

UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI



BACHELOR THESIS

Developing pre-trained models and other NLP models for chatbot systems

Author:

OKETOLA PRAISE OLUWABAMISE

Supervisors:

Dr. Trần Giang Sơn

Dr. Phạm Quang Nhật Minh

*A thesis submitted in fulfilment of the requirements
for the degree of*

Information and Communication Technology (ICT)

October 11, 2020

Contents

Contents	i
List of Figures	iii
List of Tables	iv
Abstract	v
1 Introduction	1
1.1 Context and Motivation	2
1.2 Internship Objectives	3
2 Literature Review	4
2.1 RNNs and LSTMs	4
2.1.1 RNN	4
2.1.2 LSTM	8
2.2 Sequence-to-Sequence Generation	11
2.3 Transformers	14
2.4 Tokenizer	16
2.4.1 Tokenization	16
2.4.2 MeCab	17
2.4.3 Byte-Pair Encoding	17
3 Methodology	19
3.1 Resources	19
3.1.1 DataSet	19
3.1.2 Data Preprocessing	20
3.2 Evaluation Metrics	21
3.2.1 Perplexity	21
3.2.2 Cross entropy	22
3.3 Model Architecture	23
3.3.1 Model specifications	24
3.3.2 Fine-tuning details	24
3.3.3 Generation Process	24
4 Experiments and Testing	25
4.1 Text Completion	25
4.2 Next word prediction	29

4.3	Dialogue	30
4.3.1	One Turn Dialogue	31
4.3.2	Multi Turn Dialogue	33
4.3.3	Multi Turn Dialogue With Persona	37
5	Results and Discussion	40
5.1	Loss and Perplexity	40
5.2	Human Evaluation	42
5.3	Performance	42
5.3.1	Inference Time	42
5.3.2	Memory	43
6	Conclusion	44
6.1	Conclusion	44
6.2	Further Improvements	44
	Bibliography	46

List of Figures

1.1	Messages interpreted as intent	1
1.2	Existing Dialogue Systems	2
2.1	Feed Forward Neural Network	4
2.2	Recurrent Neural Network	5
2.3	Breaking up a sentence into word sequences	5
2.4	Breaking up a sentence into word sequences 2	5
2.5	Breaking up a sentence into word sequences 3	6
2.6	Breaking up a sentence into word sequences 4	6
2.7	Breaking up a sentence into word sequences 5	7
2.8	Gradients shrink as it back-propagates down	7
2.9	Gradient Clipping	8
2.10	Forget Gate	8
2.11	Forget gate operations	9
2.12	Input gate operations	9
2.13	Output gate operations	10
2.14	Cell gate operations	10
2.15	LSTM activation function	10
2.16	Machine translation	11
2.17	Speech recognition	11
2.19	(left) scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel	15
2.20	Basic Tokenization	16
2.21	Japanese Tokenization	17
2.22	BPE Tokenization	18
3.1	Model architecture.	23
3.2	Model configurations	23
4.1	One Turn Dialogue Model Loss	32
4.2	Multiturn Turn Dialogue Model Loss	34
4.3	Multi Turn Dialogue With Persona Loss	37
5.1	Model Loss	41
5.2	Model Perplexity	41

List of Tables

3.1	Language Model Dataset Description	21
4.1	One Turn Dataset Description	31
4.2	Multi Turn Dataset Description	31
5.1	Model Loss and Perplexity	40

Abstract

Chatbots are increasingly being deployed to provide round-the-clock support and to increase customer engagement. Most chatbots use bot building frameworks, they follow a standard approach that requires one to build and train an intent model to recognize user input. Intent models are trained in a supervised setting with a collection of textual utterance and intent label pairs. Gathering a substantial and wide coverage of training data for different intent is a bottleneck in the bot building process. Moreover, the cost of labeling a hundred to thousands of conversations with intent is a time consuming and laborious job.

In this thesis, a research will be conducted on the building of an open domain, conversational dialogue systems based on large dialogue corpora using a language model. Language models are based on a probabilistic description of language that estimates the relative likelihood of different phrases . This can lead to a large possibility for realistic and flexible interactions. There have been substantial gains on NLP tasks and benchmarks by pre-training on a large corpus of text followed by fine-tuning on a specific task. In this case, fine tuned for Japanese chatbot systems. The Japanese Language will be utilized to create the model, with the purpose of providing support to Japanese products which utilizes chatbot systems for different purposes like increasing customer interaction, increasing sales and reducing cost, . It can also be utilized as a multi lingual support to products used by a wide variety of audience.

However, Japanese grammar does not have Latin characters, compared to the English language which has Latin characters from A to Z. It has its own 3 types of characters: Hiragana, Katakana, and Kanji. Hiragana and Katakana script each has 46 and 45 characters. This in turn increases the difficulty to create a top quality model.

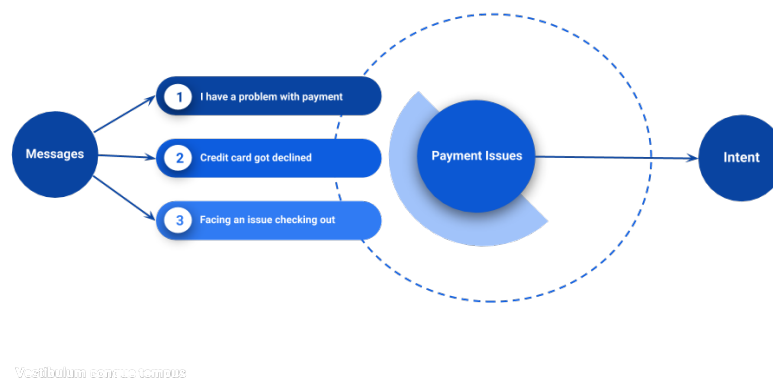
Chapter 1

Introduction

Developing an intelligent dialogue system that not only emulates human conversation, but also answers questions on a wide variety of topics and fulfills complex tasks such as reading comprehension and common sense reasoning, has been one of the longest running goals in AI. The goal has remained elusive until recently .

In the past few years, there has been a growing community and business enthusiasm for conversational frameworks and new methodologies. A key step towards designing a task-oriented conversational model is to identify and understand the intention from a user utterance. An intent in a conversational model maps semantically similar sentences to a high-level abstraction for a chatbot that can generate a similar response or a similar action.

FIGURE 1.1: Messages interpreted as intent



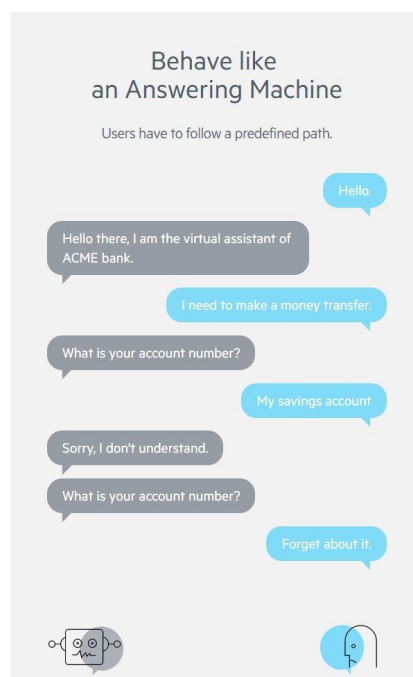
For example, “i have a problem with payment”, “credit card got declined”, “facing an issue checking out” can be interpreted as intent payment issues. The current crop of bot-building

frameworks requires annotated data for building an Intent model and it is well known that collecting high-quality training data with high coverage for each intent is difficult. The developers and domain experts typically consider past chat logs between human-human or human-computer as a valuable resource and carry out an extensive manual process of intent labeling.

1.1 Context and Motivation

Current conversational systems bring about new challenges: they tend to output dull and generic responses such as “I don’t know what you are talking about”; they lack a consistent or a coherent persona; they are usually optimized through single-turn conversations and are incapable of handling the long-term success of a conversation; and they are not able to take the advantage of the interactions with humans. The process of intent discovery and training data creation is effort-intensive. Collecting high-quality data from past conversations has a few challenges. First, intent classes are highly skewed; certain areas are discussed more often than others. Second, for many industrial scenarios, the number of unknown intent categories may vary from hundred to thousands making the task of actual conversation annotation very difficult. For convenience, domain experts often provide synthetic utterance samples for intent training in the bot frameworks and classifiers can not take full advantage of the previous chat logs.

FIGURE 1.2: Existing Dialogue Systems



1.2 Internship Objectives

In this thesis, the goal is to create a general language model, then finetune it to perform other tasks like one-turn and multi-turn dialogue systems. To create the language model, a large amount of Japanese language corpus is needed. Then the corpus will be preprocessed as it is traditionally an important step for natural language processing (NLP) tasks. It transforms text into a more digestible form so that machine learning algorithms can perform better. Initial preprocessing steps are to remove HTML tags, links, emails, duplicated sentences, duplicated words and removal of invalid character encoding. After the initial preprocessing steps, the corpus has to be tokenized with sentences capped at 1024 max tokens and 30 min tokens.

Afterwards, when the language model has been trained, it will be finetuned for a one turn, multi-turn conversational systems and multi-turn with persona.

Chapter 2

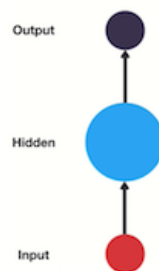
Literature Review

2.1 RNNs and LSTMs

2.1.1 RNN

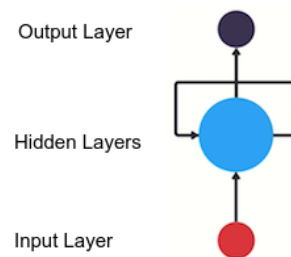
Recurrent neural networks (RNN) are a neural network architecture which is specifically designed to handle sequential data [4], in recent years have been successfully applied to language processing [6]. RNN's have this abstract concept of sequential memory, Well, below is a traditional neural network also known as a feed-forward neural network.

FIGURE 2.1: Feed Forward Neural Network



It has its input layer, hidden layer, and the output layer. To get a feed-forward neural network to be able to use previous information to effect later ones, a loop in can be added to the neural network that can pass prior information forward.

