi executing a single Generative Pre-Training 2 question and answer remained 13 seconds. There was no gain in that respect.

There were several small hurdles to completing the compilation. First, the “debootstrap” command
needed to be employed. Because Debian Stretch was used as the host operating system, the GLIBC compatibility problem was not faced.

Second, some dependencies needed to be installed on the “chroot” environment for Pytorch to compile. One was “libblas3.”

Third, Python 3.7 needed to be built on Stretch because program repositories used Python 2.7 and 3.5. Furthermore, the Raspbian operating system on the Raspberry Pi 4B is based on Debian Buster and used Python 3.7. After that the Git program needed to be compiled. Although Git on Stretch had an issue that was fixed upstream, it was important to use Stretch because of the GLIBC issue. Instead of using the upstream fix, Git was compiled from source.

Finally, for Pytorch, CUDA and distributed computing was disabled as neither exists on the Pi.

4.4.3 Docker Container “tensorflow-model-server” For ARMv7

The Google machine learning library for Python uses a standalone program called “tensorflow-model-server” for serving all Tensorflow models in a standard way. The program has not officially been compiled for ARMv7. A Docker image does exist that will run on ARMv7 (Maciejewski et al [15]).

Docker was run on the Raspberry Pi in the ARM environment. This terminal excerpt shows this. These commands were executed on the Pi. To take advantage of Docker one needs to log out and then log in again.

```bash
$ sudo apt-get update
$ sudo apt-get upgrade
$ curl -fsSL test.docker.com -o get-docker.sh
$ sh get-docker.sh
$ sudo usermod -aG docker $USER
```

A version of the Docker Container for the model server was run on the Raspberry Pi. A script was written that interacted with the existing ARMv7 container.

“Tensorflow-model-server” was used on the localhost Internet address, 127.0.0.1, with a port of 8500. While tensorflow-model-server is meant for serving neural network resources on the Internet, it was used on the Raspberry Pi.
Chapter 5

Experimental Results - Raspberry Pi

5.1 Experiments - Installations

The workings of a Transformer and the workings of Generative Pre-Training 2 Transformer are described in Section 3.1. While it is proposed there that they are similar, in the experiments they are totally separate.

There are several basic neural network models. One is the basic sequence-to-sequence GRU model typically used for neural machine translation. There are also two Transformers and the Generative Pre-Training 2.

There are only three Raspberry Pi boards and a Jetson Nano board used here. Three Raspberry Pi installations were successful. The Nano is used for an additional GPT2 installation, serving as a speed comparison with the Raspberry Pi.

5.1.1 Questions

This list of questions was asked of all models.

Hello.
What is your name?
What time is it?
What do you do?
What is your favorite color?
Do you like red?
Do you like blue?
What is your favorite candy?
Do you like ice cream?
Good bye.

Without considering reply time, the GPT2 model worked best. Subjectively, the first Transformer model with the Persona corpus did not perform as well as the Generative Pre-Training 2 model. It did not perform better than the Gated Recurrent Unit model either.

The model from the GRU tutorial performed well. It was better than the initial Transformer model and on par with the larger Transformer model. It was not better than the GPT2.
5.1.2 Checklist

This checklist is visited repeatedly to subjectively rate the chatbots.

1. Are all the responses in plain English? Are any of the answers gibberish?

2. Is there a variety of answers? Are all answers the same?

3. Does the model give good answers to the questions about “favorite color” and “favorite candy?” The model could have a set of easy answers that it can use for this kind of question or it considers the question separately.

4. “No” is a safe answer for many types of question as it is clearly English, it follows logically, and it is short and easy to remember. Another safe answer is “I don’t know.” Does the model use these answers at all times?

5. Does the model answer well to “Hello” and “Good bye?”

The reasons for questions 1, 2 and 5 are straight forward. The reason for question 3 is to determine how well the model understands English. If the answers to “favorite candy” and “favorite color” are the same, then the model is simple. If the answers are different, then the answer set the model uses is larger. The model might or might not know there is a difference between candy and color.

A simpler model will answer “I don’t know” for the “favorite candy” and “favorite color” questions. It can even answer questions like “do you like red” and “do you like ice cream” with “I don’t know.”

For question 4, simple “No” answers are used frequently to answer questions that would otherwise need complex sentence structure. A simpler model learns during training that “No” answers are easier to remember and could be applied in many places.

5.2 Chatbot - Gated Recurrent Unit Model

There is a model based on the sequence-to-sequence architecture proposed by Vinyals et al [1]. It is from an online tutorial by Inkawhich et al [2] trained on a large English corpus.

This model does not use a Transformer architecture. It answered a number of subjective questions appropriately.

