Project 2: Transformer-Based Chatbot Report

Hamzah Dweik, HAD190000

The University of Texas at Dallas

Natural Language Processing

Professor Karen Mazidi

April 2024

Please visit my Github for the source code!

System Description

Transformer Model

Here begins the tale of the creation of the Minecraft Guru. It all starts at the very top with the creation of the transformer model, thanks to the renowned paper "Attention is All You Need". I will skip past the data and cleaning and jump right into the architecture of the Transformer model itself, so that I may relay to what it was that allowed me to build a chatbot as impressively adept as the Minecraft Guru. My model was built using Tensorflow, a powerful library that has come to make machine learning faster and easier in recent times. To begin, lets take a look at the input processing side of our model.

- Padding Mask: to manage variable-length input sequences, I implemented a padding mask to ensure the model does not treat padding during sequence processing as input. This was crucial for maintaining the integrity of the model's attention mechanism.
- Look-Ahead Mask: ensuring the model predicts only based on known output, the look-ahead mask was designed to mask future tokens in a sequence during training, preventing the model from getting ahead of itself.

At the core of the transformer model, lies the Multi-Head Attention layer, which was meticulously coded to allow the model to focus on different positions of the input sequence simultaneously. The parallel processing is the key to a transformers efficiency. To match this complexity, we took advantage of multiple layers of encoders and decoders, each featuring a self-attention and a feedforward neural network, to better understand the complexities of the dependencies in the data. Alot of this core architecture is well-established and invariable. For that reason, I made use of some of the base code provided here. Next we have the custom learning rate scheduler for the actual training process. I employed this custom scheduler with the behvaior of increasing the learning rate linearly for the first few thousand training steps before decreasing it proportionally to the inverse root of the step number. This idea, along with my decision for optimization and loss, was inspired by a few different implementations of transformer models that I studied on github. That being said, I also decided on the Adam optimizer and sparse categorical crossentropy loss function due to their suitability for classification tasks with many classes (in this case, predicting the next word in a sentence give a vocabulary).

For the training of the model, I believe that you will be surprised to hear that I probably had over 100 hours of training in total. The first 40 hours were spent testing the model with different architectures and hyperparameters. The last 60 hours were spent training the model that ended up making it into the heart of my chatbot. Some quick notes about what ended up working for me: subword tokenization to reduce vocab size, doubling the usual number of attention heads from 8->16, and doubling the number of layers that I was finding to be common in many implementations (from 2 -> 4). These changes allowed for me to train in a reasonable amount of time and produce a model that is highly preformant and exhaustive.

Chatbot

As for the chatbot, I initially planned for nothing more than a text generation core, but found after testing that I preferred the chatbot be more than its bare bones. I decided to give my chatbot flesh by including the following mechanisms:

- Random fact handling: the chatbot is capable of responding to inquiries from the user asking for a random fact about minecraft. The chatbot proceeds by randomly selecting a question from a subset of the training dataset and passing it into the transformer model for text generation. I decided to do it this way because I wanted to avoid hard-coding the facts, and I knew that the model would spit out quality responses for training dataset questions.
- Self-inquiry handling: learning from the limitations of my last project, I decided to incorporate chatbot self-awareness. This chatbot is now capable of interpreting questions that are directed at it, rather than its niche speciality. A question like "who are you" or "are you a robot" are detected and responded to with predetermined responses like "I am nothing more than you friendly neighborhood chatbot". The ability to respond questions about itself added to the satisfaction of communicating with it, according to my brother Ahmad.
- Interest-tracking: rather than sticking with sentiment analysis, I thought that tracking user interests would be… more interesting. For this reason, the chatbot is now able to detect the topic of a question and store that information for safe keeping. Eventually, once enough questions are asked about this topic, the chatbot will comment on the users interest in the topic. My mother, Bushra, was especially surprised to see this.
- Response evaluation: the last major component for my chatbot is the ability for it to evaluate its own responses for relevance. This is done through the use of a pretrained embedding (word2vec) and cosine similarity. It essentially checks for similarity between the question vector and the answer vector and only returns the response to the user if its similarity exceeds 0.5 (I tested this with 20 different questions to try to find a reasonable boundary and came to this number). If the similarity between the answer and the question is lower than 0.5, the chatbot responds, quite humbly, with an answer of the form "I do not know".