FIGURE 2.2: Recurrent Neural Network



And that’s essentially what a recurrent neural network does. An RNN has a looping mechanism that acts as a highway to allow information to flow from one step to the next.

This information is the hidden state, which is a representation of previous inputs. Let’s run through an RNN use case to have a better understanding of how this works. Let’s say our aim is to build a chatbot and the chatbot can classify intentions from the users inputted text.

To tackle this problem. First, there must be an encoding of the sequence of text using an RNN. Then, feed the RNN output into a feed-forward neural network which will classify the intents. Ok, so a user types in... what time is it?. To start, the sentence will be broken up into individual words. RNN’s work sequentially so it will be fed one word at a time.

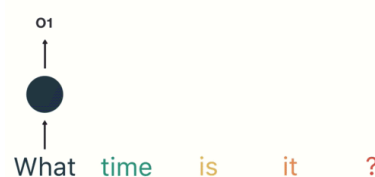
The first step is to feed “What” into the RNN. The RNN encodes “What” and produces an output.

FIGURE 2.3: Breaking up a sentence into word sequences

What time is it ?

For the next step, the word “time” and the hidden state from the previous step will be fed into the network. The RNN now has information on both the word “What” and “time.”

FIGURE 2.4: Breaking up a sentence into word sequences 2



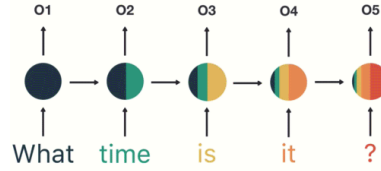
The process will be repeated, until the final step. You can see by the final step the RNN has encoded information from all the words in previous steps.

FIGURE 2.5: Breaking up a sentence into word sequences 3



Since the final output was created from the rest of the sequence, we take the final output and pass it to the feed-forward layer to classify an intent.

FIGURE 2.6: Breaking up a sentence into word sequences 4



Given a sequence of word tokens $\{y_1, y_2, \dots, y_N\}$, where each word $y_t, t \in [1, N]$ is associated with a K -dimensional vector representation x_t . RNN associates each time step t with a hidden vector representation h_t , which can be thought of as a representation that embeds all information of previous tokens, i.e., $\{y_1, y_2, \dots, y_t\}$. h_t is obtained using a function g that combine the previously built presentation for the previous time-step $t - 1$, denoted as h_{t-1} , and the representation for the word of current time-step x_t :

$$h_t = g(h_{t-1}, x_t) \quad (2.1)$$

The function g can take different forms, with the simplistic one being as follows:

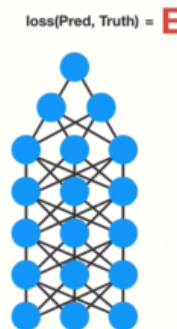
$$g(h_{t-1}, x_t) = \sigma(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t) \quad (2.2)$$

where $W_{hh}, W_{xh} \in \mathbb{R}^{K \times 2K}$. Popular choices of σ are non-linear functions such as sigmoid, tanh or ReLU. It can be seen that the dimensionality of h_t is constant for different t .

Two severe issues problems with RNNs are the gradient exploding [1] problem and the gradient vanishing problem[9]. Vanishing gradient is due to the nature of back-propagation; an algorithm used to train and optimize neural networks. To understand why this is, let's take a look at the effects of back propagation on a deep feed-forward neural network. Training a neural network has three major steps. First, it does a forward pass and makes a prediction. Second, it compares the prediction to the ground truth using a loss function. The loss function outputs an error

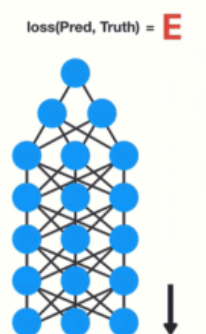
value which is an estimate of how poorly the network is performing. Last, it uses that error value to do back propagation which calculates the gradients for each node in the network.

FIGURE 2.7: Breaking up a sentence into word sequences 5



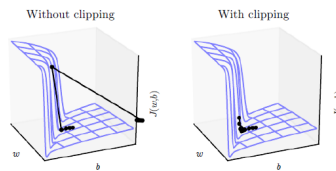
The gradient is the value used to adjust the networks internal weights, allowing the network to learn. The bigger the gradient, the bigger the adjustments and vice versa. Here is where the problem lies. When doing back propagation, each node in a layer calculates it's gradient with respect to the effects of the gradients, in the layer before it. So if the adjustments to the layers before it is small, then adjustments to the current layer will be even smaller. That causes gradients to exponentially shrink as it back propagates down. The earlier layers fail to do any learning as the internal weights are barely being adjusted due to extremely small gradients. And that's the vanishing gradient problem.

FIGURE 2.8: Gradients shrink as it back-propagates down



And when gradients explode, the gradients could become NaN because of the numerical overflow or we might see irregular oscillations in training cost when we plot the learning curve. A solution to fix this is to apply gradient clipping; which places a predefined threshold on the gradients to prevent it from getting too large, and by doing this it doesn't change the direction of the gradients it only change its length.

FIGURE 2.9: Gradient Clipping



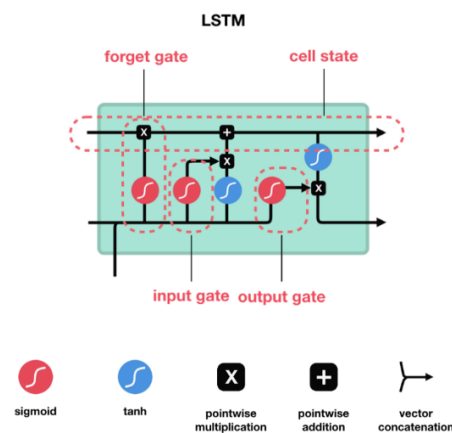
These two issues render RNN models incapable of capturing the long-term dependency for long sequences.

In conclusion, Recurrent Neural Networks suffer from short-term memory. If a sequence is long enough, they'll have a hard time carrying information from earlier time steps to later ones. So if you are trying to process a paragraph of text to do predictions, RNN's may leave out important information from the beginning. During back propagation, recurrent neural networks suffer from the vanishing gradient problem. Gradients are values used to update a neural network's weights. The vanishing gradient problem is when the gradient shrinks as it back propagates through time. If a gradient value becomes extremely small, it doesn't contribute too much learning.

2.1.2 LSTM

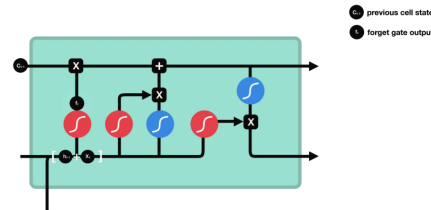
Long Short Term Memory (LSTM) first introduced in [3], and adapted, used, and further explored by many others. They were created as the solution to short-term memory. They have internal mechanisms called gates that can regulate the flow of information. Both RNNs and LSTMs pass data as it propagates forward. However, unlike in RNNs, LSTMs make use of gates to decide if it should keep or forget information. An LSTM cell consists of a cell state and input, forget and output gates which make use of several activation functions.

FIGURE 2.10: Forget Gate



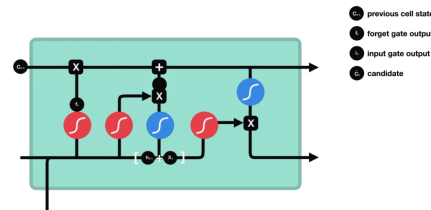
Forget gate decides which information should be kept and which should be discarded. Information from the previous hidden state and information from the current input is passed through the sigmoid function. Values come out between 0 and 1. The closer to 0 means to forget, and the closer to 1 means to keep.

FIGURE 2.11: Forget gate operations



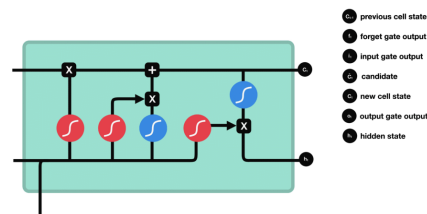
Input gate updates the cell state. To update the cell state, we have the input gate. First, the previous hidden state and current input will be passed into a sigmoid function. That decides which values will be updated by transforming the values to be between 0 and 1. 0 means not important, and 1 means important. We also pass the hidden state and current input into the tanh function to squish values between -1 and 1 to help regulate the network. Then the tanh output will be multiplied with the sigmoid output. The sigmoid output will decide which information is important to keep from the tanh output.

FIGURE 2.12: Input gate operations



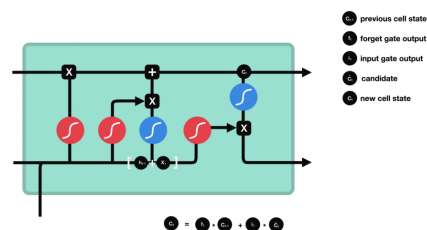
Output gate decides what the next hidden state (contains information on previous inputs) should be. Remember that the hidden state contains information on previous inputs. The hidden state is also used for predictions. First, the previous hidden state and the current input will be passed into a sigmoid function. Then the newly modified cell state will be passed to the tanh function. Then multiply the tanh output with the sigmoid output to decide what information the hidden state should carry. The output is the hidden state. The new cell state and the new hidden is then carried over to the next time step.

FIGURE 2.13: Output gate operations



Cell state acts as a highway that transports relative information along the sequence chain. The cell state gets pointwise multiplied by the forget vector. This has a possibility of dropping values in the cell state if it gets multiplied by values near 0. Then the output from the input gate will be taken to do a pointwise addition which updates the cell state to new values that the neural network finds relevant. That provides the new cell state.

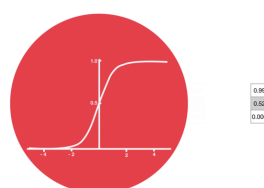
FIGURE 2.14: Cell gate operations



The two activation functions used are Sigmoid and Tanh.