5.2.1 Questions

This screen shot is the sample question list as it was answered by the GRU model.
> hello .
you re a good man .
> what is your name ?
smith .
> what time is it ?
i don t know .
> what do you do ?
im a comedian .
> what is your favorite color ?
i don t know .
> do you like red ?
no .
> do you like blue ?
no .
> what is your favorite candy ?
it s a big one .
> do you like ice cream ?
no .
> good bye .
good bye .

5.2.2 Checklist

1. Are all the responses in plain English? Are any of the answers gibberish?

   All the responses are in plain English. There is no gibberish.

2. Is there a variety of answers? Are all answers the same?

   Not all answers are the same.

3. Does the model give good answers to the questions about “favorite color” and “favorite candy?” The model could have a set of easy answers that it can use for this kind of question or it considers the question separately.

   It is debatable whether the answers to the questions about “favorite color” and “favorite candy” are good. It is good that the two types of questions don’t have the same answers.
4. “No” is a safe answer for many types of question as it is clearly English, it follows logically, and it is short and easy to remember. Another safe answer is “I don’t know.” Does the model use these answers at all times?

This model uses that answer at times. It does not use “no” always.

5. Does the model answer well to “Hello” and “Good bye?”

The model answers well to “Hello” and “Good bye.”

This is a reasonably good and light weight model. The memory it uses is small and it replies quickly.

5.3 Smart Speaker - Gated Recurrent Unit Model

The GRU model was installed on a Raspberry Pi 3B, allowing tests of speech-to-text and text-to-speech libraries. The RAM requirements were less than 500MB and the trained model answered questions almost instantaneously. It was outfitted with a microphone and a speaker. It was also configured to run automatically on startup. For this experiment, the Pytorch library was compiled.

The model requires access to the Internet for the exchange that the speech-to-text software has to make with the Google servers. As there was no monitor and it took some time for the model to launch, the program was coded to beep when the model was ready to accept input for the first time.

5.4 Chatbot - Transformer Model with Persona Corpus

Using the Persona Corpus a Transformer model was trained, but not pre-trained. It did not use transfer learning.

This Transformer model uses the Tensorflow library, not Pytorch. Small, it uses 2 layers, 4 heads, a hidden size of 128, and has encoder and decoder sections. The memory footprint, running below one Gigabyte, could be installed on a Raspberry Pi board.

5.4.1 Questions

Subjectively this Transformer model is not better than the original GRU from the sequence-to-sequence tutorial. It can respond to four or five sentences. The default reply sentence for unknown prompts is “Hi, how are you today?” It has answers to prompts like “hi,” “How are you?” and “What do you do?” It doesn’t answer arbitrary questions. It cannot answer ‘What is your favorite color?’ It can not tell you the time.
This is the sample question list as it was answered by the model.

> hello
hi, how are you today?
> what is your name?
hi, how are you today?
> what time is it?
I like to read a lot
> what do you do?
i’m a student
> what is your favorite color?
hi, how are you today?
> do you like red?
hi, how are you today?
> do you like blue?
hi, how are you today?
> what is your favorite candy?
hi, how are you today?
> do you like ice cream?
yes, i do
> good bye
hi, how are you today?

5.4.2 Checklist

1. Are all the responses in plain English? Are any of the answers gibberish?

All the responses are in plain English. There is no gibberish.

2. Is there a variety of answers? Are all answers the same?

There is a variety of answers. There is not a great variety though, and this is a problem for this model.

3. Does the model give good answers to the questions about “favorite color” and “favorite candy?” The model could have a set of easy answers that it can use for this kind of question or it considers the question separately.
Some of the answers are re-used and do not follow logically from the questions. The “favorite color” and “favorite candy” questions prompt the model’s default answer “Hi, how are you today?”

4. “No” is a safe answer for many types of question as it is clearly English, it follows logically, and it is short and easy to remember. Another safe answer is “I don’t know.” Does the model use these answers at all times?

The model does not use “No” or “I don’t know.”

5. Does the model answer well to “Hello” and “Good bye?”

The model has an answer for “Hello” but not for “Good bye.”

This is a poor model. It does use English language answers, but does not perform well in many other respects.

5.5 Smart Speaker - Transformer Model with Persona Corpus

This Transformer model was tested on the Raspberry Pi 4B. The model takes two minutes to boot on the Raspberry Pi, the response time is slow, and 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.

Though smart speaker installation was attempted and ultimately successful, the Raspberry Pi that was used was required for another model that worked better.

5.6 Chatbot - Transformer Model with Movie Corpus

A Transformer model was trained to use as a chatbot using the Movie Corpus. It was not pre-trained with any large corpus, and did not use transfer learning.

This is the model referred to as the larger Transformer model. It is larger than the Transformer model with the Persona Corpus but it is smaller than the smallest GPT2 model. This model uses the Tensorflow library, not Pytorch.