All in all, this chatbot comprehensively responds to any and all user inquiries and does so fashionably. In the end, the performance of this chatbot really does come down to the text generation and the extensiveness of the dataset and attention to the architecture of the transformer model.

Dataset and Cleaning

Dataset Description

For my chatbot, I trained the transformer model on the "minecraft-question-answer-700k" dataset, which is a synthetic Q&A dataset specifically designed to encompass a broad range of topics about Minecraft. I decided on this dataset because I felt that it was encompassing of all things Minecraft, which would enable my chatbot to truly specialize in its niche. This rich dataset, comprising of about 694,814 rows and 47,133,624 tokens, was generated by leveraging content from over 18,000 different pages from the Minecraft Wiki, employing a synthetic data generation pipeline provided by glaive.ai.

To give a better view of the dataset, I have provided a sample entry:

{
"question": "What is the first statistic to decrease when a player performs energy-intensive actions in Minecraft?",

"answer": "Saturation is the first statistic to decrease when a player performs energy-intensive actions, and it must be completely depleted before the visible hunger meter begins decreasing.",

"source": "https://minecraft.wiki/w/Food#Nourishment_value"
}

Data Cleaning

I started off by loading the dataset from a JSON file using a custom Python function load_dataset, which opens and reads the dataset file, returning its content as a list of entries (formatted as dictionaries like the entry example above).

```
dataset = load_dataset('datasets/train.json')
```

After loading the data, the next step was to preprocess the dataset a bit to ensure that it was in line with my goals for the chatbot. Seeing that I was interested in the Q&A portion of the dataset, I started by excluding the source to trim down some of the unnecessary bulk. Next, I wanted to guarantee uniformity in the lengths of our inputs. Initially, I was using the length of the maximum question and answer to pad the rest of the input, but I found that the longest question and answer generally tended to be significantly longer than almost all other inputs. For that reason, I plotted the distribution of the lengths of our question and answers and found that taking the 75th percentile length would likely greatly increase our performance without reducing quality much. For that reason, I trimmed and padded all questions to 100 characters, and all answers to

300 characters. This preprocessing was handling by the preprocess_data() function and resulted in a Pandas DataFrame that now contained better formatted data for our training.

```
data = preprocess_data(dataset)
```

The next step was to build out our vocabularies for the questions and answers. This step is critical in allowing us to transform the textual data into numerical format that our transformer model will be able to work with. To take it a step further, I decided on using TensorFlow's SubwordTextEncoder to reduce the vocabulary size while still capturing the nuances of the language. I found this step to be necessary after finding that a single epoch would take hours to complete. Thanks to the subword tokenization process, which I initially discovered from this github, I was able to reduce the time for a single epoch down to 30 minutes (running locally on my RTX3080 TI). This tokenization process reduced my vocabulary sizes from 150k and 60k down to roughly 33k each.