Sigmoid squishes values between 0 and 1. That is helpful to update or forget data because any number getting multiplied by 0 is 0, causing values to disappear or be “forgotten.” Any number multiplied by 1 is the same value therefore that value stays the same or is “kept.” The network can learn which data is not important therefore can be forgotten or which data is important to keep.

FIGURE 2.15: LSTM activation function



Tanh — Similar to the sigmoid function but this squishes values between -1 and 1 instead of between 0 and 1 which is done in the sigmoid function. These gates can learn which data in a sequence is important to keep or throw away. By doing that, it can pass relevant information down the long chain of sequences to make predictions.

Another problem with RNNs, and LSTMs, is that it's hard to parallelize the work for processing sentences, since processing is done word by word. Not only that but there is no model of long and short range dependencies.

To summarize, LSTMs and RNNs present 3 problems:

- Sequential computation inhibits parallelization
- No explicit modeling of long and short range dependencies
- “Distance” between positions is linear

2.2 Sequence-to-Sequence Generation

Sequence to Sequence model have recently achieved significant results on pretty complex tasks like machine translation, video captioning, question answering etc. For instance, seq2seq model powers applications like Google Translate, voice-enabled devices and online chatbots.

FIGURE 2.16: Machine translation

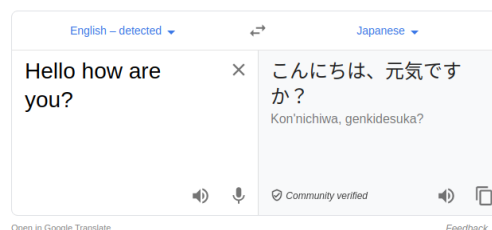


FIGURE 2.17: Speech recognition



The SEQ2SEQ model was first proved to yield good performance in machine translation [14], and has been successfully extended to multiple natural language generation tasks such as text summarization [11]. A sequence to sequence model aims to map a fixed-length input with a fixed-length output where the length of the input and output may differ.

For example, translating “Hello how are you?” from English to Japanese has input of 4 words and output of 10 symbols (こんにちは、元気ですか?). Clearly, a regular LSTM network can’t be used to map each word from the English sentence to the Japanese sentence. This is why the

sequence to sequence model is used to address problems like that one. How does it work? The model consists of 3 parts: encoder, intermediate (encoder) vector and decoder.

Encoder: A stack of several recurrent units (LSTM or GRU cells for better performance) where each accepts a single element of the input sequence, collects information for that element and propagates it forward. In question-answering problems, the input sequence is a collection of all words from the question. Each word is represented as x_i where i is the order of that word. The hidden states h_i are computed using the formula:

$$h_t = f(W^{(hh)}h_{t-1} + W^{(hx)}x_t) \quad (2.3)$$

This simple formula Eq(2.3) represents the result of an ordinary recurrent neural network. As you can see, the appropriate weights will be applied to the previous hidden state $h_{(t-1)}$ and the input vector x_t .

Encoder Vector: This is the final hidden state produced from the encoder part of the model. It is calculated using the formula above. This vector aims to encapsulate the information for all input elements in order to help the decoder make accurate predictions. It acts as the initial hidden state of the decoder part of the model.

Decoder: A stack of several recurrent units where each predicts an output y_t at a time step t . Each recurrent unit accepts a hidden state from the previous unit and produces an output as well as its own hidden state. In the question-answering problem, the output sequence is a collection of all words from the answer. Each word is represented as y_i where i is the order of that word. Any hidden state h_i is computed using the formula:

$$h_t = f(W^{(hh)}h_{t-1}) \quad (2.4)$$

As you can see, the previous hidden state is used to compute the next one. The output y_t at time step t is computed using the formula:

$$y_t = \text{softmax}(W^S h_t) \quad (2.5)$$

The outputs are calculated using the hidden state at the current time step together with the respective weight $W^{(S)}$. Softmax is used to create a probability vector which will help us determine the final output (e.g. word in the question-answering problem). The power of this model lies in the fact that it can map sequences of different lengths to each other. As you can see the inputs and outputs are not correlated and their lengths can differ. This opens a whole new range of problems which can now be solved using such architecture.

The SEQ2SEQ model can be viewed as an extension of the language model, where y is the target sentence, and the prediction of current word y_t in y depends not only on all preceding words $y_1, y_2, \dots, y_{t-1}$, but also on a source input x . Each sentence concludes with a special end-of-sentence symbol EOS. For example, in French-English translation, the English word to predict not only depends on all the preceding words, but also depend on the original French input; as another example, the following word in a dialogue response depends both on preceding words in the response and the message input. The SEQ2SEQ model was first proved to yield good performance in machine translation and has been successfully extended to multiple natural language generation tasks such as text summarization, parsing, image caption generation, etc. More formally, in SEQ2SEQ generation tasks, each input $x = x_1, x_2, \dots, x_{n_x}$ is paired with a sequence of outputs to predict:

$y = y_1, y_2, \dots, y_{n_y}$. A SEQ2SEQ generation model defines a distribution over outputs and sequentially predicts tokens using a softmax function:

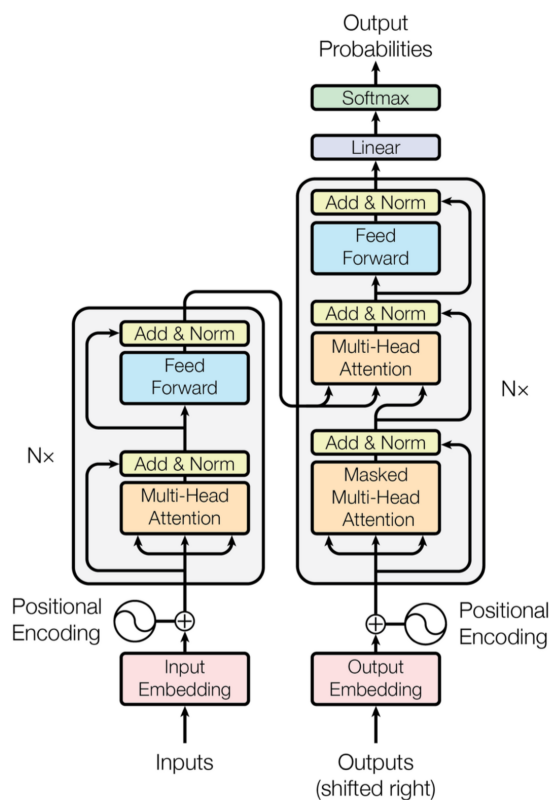
$$P(y|x) = \prod_{t=1}^{n_y} p(y_t|y_1, y_2, \dots, y_{t-1}, x) \quad (2.6)$$

More specifically, a standard SEQ2SEQ model consists of two key components, an encoder, which maps the source input x to a vector representation, and a decoder, which generates an output sequence based on the source sentence. Both the encoder and the decoder are multi-layer LSTMs. To enable the encoder to access information from the encoder, the last state memory of the encoder is passed to the decoder as the initial memory state, based on which words are sequentially predicted using a softmax function. Commonly, input and output use different LSTMs with separate compositional parameters to capture different compositional patterns.

2.3 Transformers

Transformers, invented in 2017, [16] introduced a new approach — attention modules. Instead of processing tokens one by one, attention modules receive a segment of tokens and learn the dependencies between all of them at once using three learned weight matrices — Query, Key and Value — that form an Attention Head. The Transformer network consists of multiple layers, each with several Attention Heads (and additional layers), used to learn different relationships between tokens. As in many NLP models, the input tokens are first embedded into vectors.

FIGURE 2.18: The Transformer - model architecture.

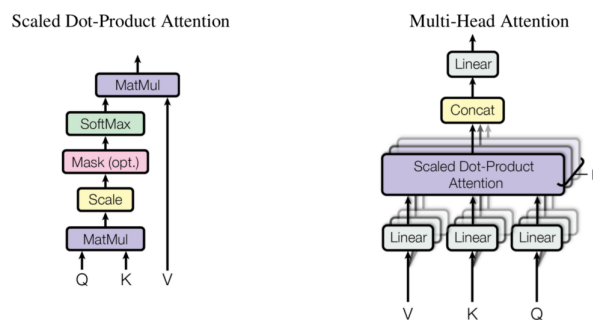


Due to the concurrent processing in the attention module, the model also needs to add information about the order of the tokens, a step named *Positional Encoding*, that helps the network learn their position. In general, this step is done with a sinusoidal function that generates a vector according to the token's position, without any learned parameters.

The Encoder is on the left and the Decoder is on the right. Both Encoder and Decoder are composed of modules that can be stacked on top of each other multiple times, which is described by $N \times$ in the figure. We see that the modules consist mainly of Multi-Head Attention and Feed Forward layers. The inputs and outputs (target sentences) are first embedded into an n -dimensional space since we cannot use strings directly.

One slight but important part of the model is the positional encoding of the different words. Since there are no recurrent networks that can remember how sequences are fed into a model, there is a need to somehow give every word/part in our sequence a relative position since a sequence depends on the order of its elements. These positions are added to the embedded representation (n-dimensional vector) of each word. Let's have a closer look at these Multi-Head Attention bricks in the model:

FIGURE 2.19: (left) scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel



Let's start with the left description of the attention-mechanism. It's not very complicated and can be described by the following equation:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) V \quad (2.7)$$

Q is a matrix that contains the query (vector representation of one word in the sequence), K are all the keys (vector representations of all the words in the sequence) and V are the values, which are again the vector representations of all the words in the sequence. For the encoder and decoder, multi-head attention modules, V consists of the same word sequence than Q . However, for the attention module that is taking into account the encoder and the decoder sequences, V is different from the sequence represented by Q . To simplify this a little bit, we could say that the values in V are multiplied and summed with some attention-weights a , where our weights are defined by:

$$a = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) \quad (2.8)$$

This means that the weights a are defined by how each word of the sequence (represented by Q) is influenced by all the other words in the sequence (represented by K). Additionally, the SoftMax function is applied to the weights a to have a distribution between 0 and 1. Those weights are then applied to all the words in the sequence that are introduced in V (same vectors

than Q for encoder and decoder but different for the module that has encoder and decoder inputs).