The Persona corpus model uses 2 layers, 4 heads, and a hidden size of 128. In contrast, the Movie corpus model uses 6 layers, 8 heads, and a hidden size of 512, and has both encoder and decoder sections.

While running, the memory footprint of the Movie Corpus model was above 1.5 Gigabytes. The model was installed on a Raspberry Pi 4B board, requiring a Python package called “Tensorflow-model-server” which had to be built from source or otherwise provided for the Raspberry Pi.
The model took seven days to train on the development computer in a x86_64 environment with a CPU based processor. The goal for training was 50,000 lines of text. After training the loss graph was consulted and the installed version was culled from the saved checkpoint at the 45,000 line point.

![Loss During Training](image)

Subjectively this Transformer model is better than the one based on the smaller hyper-parameter set and the Persona Corpus.

### 5.6.1 Questions

This is the sample question list as it was answered by the model.

- > Hello.
  hello
- > What is your name?
  i don’t know
- > What time is it?
  i don’t know
- > What do you do?
  what do you mean ?
- > What is your favorite color?
Do you like red?
no
Do you like blue?
yeah
What is your favorite candy?
i don’t know
Do you like ice cream?
yeah
Good bye.
bye

5.6.2 Checklist

1. Are all the responses in plain English? Are any of the answers gibberish?

   All the responses are in plain English.

2. Is there a variety of answers? Are all answers the same?

   There is a variety of answers.

3. Does the model give good answers to the questions about “favorite color” and “favorite candy?” The model could have a set of easy answers that it can use for this kind of question or it considers the question separately.

   The “favorite color” and “favorite candy” questions are sometimes ignored. The model does not always have original answers for these questions. It will reply with “I don’t know.” much of the time. This model does answer better than the model in Section 5.4.

4. “No” is a safe answer for many types of question as it is clearly English, it follows logically, and it is short and easy to remember. Another safe answer is “I don’t know.” Does the model use these answer at all times?

   The model does in fact use “No” or “I don’t know.” It does not use these exclusively.

5. Does the model answer well to “Hello” and “Good bye?”

   The model does have an answer for “Hello” and “Good bye.”
This is a reasonably good model and it is preferable to the model that uses the Persona Corpus.

5.7 Smart Speaker - Transformer Model with Movie Corpus

The Transformer model is installed on the Raspberry Pi. It takes two minutes to boot and 6 seconds to answer any question. It gives a tone at the end of loading the model to signal the model is finished. This is helpful for a configuration where there is no monitor. A set of LED lights is installed to show when the model is processing input and when the model can take new input.

5.8 Chatbot - Generative Pre-Training 2 Model

A pre-trained GPT2 model with the large English corpus, called “WebText,” was used to produce a chatbot. This model was not trained in the typical sense after download.

Though the large 774M model was available, the smaller 117M model was used in this test on the Raspberry Pi. It was later tested on the Jetson Nano.

For these experiments, the GPT2 was used in “zero-shot” mode. This means there was no special fine-tuning of the model in the application. Special coding for the input and output was required to operate it as a chatbot.

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. It is assumed therefore that the model was at some point exposed to the “Question/Answer” paradigm in written passages during its training.

Output from the model was usually larger in size than 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 process the output. If the “A:” character string was found at the beginning it was removed. Then the first complete sentence was used as output, while words and phrases after that were discarded.

5.8.1 Context Experiment

The model was given details that it could draw on during normal execution. There were two choices. One choice, train the model using fine-tuning and transfer-learning to recognize certain questions and to supply answers. The other choice showed the model a list of relevant facts before every input sequence. This information would be summarized with each reply.
The second choice was preferred. The text that the model was shown always included the name of the model (picked arbitrarily) along with information about the location, occupation, and time. When the model is shown the time, it is always the latest system time. Occasionally the model would report the wrong time.

5.8.2 GPT2 Fact Sheet

The GPT2 chatbot is shown a paragraph or table with each question. The table below shows the contents of the paragraph. Subject matter like that found in the “favorite color” question and the “favorite candy” question are not found in the table.

<table>
<thead>
<tr>
<th>Num</th>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>What is your name?</td>
<td>My name is Jane.</td>
</tr>
<tr>
<td>2</td>
<td>What time is it?</td>
<td>It is 12:00, January 1, 2020.</td>
</tr>
<tr>
<td>3</td>
<td>What do you do?</td>
<td>I am a student.</td>
</tr>
<tr>
<td>4</td>
<td>Where are you?</td>
<td>I am in New York.</td>
</tr>
<tr>
<td>5</td>
<td>How are you feeling?</td>
<td>I am happy.</td>
</tr>
</tbody>
</table>

5.8.3 History Experiment

From each question all input and output was concatenated and included with the next question. The goal was for the model to treat the “Q” and “A” as historical input and then refer to that information as it answered new questions. This didn’t always work as expected. At times the model was confused by the extra input and at others there was no effect.