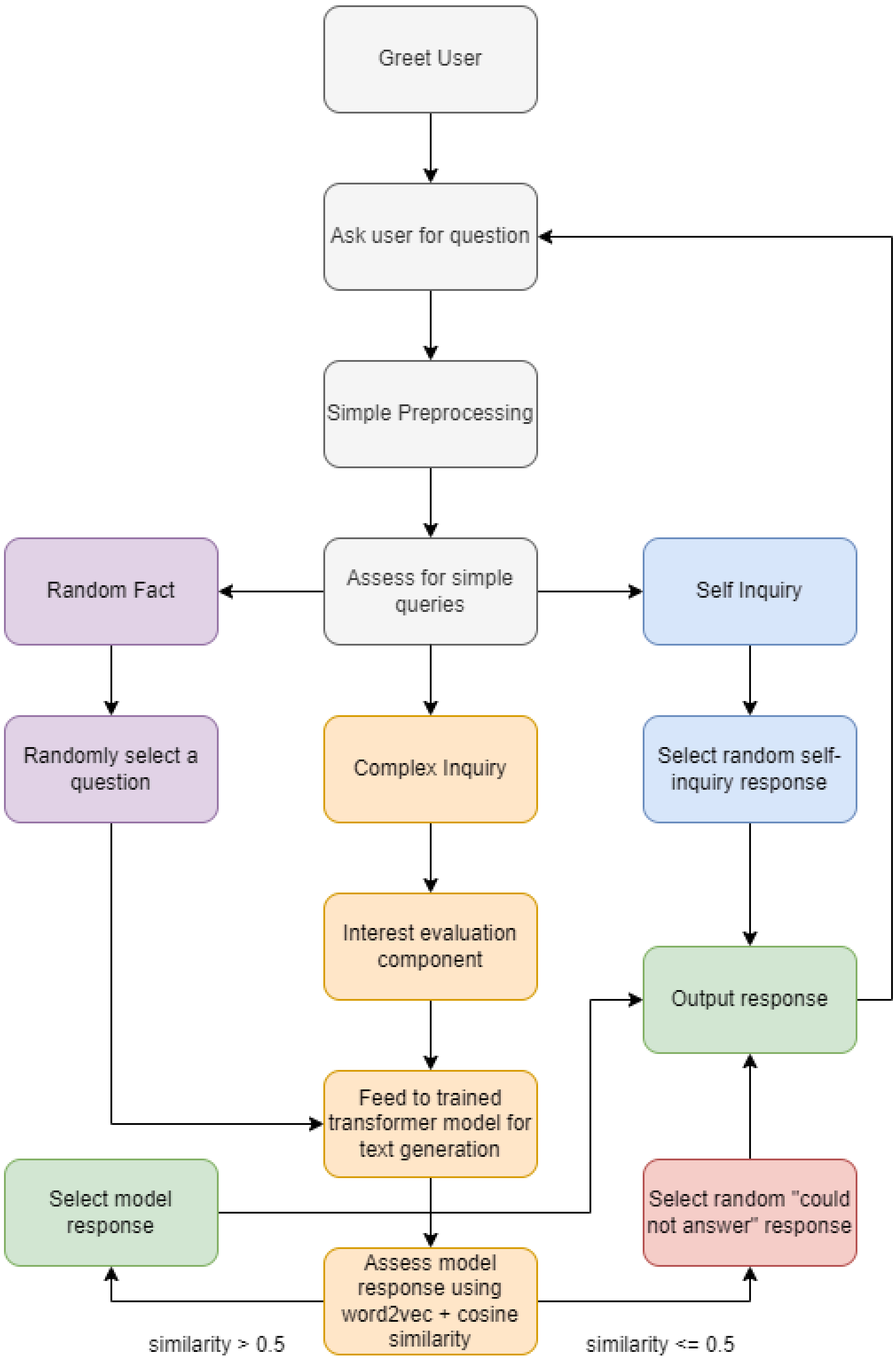
```
tokenizer_ans = create_tokenizer(train['answer'])
tokenizer_ques = create_tokenizer(train['question'])
```

With the tokenizers ready, each question and answer pair was then encoded into tokenized sequences, making them ready for model input. The last step in preparing the data came down to computing positional encodings so that I would be able to incorporate the order of the tokens into my model, since transformers have no inherent understanding of sequence order. This step helps the model understand the position of each token within a sentence. Notice here that I also computed the maximum sequence length to create an appropriate encoding.

```
pos_encoding = positional_encoding(max_seq_length, 512)
```

Through all of these steps, the dataset was transformed from raw JSON entries into a structured format that feeds directly into the transformer model. Each preprocessing step was aimed at optimizing the data for better learning outcomes in hopes that the chatbot developed would be effective and efficient in understanding and responding to user queries about Minecraft. This thorough preprocessing pipeline laid a strong foundation for training a robust and knowledgeable chatbot.