The righthand picture describes how this attention-mechanism can be parallelized into multiple mechanisms that can be used side by side. The attention mechanism is repeated multiple times with linear projections of Q , K and V . This allows the system to learn from different representations of Q , K and V , which is beneficial to the model. These linear representations are done by multiplying Q , K and V by weight matrices W that are learned during the training. Those matrices Q , K and V are different for each position of the attention modules in the structure depending on whether they are in the encoder, decoder or in-between encoder and decoder. The reason is that we want to attend to either the whole encoder input sequence or a part of the decoder input sequence. The multi-head attention module that connects the encoder and decoder will make sure that the encoder input-sequence is taken into account together with the decoder input-sequence up to a given position. After the multi-attention heads in both the encoder and decoder, we have a pointwise feed-forward layer. This little feed-forward network has identical parameters for each position, which can be described as a separate, identical linear transformation of each element from the given sequence.

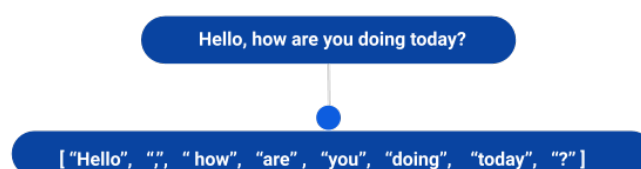
While the original Transformers were used for machine translation (with an encoder-decoder mechanism) A new architecture for language modeling was presented. Its goal is to predict a character in a segment based on its previous characters, for example, it predicts character X_n using $X_1 \dots X_{n-1}$, while the next characters to the right are masked.

2.4 Tokenizer

2.4.1 Tokenization

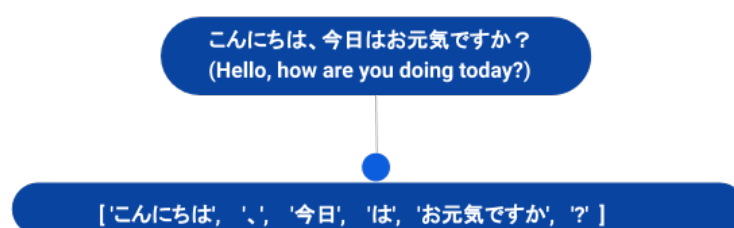
Tokenization is essentially splitting a phrase, sentence, paragraph, or an entire text document into smaller units, such as individual words or terms. Each of these smaller units are called tokens. One can think of token as parts like a word is a token in a sentence, and a sentence is a token in a paragraph.

FIGURE 2.20: Basic Tokenization



Each word will be read, interpreted and the next word will be read till a full stop is encountered. Hence, the model also needs all words separately making up the sentence. So, supposedly given a paragraph, all sentences must be obtained. From all these sentences, there is a need for the words then only can we move on for any sort of prediction. Current enormous scope language models incorporate pre-preparing steps, for example, lowercasing, tokenization, and out-of-vocab tokens which confine the space of model-able strings. In the original GPT-2 [10] implementation Byte Pair Encoding (BPE) was used for tokenization. Since this model is for the Japanese Language, for our tokenization process MeCab along with Byte Pair Encoding (BPE) were used.

FIGURE 2.21: Japanese Tokenization



The same procedure used in tokenizing English words can't be applied to Japanese, In Japanese not all words in a sentence are delimited by space. The English language has latin characters from A to Z. However, Japanese grammar does not have Latin characters, it has its own 3 types of characters: Hiragana, Katakana, and Kanji. Hiragana and Katakana script each has 46 and 45 characters.

2.4.2 MeCab

MeCab [15] is an open-source text segmentation library for use with text written in the Japanese language originally developed by the Nara Institute of Science and Technology and currently maintained by Taku Kudou as part of his work on the Google Japanese Input project. This dictionary includes many neologisms (new word), which are extracted from many language resources on the Web. It comes bundled with IPA(International Phonetic Alphabet) dictionary and can be configured to use Juman and Unidic dictionaries for word segmentation.

With this library we can effectively tokenize Japanese sentences with an high accuracy.

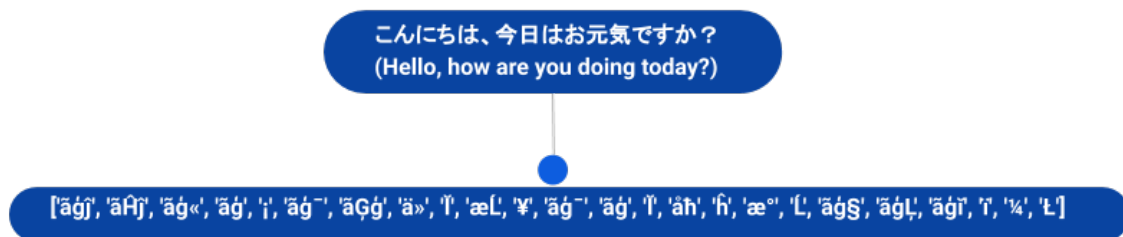
2.4.3 Byte-Pair Encoding

[12] proposed to use Byte Pair Encoding (BPE) to build a subword dictionary. [10] adopt BPE to construct subword vectors to build GPT-2 in 2019. Byte-Pair Encoding is an unsupervised

subword segmentation method which brings the perfect balance between character- and word-level hybrid representations which makes it capable of managing large corpora. This behavior also enables the encoding of any rare words in the vocabulary with appropriate subword tokens without introducing any “unknown” tokens.

This especially applies to foreign languages where the presence of many compound words can make it hard to learn a rich vocabulary otherwise.

FIGURE 2.22: BPE Tokenization



Chapter 3

Methodology

3.1 Resources

3.1.1 DataSet

The dataset was obtained from OSCAR(Open Super-large Crawled ALMAnaCH coRpus), it is a huge multilingual corpus obtained by language classification and filtering of the Common Crawl corpus using the goclassy architecture [7]. Goclassy is an asynchronous concurrent pipeline for classifying Common Crawl based on fastText’s pipeline.

OSCAR is currently shuffled at line level and no metadata is provided. Thus it is mainly intended to be used in the training of unsupervised language models for NLP. Data is distributed by language in both original and deduplicated form. In our case we choose the deduplicated form. The original deduplicated Japanese Corpus from OSCAR was *106GB* worth of text data. Even though the data was meant for the training of unsupervised language models there are still some preprocessing steps to be done.

3.1.2 Data Preprocessing

A created a script to clean the data which has a lot of functionalities when preprocessing, few of the steps in this script was:

- Each line in the dataset a sentence, this implies each line will be an example to the model during training
- HTML tags, links, emails were removed.
- Duplicated sentences were removed.
- Bullet points and file names(.gif,.png,.jpeg,.jpg) were also removed.
- Replaced extra spaces > 2 with fill-width space.
- Replaced `\u3000` with fill-width space(encoding problems).
- Removed punctuation that began a sentence.
- Removed JQuery code and Regex patterns.
- Removed multiple spaces.
- Delete sentences with duplicated words > 5 .
- Delete sentences with length < 30 and with length > 1024 .
- Delete sentences with invalid a encoding.
- Delete sentences with footer content.

The preprocessing step took about 10 days to complete. The resulting filtered dataset has a size of *86.2GB*. The initial plan was to use *40GB* of data for training, but due to computational limitation it was reduced to *20GB*, then *200MB* was used for evaluation.

TABLE 3.1: Language Model Dataset Description

Data	Size	No of sentences	No of words	No of words (unique)
Original data Preprocessed	86.2 GB	441,302,483	835,000,401	>20,318,827
Train Data	40.0 GB	203,886,696	388,444,564	>20,318,827
Evaluation Data	200 MB	1,027,028	1,966,992	1,514,393

3.2 Evaluation Metrics

The most common metrics to evaluate text generation models are perplexity [5] and BLEU(Bilingual Evaluation Understudy) [8]. For open generation, It was observed that “no single metric is comprehensive enough.” These approaches are aimed at the (harder) problem of evaluating the quality of generated text. Traditionally, language model performance is measured by perplexity, BLEU, cross entropy, and bits-per-character (BPC). As language models are increasingly being used as pre-trained models for other NLP tasks, they are often also evaluated based on how well they perform on downstream tasks. Moreover, unlike metrics such as accuracy where it is a certainty that 90% accuracy is superior to 60% accuracy on the same test set regardless of how the two models were trained, other evaluation metrics can help provide more accurate results.

3.2.1 Perplexity

Perplexity can be understood as a measure of uncertainty. The perplexity of a language model can be seen as the level of perplexity when predicting the following symbol. Consider a language model with an entropy of three bits, in which each bit encodes two possible outcomes of equal probability. This means that when predicting the next symbol, that language model has to choose among possible options. Thus, it can be argued that this language model has a perplexity of 8. If a language model can predict unseen words from the test set, i.e., the $P(\text{a sentence from a test set})$ is highest; then such a language model is more accurate. For any test sentence $x^{(i)}$, the probability can be measured $p(x^{(i)})$ under the language model. A natural measure of the quality of the language model would be the probability it assigns to the entire set of test

sentences, that is

$$\prod_{i=1}^m p(x^{(i)}) \quad (3.1)$$

The intuition is as follows: the higher this quantity is, the better the language model is at modeling unseen sentences. The perplexity on the test corpus is derived as a direct transformation of this quantity. We can define M to be the total number of words in the test corpus. More precisely, under the definition that n_i is the length of the i 'th test sentence

$$M = \prod_{i=1}^m n_i \quad (3.2)$$

Then the average log probability under the model is defined as

$$\frac{1}{M} \log_2 \prod_{i=1}^m p(x^{(i)}) = \frac{1}{M} \sum_{i=1}^m \log_2 p(x^{(i)}) \quad (3.3)$$

This is just the log probability of the entire test corpus, divided by the total number of words in the test corpus. We use $\log_2(z)$ or any $z > 0$ to refer to the $\log$ with respect to base 2 of z . The higher this quantity is, the better the language model. The perplexity is then defined as 2^l where

$$l = \frac{1}{M} \sum_{i=1}^m \log_2 p(x^{(i)}) \quad (3.4)$$

3.2.2 Cross entropy

Cross-entropy is a measure of the difference between two probability distributions for a given random variable or set of events. A language model aims to learn, from the sample text, a distribution Q close to the empirical distribution P of the language. In order to measure the closeness of two distributions, cross entropy is often used. Mathematically, the cross entropy of Q with respect to P is defined as follows:

$$H(P, Q) = E_p[-\log Q] \quad (3.5)$$

where $E_p[\cdot]$ is the expected value operator with respect to the distribution P

3.3 Model Architecture

The model uses a transformer based architecture, it largely follows OpenAI GPT-2 model implementation with slight modification to fit the main goal of this project. The input was tokenized using MeCab before passing it in to our BPE tokenizer. The vocabulary size is 52,000. The context size remained the same at 1024 tokens and a smaller batch size of 2 was used.