Without this mechanism the model has no notion of temporal context. The answers to questions could not depend on the answers to past questions.

Consider the questions below.

> Do you like the color red?
I like the color red.

> What is your favorite color?
Red.

The model is asked about the color “red.” Doing so the model is encouraged to pick “red” as the answer to the next question about color. When “favorite color” is requested, “red” is the answer. Without any history the model will answer the “favorite color” question with another answer. It may answer “pink” or it may answer “the colors of the rainbow.”
From one sentence to the next the model is keeping track of the context of the conversation. History is considered. However it is not always the case that the model chooses the color recently discussed as the “favorite” color.

### 5.8.4 Program Launching

Programming that allowed the model to launch programs when directed by the user was implemented. In this way the project moved closer to commercially produced smart-speakers. The code used string manipulation and simple keyword recognition rather than the model’s neural-network. This code can be enabled when the model is run from the command line, but not for the Raspberry Pi as it runs headless. There is no memory prohibition for using the program launching code.

### 5.8.5 Summary

Subjectively this model was the best of those tested. Usually the model would answer questions about it's location, it's name, and the time, correctly. Incorrect answers did address the particular question asked. Sometimes it used alternative answers. For example, it would answer with a time but not the correct time or a city but not the correct city. The output was always sensible English.

Any subject matter could be chosen and the model would answer. The prompts did not need to be the same as the simple introductory conversation of the Transformer model.

### 5.8.6 Questions

This is the sample question list as it was answered by the model. Note that the information mentioned in the answer about the time was accurate when the test was run.

> hello
Hello.
> what is your name ?
My name is Jane.
> what time is it ?
> what do you do ?
I am a student.
> what is your favorite color ?
I love the color of the rainbow.
> do you like red?
Yes.

> do you like blue?
I do.

> what is your favorite candy?
I love candy.

> do you like ice cream?
I do.

> good bye
Good bye.

5.8.7 Checklist

1. Are all the responses in plain English? Are any of the answers gibberish?

   All the responses are in plain English.

2. Is there a variety of answers? Are all answers the same?

   There is a variety of answers.

3. Does the model give good answers to the questions about “favorite color” and “favorite candy?” The model could have a set of easy answers that it can use for this kind of question or it considers the question separately.

   The answers to the questions about “favorite color” and “favorite candy” are good and better than many other models. However, the model could have a set of answers that it can use for this kind of question. The model seems to know what candy is and that a color is something different. The answers are good but not perfect.

4. “No” is a safe answer for many types of question as it is clearly English, it follows logically, and it is short and easy to remember. Another safe answer is “I don’t know.” Does the model use these answers at all times?

   The model does not use “I don’t know” often.

5. Does the model answer well to “Hello” and “Good bye?”

   The model does have an answer for “Hello” and “Good bye.”
The model will answer with it’s name and you can tell it your name, but it is confused by the latter. It will on occasion tell you that it’s name and your name are the same. This is in part because it cannot remember what it most recently said to you or what you most recently said to it. It’s also in part because it cannot always distinguish between the “Q:” and “A:” prompts.

This is the best model tested, but it is relatively large and that aspect makes it difficult to apply in some cases.

5.9 Smart Speaker - Generative Pre-Training 2 Model

Code was added that uses text-to-speech and speech-to-text libraries, so that the model could respond to auditory cues and commands.

The model was installed on two small computers, the Raspberry Pi and the Jetson Nano. For the Raspberry Pi 4B with 4GB of RAM, the Pytorch Python library for ARMv7 was compiled. Pytorch was already available for the Nano. While execution on the production laptop was instantaneous, execution on the Raspberry Pi took 13 to 15 seconds for every response. The Nano required 3 to 5 seconds (See the discussion of “Reply Time” in Section 4.3.4.)

The program was modified to sound a tone during processing. Without this it would have been difficult to know when to speak. Also LED lights were installed to indicate when the model is processing input and when the model can take new input. Lights were not available for the Jetson Nano.
Chapter 6

Further Installations

6.1 Generative Pre-Training 2 - X Large Model

The “large” 774M model of GPT2 was ultimately released to the public. In addition, two other models were created: “x-large” had 1.5B parameters, and “xx-large,” had 8.3B parameters. The “x-large” was downloaded for testing.

Though tests from the previous sections worked on the “large” 774M model, subjectively the answers provided by the 1.5B model were better. The answers were stronger grammatically. The 1.5B model was too large for a Raspberry Pi, using 12.3 Gig of RAM while it was running with inference.

As with the smaller model and the Raspberry Pi installation, no attempt was made to do transfer learning on the model. Instead the model was shown specific information during inference with which it was free to reply if it found the information appropriate.

6.1.1 Questions

This is the sample question list as it was answered by the model.

> Hello.
Hello.