Dialog Tree


Greet User
Ask user for question
Simple Preprocessing
Random Fact
Assess for simple queries
Self Inquiry
Randomly select a question
Complex Inquiry
Select random self-inquiry response
Interest evaluation component
Output response
Feed to trained transformer model for text generation
Select model response
Select random "could not answer" response
Assess model response using word2vec + cosine similarity
similarity > 0.5
similarity <= 0.5

Sample Dialog

Remembering the user:

```
Minecrat Guru: Hey, I am the minecraft guru, Who do I have the pleasure of speaking with today?
User: Ham
Minecrat Guru: Welcome back Ham!
Minecrat Guru: If you've got questions about Minecraft, you're in the right place. I perform best
with well-formulated and detailed questions. You can type 'q' at any point to stop talking to me.
For now, do you have any queries regarding Minecraft strategies or updates?
Ham: What are some of the different types of animals that are in the game?
```

New user:

```
Minecrat Guru: Good evening! I am the minecraft guru, Who do I have the pleasure of speaking with today?

User: Adam
Minecrat Guru: It's nice to meet you Adam.
Minecrat Guru: I'm a bot with a wealth of knowledge about Minecraft. I perform best with well-formulated
 and detailed questions. You can type 'q' at any point to stop talking to me. For now, let me know what
you're curious about in relation to Minecraft.
Adam: im new to minecraft, how do i build a pickaxe
```

Notice how well the model is able to differentiate between questions, even when only one word is changing between them. Also, if you look near the end, you can see that after asking many questions about mobs, the chatbot comments on the users interest in mobs.

```
Minecraft Guru: Do you have any other questions?
Ham: Is the wither considered a mob?

Minecraft Guru: A wither is a boss mob that can be found in the game Minecraft.
Minecraft Guru: Ham, do you have a follow-up question or another topic in mind?
Ham: Is a skeleton considered a mob?

Minecraft Guru: A skeleton horse is a undead mob that can be found in the game.
Minecraft Guru: Do you have any other questions?
Ham: Is a spider considered a mob?

Minecraft Guru: A spider is a type of mob that can be found in the game Minecraft.
Minecraft Guru: Ham, do you have a follow-up question or another topic in mind?
Ham: Is a zombie considered a mob?

Minecraft Guru: A zombie is an undead mob that attacks players and other mobs.
Minecraft Guru: Do you have any other questions?
Ham: What are some of the different types of mobs that are in the game?
Minecraft Guru: Wow, you really seem interested in Mobs (Enemies and Animals)!

Minecraft Guru: In Minecraft, mobs that are spawned in the game include zombies, skeletons, spiders, a
nd endermen. These mobs are considered to be part of the game's ecosystem, and can be found in various
 biomes and structures.
Minecraft Guru: Do you have any other questions?
Ham:
```

Even with as much training as we did, there are some occasionally incoherences and hallucinations.

```
Minecraft Guru: Do you have any other questions?
Ham: At what level can you find diamond in minecraft?

Minecraft Guru: You can find diamond ore in diamond ore by mining it with an iron pickaxe or better.
Minecraft Guru: Ham, do you have a follow-up question or another topic in mind?
Ham: What is the recipe for cake?

Minecraft Guru: The cake is a decorative item that can be placed in the game world.
Minecraft Guru: Do you have any other questions?
Ham:
```

Tell me a random fact, oh chatbot.

```
Minecraft Guru: The random tick rate is always 20 ticks per second.
Minecraft Guru: What else would you like to know?
Ham: Tell me a random fact about minecraft

Minecraft Guru: You can obtain diamonds by mining them with a diamond pickaxe in Minecraft.
Minecraft Guru: Ham, do you have a follow-up question or another topic in mind?
Ham: Tell me another random fact

Minecraft Guru: In the Bedrock Edition of Minecraft, issues related to the "Golden Carrot" are maintai
ned on the bug tracker. Report issues there.
Minecraft Guru: Is there anything else you're curious about, Ham?
Ham: Tell me yet another random fact

Minecraft Guru: A quartz pillar (in Java Edition) or pillar quartz pillar (in Bedrock Edition) is a va
riant of the pillar quartz pillar.
Minecraft Guru: Ham, do you have a follow-up question or another topic in mind?
Ham:
```