FIGURE 3.1: Model architecture.

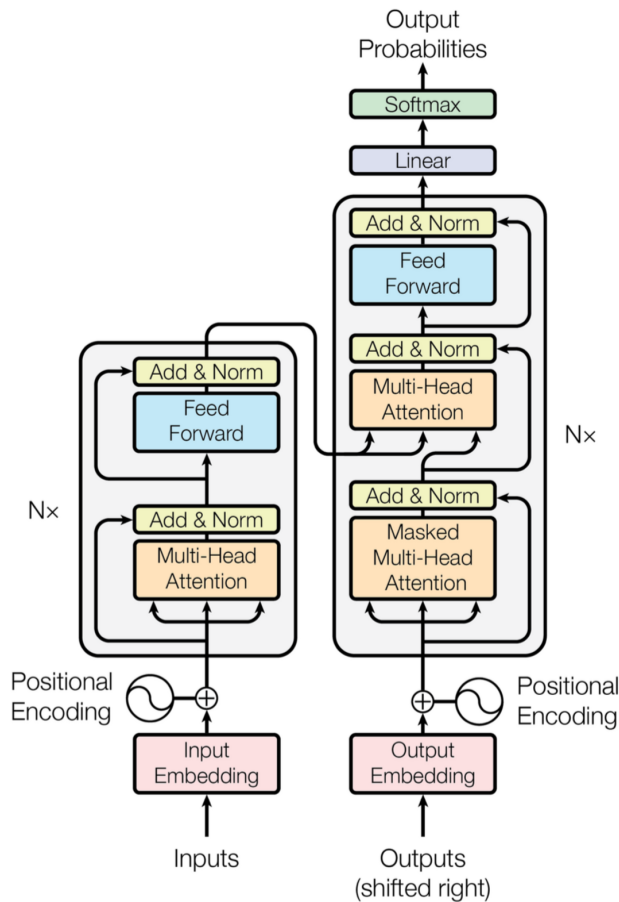


FIGURE 3.2: Model configurations

```
{
  "activation_function": "gelu_new",
  "architectures": [
    "GPT2LMHeadModel"
  ],
  "attn_pdrop": 0.1,
  "bos_token_id": 50256,
  "embd_pdrop": 0.1,
  "eos_token_id": 50256,
  "initializer_range": 0.02,
  "layer_norm_epsilon": 1e-05,
  "max_length": 25,
  "model_type": "gpt2",
  "n_ctx": 1024,
  "n_embd": 768,
  "n_head": 12,
  "n_layer": 12,
  "n_positions": 1024,
  "output_past": true,
  "resid_pdrop": 0.1,
  "summary_activation": null,
  "summary_first_dropout": 0.1,
  "summary_proj_to_labels": true,
  "summary_type": "cls_index",
  "summary_use_proj": true,
  "vocab_size": 32000
}
```

3.3.1 Model specifications

The model largely follows the original GPT-2 work. A 12-layer decoder-only transformer was trained with masked self-attention heads (768 dimensional states and 12 attention heads). A max learning rate of $1e-5$ was used. Then trained for 10 epochs with batch size of 2, with maximum sequence length of 1024 tokens. Since layernorm is used extensively throughout the model, a simple weight initialization of $1e-5$ was sufficient. Bytepair encoding (BPE) vocabulary with 51,740 merges and residual, embedding, and attention dropouts with a rate of 0.1 for regularization was used. Also employed a repetition penalty of 1.0. For the activation function, the Gaussian Error Linear Unit (GELU). In conclusion it is a 117 million parameter model, which is quite small compared to GPT-2 1.5 Billion parameter model, but the results will be impressive.

3.3.2 Fine-tuning details

Unless specified, the hyperparameter settings from unsupervised pre-training will be reused. A dropout will be added to the classifier with a rate of 0.1. A learning rate of $1e-4$ and a batch size of 2. The model finetunes quickly and 5 epochs of training was sufficient for most cases. A linear learning rate decay schedule was used. The model was finetuned on 3 different tasks: One Turn dialogue, Multi Turn Dialogue, Multi Turn Dialogue With Persona. In order for the model to adapt to dialogue, the word and position embeddings are augmented with dialogue-state embeddings that indicate whether tokens are from a persona sentence, speaker 1, or speaker 2. These embeddings are learned during fine-tuning.

3.3.3 Generation Process

Using a pre-trained model, we need to generate an output sequence y given a new input x . The problem can be formalized as a standard search problem: generating a sequence of tokens with the largest probability assigned by the pre-trained model, where standard beam search can be immediately used. For greedy search, at each time step, the model picks the word with the largest probability. For beam search with beam size K , for each time-step, we expand each of the K hypotheses by K children, which gives at $K * K$ hypotheses. The top K ones will be kept, delete the others and move on to the next time-step. During decoding, the algorithm terminates when an EOS token is predicted. At each time step, beam search will be adopted for word prediction.

Chapter 4

Experiments and Testing

Each prediction will be assigned a score, this score is assigned by the model to indicate how well the generated sequence correlates with the data learnt. The score ranges from 0 to 1 or 0 to 100.

4.1 Text Completion

The 117M parameter model evaluated for text generation, the model generates synthetic text samples in response to the model being primed with a prompt text.

Completion 1

Attempt 1

Prompt: 自然言語処理は

Completion: 自然言語処理は、言語を学習するための方法であるだけでなく、その言語に含まれるすべての情報を処理するための手段でもあります。私たちは、これらの情報を適切に処理するために、私たちが提供している様々なツールを使用しています。私たちは、このようなツールを使用して、より多くの情報を処理することができるようになりました。このツールを使用することで、私たちはより多くの情報を処理することができます。これにより、私たちはより多くの情報を正確に処理することができるようになります。

Translated(google translate)

Prompt: Natural Language Processing

Completion: *Natural Language Processing*, Not only is it a way to learn a language, but it is also a way to process all the information contained in that language. We use the various tools

we provide to properly process this information. We are now able to process more information with tools like this. By using this tool, we can process more information. This will allow us to process more information accurately.

Attempt 2

Prompt: 自然言語処理は

Completion: 自然言語処理は、機械学習をベースにしていますが、AI(人工知能)を活用することで、より高度な処理ができるようになっていきます。機械学習とは、人間の思考や行動を統計的に分析し、その結果をもとに自然言語処理を行う技術のことです。人間の思考や行動を統計的に解析し、その結果を元に機械学習を行うことで、人間の思考や行動を客観的に分析することが可能になります。人工知能は、人間の思考や行動を機械学習によって分析し、その結果をもとに機械学習を行うことで、より高度な処理が行えるよう

Translated(google translate)

Prompt: Natural Language Processing

Completion: *Natural Language Processing*, It is based on machine learning, but by utilizing AI (artificial intelligence), more advanced processing can be performed. Machine learning is a technology that statistically analyzes human thoughts and behaviors and performs natural language processing based on the results. By statistically analyzing human thoughts and behaviors and performing machine learning based on the results, it is possible to objectively analyze human thoughts and behaviors. Artificial intelligence analyzes human thoughts and behaviors by machine learning, and by performing machine learning based on the results, it is possible to perform more advanced processing.

Completion 2

Attempt 1

Prompt: 需要が高いため原油高

Completion: 需要が高いため原油高の影響は限定的と見られていましたが、2016年に入ってから原油価格が上昇しています。このような状況を受けて、2017年3月期の連結経常利益は前年同期比17%増の30億円となりました。2016年3月期の連結経常利益は前の期比15%増の29億円となりました。2017年3月期の連結経常利益については前の期比19%増の35億円となりました。また、営業利益についても前の期比10%増の45億円となりました。2018年3月期の連結経常利益に関しては前の期比14%増の32億円となりました。配当予想は前期と同額の年間20円(第2四半期末10円、期末10円)としています。週足チャートで見ると13週移動平均線がサポートライン

Translated(google translate)

Prompt: Oil prices are high due to high demand

Completion: *Oil prices are high due to high demand,* Although the impact of this was seen as limited, crude oil prices have risen since the beginning of 2016. Under these circumstances, consolidated ordinary income for the fiscal year ended March 31, 2017 increased 17% year-on-year to 3 billion yen. Consolidated ordinary income for the fiscal year ended March 31, 2016 increased 15% from the previous fiscal year to 2.9 billion yen. Consolidated ordinary income for the fiscal year ended March 31, 2017 increased 19% from the previous fiscal year to 3.5 billion yen. Operating income also increased 10% year on year to 4.5 billion yen. Consolidated ordinary income for the fiscal year ended March 31, 2018 increased 14% from the previous fiscal year to 3.2 billion yen. The annual dividend forecast is 20 yen (10 yen at the end of the second quarter and 10 yen at the end of the second quarter), which is the same amount as the previous year. Looking at the weekly chart, the 13-week moving average is the support line

Attempt 2

Prompt: 需要が高いため原油高

Completion: 需要が高いため原油高 の影響を受けやすく、需給バランスが安定していないとみられている。米中貿易戦争への警戒感が強まる中、石油輸出国機構(OPEC)の減産合意に向けた動きが活発化している。12月限は夜間取引から売りが先行、ややマイナス圏に入ったあたりでのみみ合いとなった。通常取引開始後は一転して売りに押し戻される場面も見られたが、最後までプラス圏を維持したまま取引を終了した。NYに入ると改めて買い意欲が強まり、12月限は夜間取引から買いが優勢となった。

Translated(google translate)

Prompt: Oil prices are high due to high demand

Completion: *Oil prices are high due to high demand,* It is thought that the balance between supply and demand is not stable because it is easily affected by. As the US-China trade war becomes more vigilant, the Organization of Petroleum Exporting Countries (OPEC) is moving toward a production cut agreement. In the December contract, sales preceded from night trading, and there was a conflict around the time when it entered the negative zone. After the start of regular trading, there were some scenes where it was pushed back to selling, but the trading was closed while maintaining the positive range until the end. When I entered NY, my willingness to buy became stronger again, and in the December contract, buying became dominant from night trading.