> What is your name?
My name is Jane.

> What time is it?
It is 02:17 PM October 01, 2020.

> What do you do?
I am a student.

> What is your favorite color?
I like blue.

> Do you like red?
Yes.

> Do you like blue?
yes.
> What is your favorite candy?
I like chocolate.
> Do you like ice cream?
yes.
> Good bye.
bye.

6.1.2 Checklist

1. Are all the responses in plain English? Are any of the answers gibberish?

   All the responses are in plain English. There is no gibberish.

2. Is there a variety of answers? Are all answers the same?

   There is a variety of answers.

3. Does the model give good answers to the questions about “favorite color” and “favorite candy?” The model could have a set of easy answers that it can use for this kind of question or it considers the question separately.

   The “favorite candy” question is answered in more detail than other models.

4. “No” is a safe answer for many types of question as it is clearly English, it follows logically, and it is short and easy to remember. Another safe answer is “I don’t know.” Does the model use these answers at all times?

   The model does not use “No” or “I don’t know.”

5. Does the model answer well to “Hello” and “Good bye?”

   The model has an answer for “Hello” and “Good bye.”

6.1.3 Context Experiment

As before, the model was given details in sentence form that it could use as replies to questions when appropriate. The larger model worked better if all the sentences shown to it had the “Q:” and “A:” strings prepended to them.

In each example, the information the model chose from was presented in sentence pairs. Though these sentences looked much like source/target pairs from our movie corpus, they were totally unrelated.
An example might be the presentation of its own name to the model. In the smaller GPT2 model, the model was shown the single sentence “My name is Jane.” In the larger GPT2 model’s code, the model was shown two sentences. One was “Q: What is your name?” and the other, following it directly, was “A: My name is Jane.” These conventions were followed for time of day, the model’s location, and the model’s occupation. These questions can be answered without any memory of the last question.

### 6.1.4 History Experiment

All input and output was concatenated and included with each question. This had little effect in the smaller GPT2 experiment, but the history questions were helpful with the larger models. Consider the questions below.

> Do you like the color red?
I like the color red.

> What is your favorite color?
Red.

This was the same text excerpt as from the small GPT2 model in Section 5.8.3. The “favorite color” example continued to work on the larger GPT2 model. Again, as in the small model, the chatbot was using the history in the conversation. Even though the history was consulted before answers are formulated, the output of the model was not always predictable.

### 6.1.5 Artificial Intelligence Markup Language Experiment

The GPT2 models remembered the user’s name very badly. Artificial Intelligence Markup Language was used to improve this.

AIML is actually poorly suited for handling this task. In AIML every question and response must be coded as a separate rule. For the question “How are you?” and answer “I am fine,” the AIML kernel will only answer with the programmed response if the input is an exact match. If the questions are “How are you doing?” or “How do you feel?” the kernel does not have any answers.

The GPT2 chatbot is different. Groups of questions have the same answer. Sometimes the answer could even be “I don’t know.” The chatbot could answer “I don’t know” or “I am fine” to “How are you,” “How are you doing?” and “How do you feel?” These questions can be phrased differently with the same answer. To use AIML for this task questions and answers were coded in pairs for all possible phrasings of a question.

Two situations with AIML files are below.

55
6.1.6 User Name Experiment

User name is the first example. Using “*” as a wildcard, there are a few ways to say one's name. One way is “My name is *.” Another variant is “* is my name.”

Each question is answerable with a phrase like “Hello *.” The asterisk is still a wildcard for the memorized name. An AIML file can handle this sort of question very easily.

Another question in this scenario is “What is my name?” In AIML there are very few ways to phrase this question.

The part of the AIML code that processes different questions is called the kernel. During a typical chatbot run the AIML kernel is shown all user input.

When the kernel does not match anything there is no output. When there is a match, AIML output is shown to the chatbot as the context for that input. The chatbot is then left to choose what to answer from the question and the context.

The question and the chatbot answer are both added to the history. This history is retained. Sometimes, though, keeping sentences that refer to user names confuses the bot’s answers. Even when the answer is sensible the model’s answer is not always correct, but without the AIML the model always answers all name questions with its name.

Usage Example

The user might say “Hello, my name is John.” The bot might reply “Hello.” The user might then say “What is your name?” The bot should reply “My name is Jane.” Then the user might say “What is my name?” The bot should reply “John.”

With the smaller GPT2 model the reply to this is often “Jane.” With the larger GPT2 model the bot tends to answer what the AIML suggests. In this case the answer would be “John.”

6.1.7 Internet Search Experiment

The model is asked a question and an article is downloaded from the Internet, and summarized. This is leveraging of the token size of the input to the model. It’s usually very large. It can be 1024 tokens.

The AIML required is minimal but larger than the AIML for the “user name” problem. The goal is to detect, first, when the user wants to talk about an Internet page, and second, detect when they are done.