Irrelevant questions? Not a problem.

```
Ham: What is the best way to learn python?

Minecraft Guru: I'm unsure how to answer that effectively. Can you phrase it differently?
Minecraft Guru: Is there anything else you're curious about, Ham?
Ham: Who is lebron james?

Minecraft Guru: I'm sorry, I couldn't provide a useful answer to that. Could you rephrase the question
 more clearly?
Minecraft Guru: Do you have any other questions?
Ham:
```

Self-inquiry questions? Also handled.

```
Minecraft Guru: Do you have any other questions?
Ham: Who are you?

Minecraft Guru: Call me your Minecraft guide, ready to assist with any information you need.
Minecraft Guru: Do you have any other questions?
Ham: Tell me about yourself

Minecraft Guru: Hello! I'm crafted by humans to assist in your Minecraft adventure.
Minecraft Guru: What else would you like to know?
Ham: Are you a robot?

Minecraft Guru: I'm your friendly neighborhood chatbot, designed to assist you with Minecraft queries.
Minecraft Guru: Ham, do you have a follow-up question or another topic in mind?
Ham: 
```

Sample User Models

From this view, you can see how the user models look. They are able to effectively keep track of a user's interest by noting how often they ask questions about it. In this case, you can see that Adam asked questions with much more breadth than Ham but Ham asked at greater depth. In this case, the chatbot would be able to tell that Ham is particularly interested in 'Mobs'.

```
Name: Ham
Interests: defaultdict(<class 'int'>, {'Mobs (Enemies and Animals)': 6, 'Farming and Food Production': 1, 'Mi
ecraft Marketplace': 1, 'Biomes and Exploration': 1, 'World Generation': 1})
Name: Adam
Interests: defaultdict(<class 'int'>, {'World Generation': 1, 'Environmental Effects': 1, 'Game Mechanics': 1,
 'Trading with Villagers': 1, 'Music and Sounds': 1, 'Texture Packs and Skins': 1, 'PvP (Player vs. Player)':
1, 'Biomes and Exploration': 1, 'Minecraft Events (like Minecon)': 1, 'Events and Competitions': 1})
```

Model Evaluation

Each person will be asked the following questions:
1. **Response Accuracy** - "The chatbot consistently provides accurate and detailed information."
2. **Response Relevance** - "The responses from the chatbot are always relevant to my questions."
3. **Interaction Quality** - "Interacting with the chatbot feels engaging and natural."
4. **Timeliness of Responses** - "The chatbot provides timely and prompt responses."
5. **User Satisfaction** - "I am satisfied with the overall experience and information provided by the chatbot."

They will then score each of the attributes of the chatbot using the following metric:

1 - Strongly Disagree
2 - Disagree
3 - Neutral
4 - Agree
5 - Strongly Agree

	Accuracy	Relevance	Quality	Timeliness	Satisfaction	Averages
Hamzah	5	5	4	3	5	**4.4**
Bushra	4	4	4	3	5	**4**
Adam	5	5	5	5	5	**5**
Ahmad	4	5	5	4	5	**4.6**
Averages	**4.5**	**4.75**	**4.5**	**3.75**	**5**	**4.5**

Therefore, this chatbot receives an overall evaluation score of 4.5/5. This is very high considering the use of a transformer model on relatively small data. When starting, I was expecting far more hallucination and gibberish than actually produced. Let's do a quick breakdown of what I believe to be its strengths and weaknesses:

- Strengths: quality of response, relevance, being able to exhaustively answer inquiries
- Weaknesses: timeliness, occasionally hallucinations/incoherence

Given more time, I would try to implement data streaming to elimination issues with timeliness. It takes no more than 2 seconds, but it still detracts from the user experience.