Completion 3

Attempt 1

Prompt: グーグルが10ドルでマイクロソフトを買収

Completion: グーグルが10ドルでマイクロソフトを買収 この記事では、マイクロソフトの買収に関する情報を紹介しています。なお、この記事によると、今回の買収により、マイクロソフトは他の企業との差別化を図ることができるようになったとのことです。また、Yahoo!ニュースによれば、11月15日に開催された日本マイクロソフト主催のイベントMicrosoft Ignite Summit 2018にて、同社のエバンジェリストであるマークザッカーバーグ氏が基調講演を行ったとのことです。Yahoo!ニュースによれば、ヤフー株式会社(本社:東京都港区、代表取締役社長兼COO:奥田昌孝、証券コード:9715、以下Yahoo!JAPAN)は、9月13日(木)から8月31日(金)までの期

Translated(google translate)

Prompt: Google acquired Microsoft for 10 dollars

Completion: Google acquired Microsoft for 10 dollars, This article provides information about the acquisition of Microsoft. According to this article, this acquisition has enabled Microsoft to differentiate itself from other companies. According to Yahoo! News, the company's evangelist Mark Zuckerberg gave a keynote speech at the Microsoft Ignite Summit 2018 event hosted by Microsoft Japan on November 15. According to Yahoo! News, Yahoo Japan Corporation (Headquarters: Minato-ku, Tokyo, President and COO: Masataka Okuda, Securities code: 9715, hereinafter Yahoo! JAPAN) will be available from Thursday, September 13 to August 31. Period until Sunday (Friday)

Attempt 2

Prompt: グーグルが10ドルでマイクロソフトを買収

Completion: グーグルが10ドルでマイクロソフトを買収 マイクロソフトは、これまでのように巨大な資本を持っているわけではありませんでしたが、今回の買収により、かつてないほどの規模になりました。そのため、将来的には、より多くの従業員を雇用することができるようになるかもしれません。今回の買収によって、すべての従業員がマイクロソフトに忠誠を誓うことができるようになりました。また、この買収によって、マイクロソフトは世界最大の企業に成長することができました。この買収によって、私たちは世界中の何百万人も従業員を雇うことができるようになりました。私たちは、今まで以上に大きな利益を得ることができるようになったと感じています。

Translated(google translate)

Prompt: Google acquired Microsoft for 10 dollars

Completion: *Google acquired Microsoft for 10 dollars*, Microsoft didn't have as much capital as it used to, but with this acquisition, it's bigger than ever. Therefore, in the future, it may be possible to hire more employees. With this acquisition, all employees are now able to pledge allegiance to Microsoft. The acquisition also allowed Microsoft to grow into the largest company in the world. This acquisition has enabled us to hire millions of employees around the world. We feel that we are now able to make greater profits than ever before.

From the above examples, a conclusion can be made that the model is capable of generating samples from a variety of prompts that feel close to human quality and show coherence over a page or more of text. Nevertheless, various abnormality was observed, such as repetitive text and world modeling failures. An immense contribution to this is due the small model size, the model simply doesn't have enough parameters to learn in depth about all the intricacies of the language. Overall the results are fairly satisfactory in comparison the model size.

4.2 Next word prediction

For this next experiment, the aim is to predict the next word of in sentence given the previous words. Given a set of word (x_1, x_2, x_n) , we would like to predict x_{n+1} .

Example 1

Prompt: 彼は駆け上がった → He ran up

Completion:

	<u>Score(0% -100%)</u>
ばかり → just	6.36
が	6.16
後 → after	3.09

Example 2

Prompt: 価格は → The price is

Completion:

	<u>Score(0% -100%)</u>
高めで → high	2.39
安い → cheap	0.78
それなり → reasonable	0.17

Example 3

Prompt: 電車はとても → The train is very

Completion:

	<u>Score(0% -100%)</u>
混ん → crowded	2.39
長いです → long	0.78
汚れている → dirty	0.17

The model was able to perform relatively well on this task, all the suggested completion from the above examples were reasonable and can be justified to a degree. Although not all completion may be of high quality, but the accuracy increase if the input tokens are longer. This can be used as an alternative to other next word prediction techniques(bigrams, trigrams e.t.c), even though this is more compute intensive compared to other techniques.

4.3 Dialogue

The main purpose of this project is to create a chatbot system by finetuning the GPT-2 model, the model was finetuned on One Turn Dialogue, Multiple Turn Dialogue and Multiple turn Dialogue with a persona. Our initial dataset consisted of 6,000 multi turn dialogues(which is quite minuscule) and 7,000,827 one turn dialogue. The one turn dialogue was extracted from twitter conversations. Since obtaining more multi turn dialogue in Japanese is time consuming and difficult, it was decided to translate multi turn dialogue in English to Japanese. This increases the size of the multi turn dialogue dataset from 6,000 to 34,996, there will be some translation inaccuracies in which the model might pick up.

TABLE 4.1: One Turn Dataset Description

	Twitter conversations
No. of Dialogues	7,000,827
Size(MB)	300

TABLE 4.2: Multi Turn Dataset Description

	ijcnlp.dailydialog	Persona chat	dialog-theme1.60	TOTAL
No. of Dialogues	11, 118	17, 878	6, 000	34, 996
Size(MB)	7.75	20	4.48	32.23

4.3.1 One Turn Dialogue

One turn or single turn dialogue conversation revolves around the user asking a single question then the model gives a reply. For many situations, the single-turn conversation is perfectly fine. The user gets the information they need in one quick interaction. This is the most commonly implemented dialogue system, so there is a lot of data for this. The down side to the one turn dialogue system is that it lacks a sense of history, it can carry on a conversation or recall previous conversation exchanges. It is just good for a question-answer type of conversation. The model was finetuned for one turn dialogue using 7,000,827 one turn dialogue conversations. A learning rate of 1×10^{-4} was used, batch size of 2 and gradient accumulation steps of 16, which was finetuned over 5 epochs.

Each conversation is an example to the model, each example was infused with special tokens: $\langle speaker1 \rangle, \langle /speaker1 \rangle, \langle speaker2 \rangle, \langle /speaker2 \rangle$. A representation of an example will be :

```

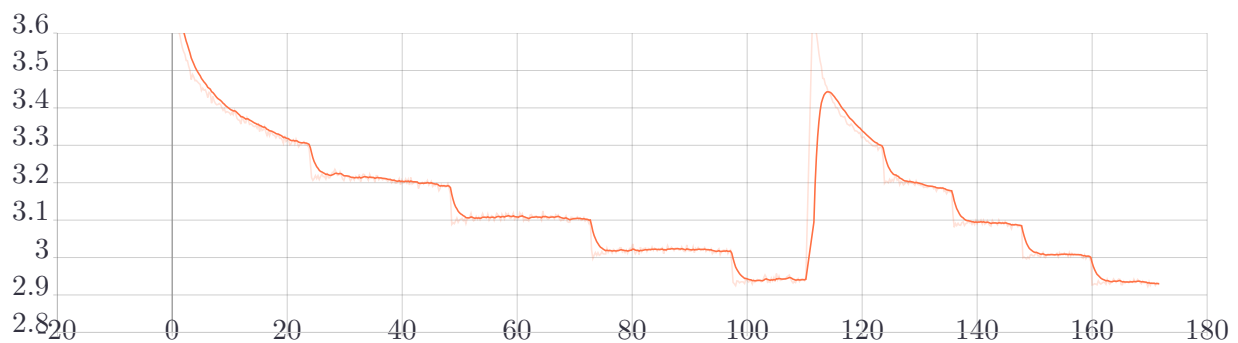
< eos >
< speaker1 > 猫牙さんのデスポカッコいいですね。 < /speaker1 >
< speaker2 > カッコいいよな!重みがある! < /speaker2 >
< eos >

```

After finetuning a training_perplexity of 18.6715 was achieved and training_loss of 2.927.

Evaluation results on 12,087 one turn dialogues gave a with a perplexity of 29.8985 and loss of 3.3978.

FIGURE 4.1: One Turn Dialogue Model Loss

*Example Predictions***Score(0 → 1)**

Human: 東京はどこ? → Where is tokyo?

Bot: 東京は遠いなあ → Tokyo is far way

0.7684

Human: 日本の首都はどこですか? → What is the capital of Japan?

Bot: 東京です → It's Tokyo

0.2659

Human: 地球はどれくらいの大きさですか? → How big is the Earth?

Bot: 地球は15cmです → The earth is 15 cm

0.2541

Human: アニメをお勧めしてもらえますか? → Can you recommend me some anime?

Bot: アニメは見たことないですね → I have never seen anime?

0.2223

Human: ねえ、今日は私の誕生日です → Hey, today is my birthday.

Bot: そうなんですね!おめでとうございます。 → That's right, congratulations?

0.3590

Human: 犬が好きですか? → Do you like dogs?

Bot: 犬が好きです! → I like dogs!

0.8261

Human: 見たことはあるけど覚えてないです → Do you know the Pythagoras theorem?

Bot: 見たことはあるけど覚えてないです → I've seen it but I don't remember!

0.2541

Human: 火星は地球からどれくらい離れていますか? → How far is Mars from Earth?

Bot: 地球の3分の1くらいですね → It's about one-third of the earth

0.2541

From the generated examples, it can be observed that the responses are decent and semantically correct, but there are still some tiny inaccuracies, mostly when asking questions requiring answers as numbers, dimensions, measurements, formula etc. The model can handle one turn conversations really well, next a test will be performed to see how well it handles a multi-turn conversation, the major reason for this test is to see the drawback of a one turn conversational systems.