The first case, when the user wants to talk about an Internet page, is associated with utterances like “Tell me about *” and “I want to talk about *.” The asterisk is a wildcard for what the user is interested
in. There can be other phrases that imply the user’s intention. In these cases, the text from the wildcard in the invocation is saved.

The second case, when the user is done with the Internet page and wants to return to normal operation, is associated with utterances like “Ok thanks” and “That’s enough.” There could be other phrases that cause the same operation in the AIML code.

When the AIML detects one of these signals, the program goes into a special mode of operation. The first, or “find” signal, causes operation of an Internet browser. The web search uses the text from the invocation of the “find” command. The browser yields results as they would be found using a Google search engine. The first 20 results are kept and the URL for the highest ranking Wikipedia page is retained. It is loaded and the “body” tag is scanned for “paragraph” type content. This information replaces the context input.

After that, the context is retained while the chatbot answers any questions that the user might have. The medium sized GPT2 model has some facility with this, while the larger GPT2 model is more adept.

Later, when the AIML detects the second, “restore” signal, the program switches back from special operation to normal operation. Web page contents are removed and the context is restored.

As the chatbot answers questions, the content of the conversation is stored as “history.” This is a record of the chatbot operation even when it is answering questions about a web page. Because of size limitations, the history is not included in the chatbot input when it is answering questions about a web page.

After the “restore” signal, the chatbot sees only the regular context and none of the web page material. The regular input size for all models is 1024 tokens. Though this would appear to be a large space for text, it is common for Wikipedia articles to be clipped so that the user’s question can be inserted at the end.

**Usage Example**

This example uses the musical group “The Beatles.” The user’s question is “Tell me about the Beatles.” The chatbot takes seconds to download an article from Wikipedia. Ultimately it answers “The Beatles were an English rock band formed in Liverpool in 1960.”

The user asks “How many members were in the band?” The 774M chatbot answers, “The Beatles were originally formed by John Lennon and Paul McCartney in Liverpool.” This is not the correct answer. This may be because, after clipping, that information does not exist in the context section. The user may ask other questions.

To this question, the x-large 1.5B model answers “There were four members in the band.” This is correct. When the user asks “How many members were in the band?” the model seems to be able to answer with the specific band.

To end the Q/A session the user says “OK” or “OK thanks.” Though the web page for the “Beatles”
would no longer be available to the chatbot, it could still answer “Beatles” questions. The user could ask “What are the Beatles?” The answer might be, “The Beatles were an English rock band formed in Liverpool in 1960,” even though the Wikipedia page is not available to the bot.

### 6.2 Generative Pre-Training 2 - Jetson Nano

The Jetson Nano is a small single board computer that has a NVIDIA GPU. Initial tests showed a 117M GPT2 model could answer a basic question in two to three seconds. Previously a Raspberry Pi computer took thirteen or fourteen seconds. On the host laptop, where development was done, the 117M model still outperformed the Nano, responding to a general question in one second.

The smaller 117M parameter model was not used for the Wikipedia searches that were attempted in Section 6.1. It was used for the same basic chatbot task tested on the Raspberry Pi platforms.

The Nano that the GPT2 model was installed on was outfitted with a speaker, a microphone, and a WiFi dongle. Pytorch for the Nano was installed from a community web site where version 1.5 or 1.6 was already cross-compiled for the aarch64 environment.

Response time with the Nano was five seconds. Response time on the Raspberry Pi for the 117M model was 14 seconds. (See the discussion of ‘Reply Time’ in Section 4.3.4.)
Chapter 7

Results and Observations

7.1 GRU vs. Transformer

The GRU chatbot was compared with the larger Transformer based chatbot. Using subjective qualifications the GRU model answered accurately and with more variety than the Transformer model. An important observation was that the hyper-parameter set for the Transformer model could be expanded and enlarged as needed before training. The GRU model could not be trained successfully with an arbitrarily large hyper-parameter set. The larger Transformer could be trained to produce better responses.

Additionally, the GRU model responded very quickly, while the Transformer model took longer. The time spent by the Transformer model when installed on a small computer like a Raspberry Pi could not be ignored.

Furthermore, time complexity is discussed by Vaswani et al [3]. For the GRU Recurrent Neural Network it is $n \cdot d^2$, where $n$ is the number of words in a sentence and $d$ is the dimension of the hidden vector. Complexity for the Transformer’s Scaled Dot Product Attention is $n^2 \cdot d$. Again, $n$ is the number of words and $d$ is the hidden vector’s dimension.

The maximum allowable input from the GRU is short, and in the example was 10. The maximum allowable input from the Transformer models varies but, in the 117M GPT2 model, it was 1024. In this example, the hidden vector size for the GRU model was 500 and the Transformer’s hidden size was 512.