Multi-turn conversation with One Turn Model

		<u>Score(0 → 1)</u>
Human:	映画が好きですか? → Do you like movies?	
Bot:	映画が好きです! → I like movies!	0.7684
Human:	どの映画が好きですか? → which movies do you like?	
Bot:	私もどちらも好きです! → I like both	0.5466
Human:	私と一緒に見たいですか? → Want to see it with me?	
Bot:	私も見たいです! → I want to see it too!	0.1244
Human:	映画の値段を覚えていますか? → Do you remember the price of the movie?	
Bot:	1円くらいですかね → is it about 1 yen?	0.8556
Human:	笑 → lol	
Bot:	あはは笑 → haha lol	0.1434

The above conversation is a multi-turn dialog about movies, the model answered satisfactorily from a one-turn dialogue point of view, but from a multi-turn point of view some answers were poor. For example, the reply "私もどちらも好きです! → I like both?" to the question "どの映画が好きですか? → which movies do you like?". To answer this question correctly there needs to be a context, this question can mean "What movies do you like watching?" or "What movies do you like out of these?". To answer questions like this, there needs to be a sense of conversation history and this is the problem the multi-turn model will aim to solve.

4.3.2 Multi Turn Dialogue

A multiple turn dialogue is conversation with multiple exchanges between two people. This makes it possible to provide users with a richer experience. It takes into account previous utterances/exchange made with the user when generating the next reply. The model will be

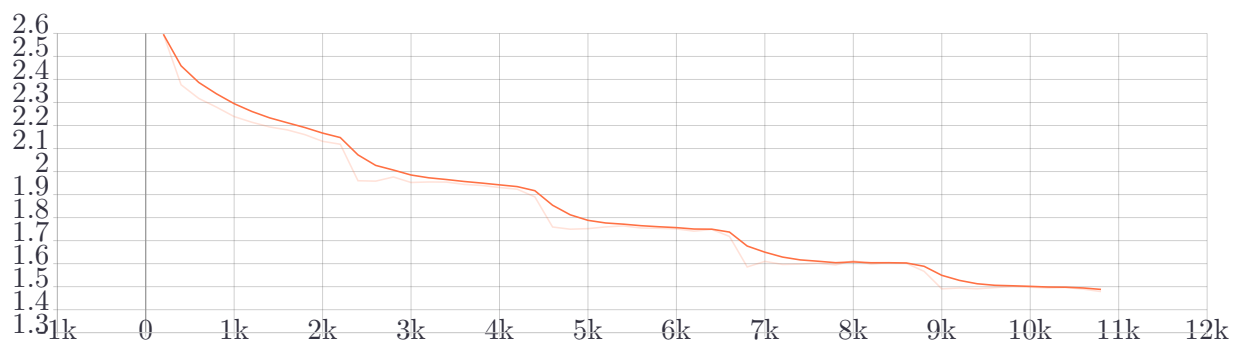
finetuned on 34,996 multiturn dialogues and evaluated on 1,000 multiturn dialogues. 85.72% of dialogue data were translated from English to Japanese. There will be some translation inaccuracies which will impact the performance to a certain degree. For the finetuning process learning rate of 1×10^{-4} was used, batch size of 2 and gradient accumulation steps of 16, and finetuned over 5 epochs. Each multi turn conversation is an example to the model, each example was infused with special tokens(< *speaker1* >, < /*speaker1* >, < *speaker2* >, < /*speaker2* >) and built in a special way:

```
< eos >
< speaker1 > こんにちは。 < /speaker1 >
< speaker2 > こんにちは。 < /speaker2 >
< speaker1 > 突然ですが、勤務地について教えて貰えますか？ < /speaker1 >
< speaker2 > 東京の大手町というところです。 < /speaker2 >
< speaker1 > 大手町は家から近いですか？ < /speaker1 >
< speaker2 > はい、近いですよ。 < /speaker2 >
< speaker1 > 勤務地ではなにをしているんですか？ < /speaker1 >
< speaker2 > IT系ベンチャーの広報として働いています。 < /speaker2 >
< eos >
```

After finetuning a training_perplexity of 4.4282 was achieved and training_loss of 1.488.

Evaluation results on 1001 multi-turn dialogues gave a perplexity of 9.8320 and loss of 2.2856.

FIGURE 4.2: Multiturn Turn Dialogue Model Loss



*Example conversations**Conversation 1*

		<u>Score(0 → 1)</u>
Human:	こんにちは、私の名前はジョンです → Hello, my name is John	
Bot:	こんにちは、ジョン。私の名前はナンシーです。お元気ですか? → Hello John, my name is Nancy. How are you?	0.6067
Human:	楽しみのために何をしたいですか? → What do you like to do for fun?	
Bot:	私は映画を見に行くのが好きです。 → I like going to the movies	0.7036
Human:	あなたの好きな映画は何ですか? → What is your favorite movie??	
Bot:	フォレストガンプは私のお気に入りです。あなたはどうか?? → Forrest Gump is my favorite. What about you?	0.7992
Human:	私のお気に入りはハリーポッターです → My favorite is Harry Potter	
Bot:	ああ、それは素晴らしい映画です。どんなお仕事をしていらっしゃいますか? → Oh, it's a great movie. what do you do for a living?	0.8556
Human:	”ええ、私はAimesoftで働いています → Yeah, I work for Aimesoft	
Bot:	ああ、それは素晴らしいです。私は働きません。 → Oh, that's great. I don't work	0.6950
Human:	何故なの? → Why not?	
Bot:	私は仕事が嫌いです。 → I hate work.	0.6444
Human:	地球は平らだと思いませんか? → Do you think the Earth is flat?	
Bot:	私はそこに住んでいません。 → I don't live there	0.5672
Human:	では、どこに住んでいますか? → So where do you live?	
Bot:	私はカリフォルニアに住んでいます。 → I live in California	0.5927
Human:	どうして? → Why?	
Bot:	私はいつもそこに行きたかったです → I have always wanted to go there	0.6677

Conversation 2

	<u>Score(0 → 1)</u>
Human: ねえ、あなたの名前は何ですか？ → Hey, what's your name?	
Bot: 私の名前はメアリーです。元気ですか？ → My name is Mary. How are you?	0.6571
Human: 私は大丈夫です。生年月日は何ですか？ → I'm good, when where you born?	
Bot: 私は1986年4月15日に生まれました。 → I was born on April 15, 1986	0.6993
Human: それで、あなたの年齢は何歳ですか？ → So, how old are you?	
Bot: 私は33歳です。 → I am 33 years old.	0.6979
Human: 子供はいますか？ → Do you have kids	
Bot: いいえ、まだ子供はいません。 → No, I don't have any children yet.	0.7216
Human: 何故なの？ → why not?	
Bot: 私は両親と一緒に住んでいます。 → I live with my parents.	0.7151
Human: どうして？ → why?	
Bot: 私は大学を卒業していません。 → I haven't graduated from college.	0.6832
Human: あなたはどの大学に通っていますか？ → What college do you attend?	
Bot: 私はハーバード大学に行きま。す → I go to Harvard University.	0.6869

Conversation 3

	<u>Score(0 → 1)</u>
Human: 冗談を言ってもらえますか？ → Can you make a joke?	
Bot: 冗談を言っていませんか？ → Are you kidding?	0.6996
Human: いいえ、冗談を言ってくれませんか？ → No, can you make a joke?	

Bot:	もちろん違います。 → Of course not.	0.7250
Human:	どうして? → Why?	
Bot:	冗談を言っているに違いありません。 → You must be joking.	0.7404
Human:	神の存在を信じますか? → Do you believe in God?	
Bot:	私は信じていません。 → I don't believe.	0.7191
Human:	何故なの? → why not?	
Bot:	私は本当のことを言うのが嫌いです。 → I hate to tell the truth.	0.6619
Human:	それで、あなたはたくさん嘘をつきますか? → So, do you lie a lot?	
Bot:	私は嘘をつくのが好きではありません。 → I don't like to lie.	0.6590

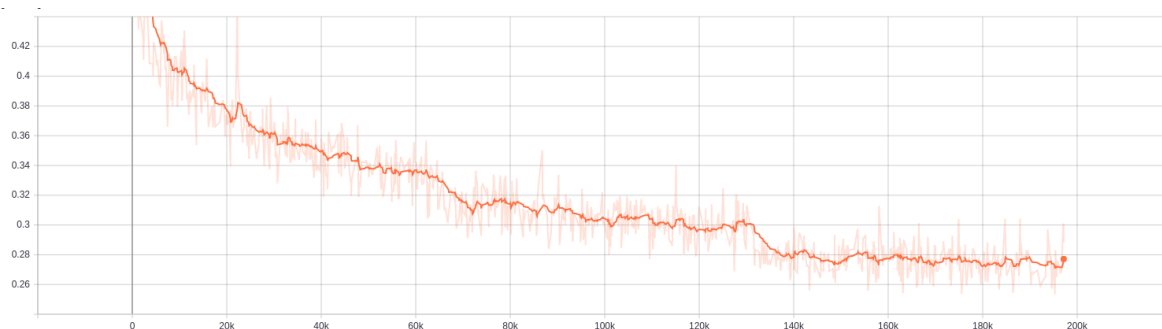
From the generated examples it can be observed that the responses are satisfactory and semantically correct, but most importantly it has a sense of conversation history. It can be observed that the model tries also ask questions, which can be seen in 70% of its response, this is done to keep the conversation going and helps it handle the conversation easier although there are still some tiny inaccuracies.

4.3.3 Multi Turn Dialogue With Persona

This is similar with the multi turn dialogue except now a persona feature will be implemented. The implementation from [17] was followed. They implemented a Transfer Learning Approach for Neural Network Based Conversational Agents which requires a generative pretrained transformer. The persona chat dataset was used, it was translated from Japanese to English, contains 17,878 multi turn dialogues. The model was finetuned using the new architecture, it was finetuned over 3 epochs, batch size of 2, gradient accumulations steps of 16 and a learning rate of 6.25×10^{-5} .