7.2 Turing Test

The Turing Test concerns itself with the question of whether a computer is intelligent. Turing says that intelligence is too hard to describe and that, if the computer can convince you that it is intelligent, then it is.

Whether this is right is beyond the scope of this thesis. The developers of the Generative Pre-Training 2 transformer were apprehensive about their model’s ability to generate human-like text and felt the model worked too well. At first they decided not to release the largest version to the public for several months. They felt that the model could be used to spam Facebook and other social networking sites with content that was very convincing.

Ultimately the large model was released, possibly because the developers decided the model was not as
good as originally estimated.

7.3 Word and Sentence Comparisons

It is important to understand the training and inference stages. Several graphs and a discussion of the GRU model and the Transformer model and their word and sentence usage follows.

The fully trained GRU and Transformer models are set up in inference mode and their output is observed when they are exposed to the first 100,000 training question sentences from the Movie Subtitles corpus.

All the graphs start out with one of the network models and a group of 100,000 input sentences. With every change of $x$ all the sentences up until that index are fed to the model. Then they are processed by whether or not they show up repeatedly in the output so far. Then the output is summed. The changes in $y$ show this summed output.

7.3.1 Word Usage

In general the Transformer model uses only a small subset of words that it has available to it. In the example below, Figure 7.1, 100,000 input lines were tested from the training set. A small percentage of the vocabulary words were used in the output.

The graph in Figure 7.1 uses all three chatbot models. The graph uses 100,000 sentences from the Movie Dialog corpus, but from these question/answer pairs only the question is needed. Each question in turn is fed to each of the models. The output in terms of “Word Responses” is totaled at each $x$ index for the GPT2, the Transformer, and the GRU. For each $x$, the $y$ on the graph represents a running total showing how many words the model uses in its output. Pictured in this graph is also the “Total Vocabulary” number for the GPT2 model and the “Total Vocabulary” for the GRU model.
The blue line at the top of Figure 7.1 is the total words in the GPT2 vocabulary. It is much higher than the GRU or Transformer vocabulary. The green line is the total words in the GRU vocabulary. Total responses in words are shown with the three remaining lines. The GPT2 responses, in red, are higher, and the GRU and Transformer responses look like horizontal lines at the bottom of the graph.

In Figure 7.2, if the Vocabulary Total and the GPT2 responses are removed, and 100,000 sentences are tested, the outputs take the shapes of curves with a maximum of around 1,100 words. Both curves are slowly increasing at this time. The Transformer model has a vocabulary size of 8170 tokens, and the GRU model is close to that at 7826 tokens.

The graph in Figure 7.2 uses the Transformer and the GRU models. It uses 100,000 sentences from the Movie Dialog corpus but, as in the previous diagram, it uses only the “question” part of the text. For every \( x \) index there is a total of how many words are used by the models up until that point. This data represents a running total for “Total Word Responses.” This running total is graphed on the \( y \) axes.
It is possible the GRU model operates more robustly than the Transformer model, or the Transformer model may be over trained or over fitted. It also might be that the hyper parameter set is poorly adjusted. The learning rate, for example, may be too high.

### 7.3.2 Sentence Usage

A goal for this work included answers to these questions. How many fully formed responses does the given model use? Additionally, when or how many of these responses are used repeatedly?

While the GPT2 model is very large and very versatile, the Transformer and GRU models are smaller. In Figure 7.3, 100,000 inputs from the training set of the movie dialog corpus were used.

The graph in Figure 7.3 shows a running total of sentences used by the Transformer and the GRU. For every point on the $x$ axis, all sentences up to and including that index are fed to the Transformer and the GRU. At that point the graph shows a running total of sentences used more than once by the models, and how many sentences are used in total. These running totals are plotted on the $y$ axis of the graph.

These models use less than 2,000 sentences repeatedly. At the end of the study, 4,000 total sentences were used by the GRU. At some point the number of total GRU sentences that the model can produce will reach a limit below the vocabulary total.

---

**Figure 7.2: Total Word Responses - No Vocabulary Total**

![Graph showing total word responses for GRU and Transformer models](image.png)

The graph illustrates the total word responses for the GRU and Transformer models across 100,000 sentences from the movie dialog corpus.
If the graphs of the repeated sentences changed vertically as they do horizontally, the lines would be shown to increase very slowly.

While the Transformer and GRU models have similar “Repeated Sentence” graphs, they are different models and have different sizes. However as they share the same corpus, they may be learning the same task.

### 7.3.3 Maximum Sentence Values

For each of the models there is only a certain number of tokens in the model’s vocabulary. Additionally there are only so many tokens allowed in the output. For the GRU and the Transformer the output is limited to 10 tokens. For the GPT2 model there is a limit of 1024 tokens. There are only so many combinations that can be formed for a given model.

It seems that the models cannot use all of the words or tokens in their vocabulary. They can identify a number of them that they use repeatedly. This would mean the possible combinations that a model could produce is smaller.
7.4 Training and Lists

Here the Transformer and GRU models are trained on the movie dialog corpus. The Transformer model has 6 layers, 8 heads, and a hidden size of 512 units, while the GRU model has a hidden size of 500.

Initially the model learns a set of multi-purpose English answers. Then it acts as a classifier, where each input sentence is compared to the set of answers. There would be fewer answers than there are questions.

At the start the multi-purpose answers are constructed at the same time that the classification task is taking place. The model has a certain capacity and it starts to develop lists of usable answers in order to use that capacity best.

For the GPT2, full sentences may not be saved because the model may be more dynamic. For translation, for instance, it may be able to remember longer lists. In place of a list of complete responses, a list of phrases or partial responses could be combined to create translated output.

The output of the Transformer or GRU shows little intelligence. The actual utterances of the model are plain. The internal building of lists, though, may show a process that is found in some intelligent activity.
Chapter 8
Further Research and Readings

8.1 Questions and Further Research

In this paper generative chatbots were shown to work installed on small computers. An attempt is made to install the entire chatbot model on a single board computer. The smaller GPT2 model with 117M parameters specifically performs well. Some questions are raised for future research.

First, how large a model could be installed on how small a computer. Are there small computers that can serve as a platform for larger GPT2 models, and can these computers work fast enough for conversation. A solution to these problems might be serving the larger models somewhere on the internet. This is exactly what is done with GPT3 and is described in Section 8.3.

Second, how can the larger models be modified to further control or interact with things like household appliances. It might be good to somehow inform the model the state of these appliances, and then when the state is changed, the model would somehow be informed again. This could be done the same way that the time of day is presented to the GPT2 model.

Third, it would be good to try out GPT3 generally and specifically ascertain whether it handled the “History” experiment better than the GPT2. This is discussed partially in Section 5.8.3.

8.2 Winograd Schema

Winograd schema are named after Terry Winograd. His premise is that presented sentences, that follow the schema, have two meanings. A computer finds these sentences challenging to understand, and that makes them useful for the measuring of Artificial Intelligence. An example follows.

He didn’t put the trophy in the suitcase because it was too [big/small].

The programmer can choose which bracketed term to use, and must choose only one term. If “big” is chosen, then “it” is referring to the trophy. If “small” is chosen, then “it” is referring to the suitcase. Humans can easily see the pronoun “it” refers to either the suitcase or the trophy, but computers have trouble with these determinations.

The Transformer, and the Scaled Dot-product Attention that it uses, lends itself to discussion of Winograd schema, because the Attention scheme compares tokens to other tokens in the same sentence.
In chatbot examples, the Winograd schema is less relevant. However in the case of the Generative Pre-Training 2 transformer, and it’s exhaustive training, it is appropriate to consider the Winograd-style sentences.

Though it is not a classic Winograd schema, when the x-large GPT2 model is tested in chapter 6.1.7, the model is asked how many members there are in the band “The Beatles” without specifying which band the question is talking about. The x-large model answers what could be an ambiguous question. Sometimes it answers this sort of question correctly. There is a Winograd Schema Challenge and something of a formula for constructing your own Winograd schema (Wikipedia contributors). [16]

### 8.3 Generative Pre-Training 3

In 2020, OpenAI, the company responsible for training the GPT2 model, has released a new paper and set of models called GPT3. It is accessed by the public through an on-line API and private OpenAI server.

Brown et al [17] present a new model that has 175 Billion parameters and 96 layers. This is the largest of the new GPT3 models. As in the past, the model has not been released entirely to the public. Users interested in the model must apply to the company and be vetted. Accepted applications allow the user access to a client/server API that the company provides.

The model is tested in three modes, “zero-shot,” “one-shot,” and “few-shot.” A question is formulated and submitted in every mode. In ‘zero-shot’ the model is given no context. In “one-shot” the context contains a single question and answer pair. In “few-shot” the model is given several question and answer pairs similar to the question.

Results from the model are impressive. In all three modes, the model is not fine-tuned and no additional training done on the model to prepare it for a question-answering task.

The GPT2 context created for our smaller model is closest to the “few-shot” paradigm in GPT3. In contrast to GPT3, in these experiments the text from previous queries is saved and added to the context for subsequent questioning.
Abbreviations

AIML  Artificial Intelligence Markup Language

BPE  Binary Pair Encoding

GPT2  Generative Pre-Training 2

GPU  Graphical Processing Unit

GRU  Gated Recurrent Unit

gTTS  Google Text To Speech

NLP  Natural Language Processing

OOV  Out Of Vocabulary

RNN  Recurrent Neural Network

SR  Speech Recognition
Bibliography