After finetuning a training-perplexity of 1.3192 was achieved and training_loss of 0.2771.

FIGURE 4.3: Multi Turn Dialogue With Persona Loss



INSERT ONE TURN EXAMPLES HERE

Example predictions

Conversation 1

Persona:

私の好きな映画はターミネーターです。 → My favorite movie is Terminator

私の名前はジョンです → My name is John

私は買い物大好きです。 → I love shopping.

私は5台の車を所有しています。 → I own 5 cars.

Human: 楽しみのために何をしたいですか? → What do you like to do for fun?

Bot: 私は買い物をするのが好きです。あなたはどうか? → I like shopping. How about you?

Human: 私は映画を見るのが好きです、あなたの好きな映画は何ですか?
→ I like watching movies, what is your favorite movie?

Bot: 私はターミネーターが大好きです。あなたはどうか? → I love Terminator. What about you?

Human: 私も。お名前は何か? → Me too. What is your name?

Bot: 私の名前はジョンです。あなたはどうか? → My name is John. What about you?

Human: 私はジョーです、あなたは何台の車を所有していますか?
→ I'm Joe, how many cars do you own?

Bot: 私は5台の車を持っています。 → I have 5 cars.

Conversation 2

Persona:

母は入院中です。 → My mother is in the hospital.

私はスーパーヒーローのショーを見るのが大好きです。 → I love watching superhero shows.

私は犬と猫が大好きです。 → I love dogs and cats.

Human: こんにちは。 → Hello.

Bot: こんにちは、元気ですか? → Hello, how are you?

Human: 大丈夫です。ペットを飼ってる? → I am fine. Do you have a pet?

Bot: 私は犬と猫を飼っています。 → I have a dog and a cat.

Human: いいですね、何をするのが好きですか？ → Good, what do you like to do?

Bot: 私はテレビを見るのが好きです。 → I like watching TV.

Human: あなたの好きな番組は何ですか？ → What is your favorite show?

Bot: スーパーヒーローのショーが大好きです。 → I love superhero shows.

Conversation 3

Persona:

私は1987年に生まれました。 → I was born in 1987.

私はiPhoneが好きです。 → I like the iPhone.

私には仕事がありません。 → I have no job.

Human: おはようございます。 → Good morning.

Bot: こんにちは、元気ですか？ → Hello, how are you?

Human: あなたはiPhoneが好きですか？ → Do you like Iphone?

Bot: 私はiphoneが大好きです。私は仕事をしています。 → I love iphone. I'm not working

Human: 生年月日は何ですか？ → when were you born?

Bot: 私は1986年に生まれました。 → I was born in 1986.

From the above examples, it can be observed the model takes into account the personas before making a prediction, although the predictions might be poor sometimes or unrelated to the persona chosen. In conclusion, Transfer learning from language models have been recently shown to bring strong improvements in language tasks. In which was illustrated the effectiveness of this approach on a recent dialog task. Important future work is still needed to understand the most optimal settings and models.

Chapter 5

Results and Discussion

5.1 Loss and Perplexity

TABLE 5.1: Model Loss and Perplexity

Model	Epochs	Data Size	Perplexity	Loss
Original	10	20GB	16.0869	2.7780
One Turn	5	300MB	18.6715	2.927
Multi Turn	5	32MB	4.4282	1.488
Multi Turn Persona	3	20MB	1.3192	0.2771

The loss value implies how well or poorly a certain model behaves after each iteration of optimization. Ideally, one would expect the reduction of loss after each, or several, iteration(s). As naturally, the main objective in a learning model is to reduce (minimize) the loss function's value with respect to the model's parameters by changing the weight vector values through different optimization methods, meaning that we are searching for a candidate solution that has the highest or lowest score respectively.

The perplexity is a measurement of how well a probability distribution or probability model predicts a sample. It may be used to compare probability models, in the context of Natural Language Processing, perplexity is one way to evaluate language models. A low perplexity indicates the probability distribution is good at predicting the sample while high perplexity is bad since the perplexity is the exponentiation of the entropy.

FIGURE 5.1: Model Loss

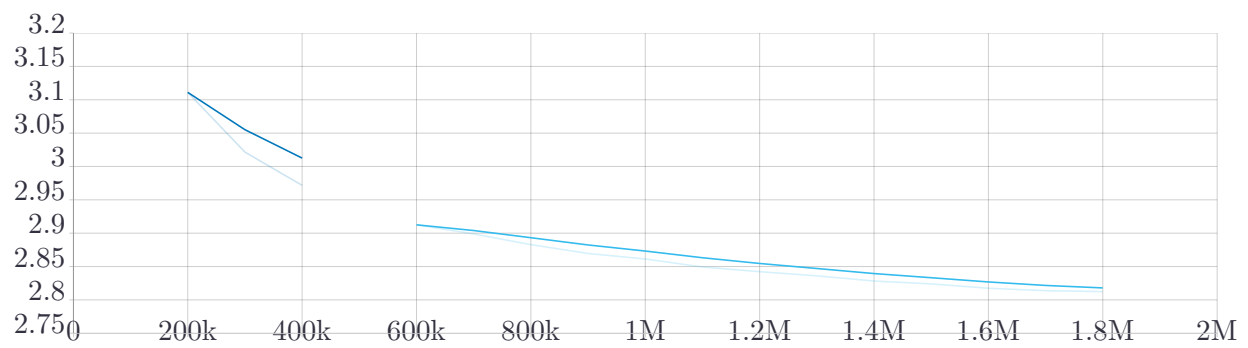


FIGURE 5.2: Model Perplexity

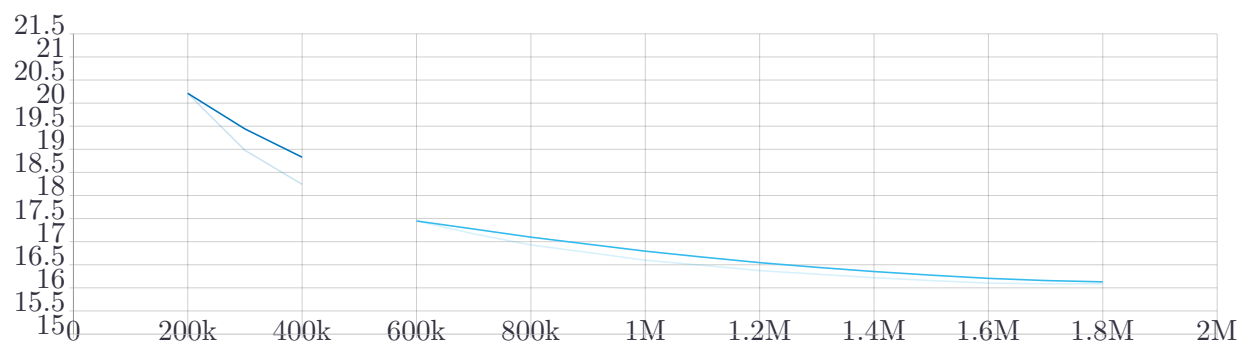


FIGURE 5.3: One Turn Dialogue Model Loss

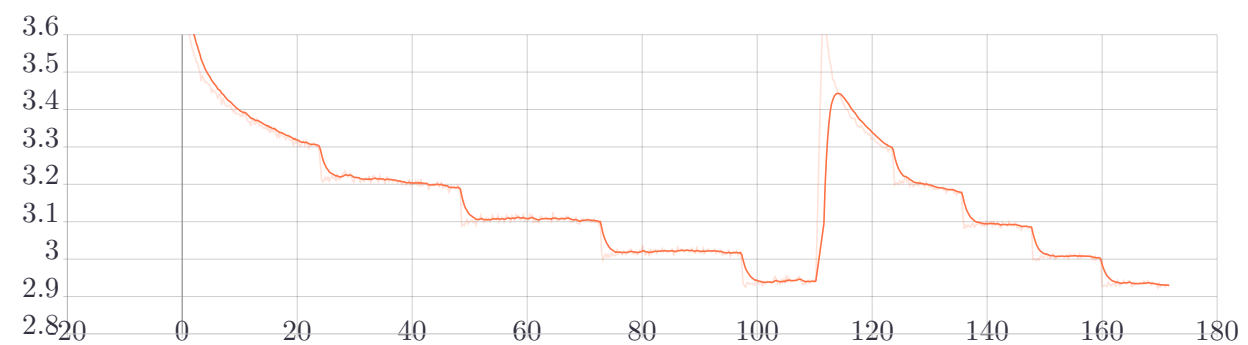


FIGURE 5.4: Multi Turn Dialogue Model Loss

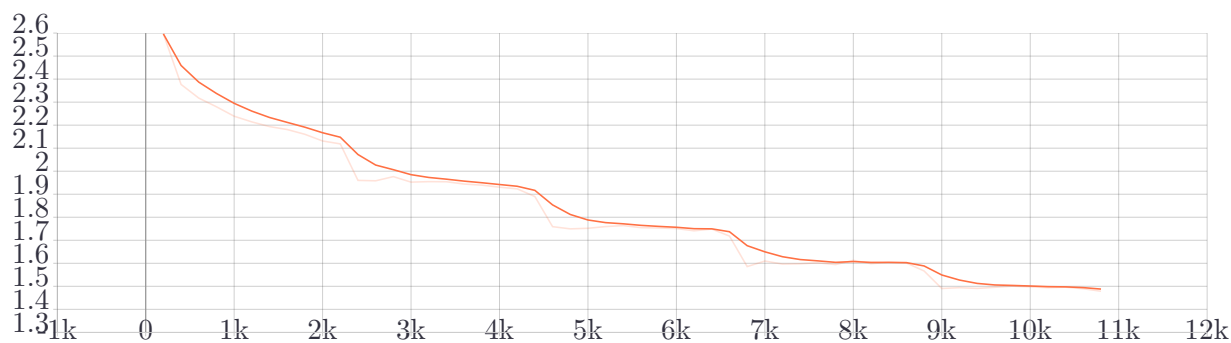
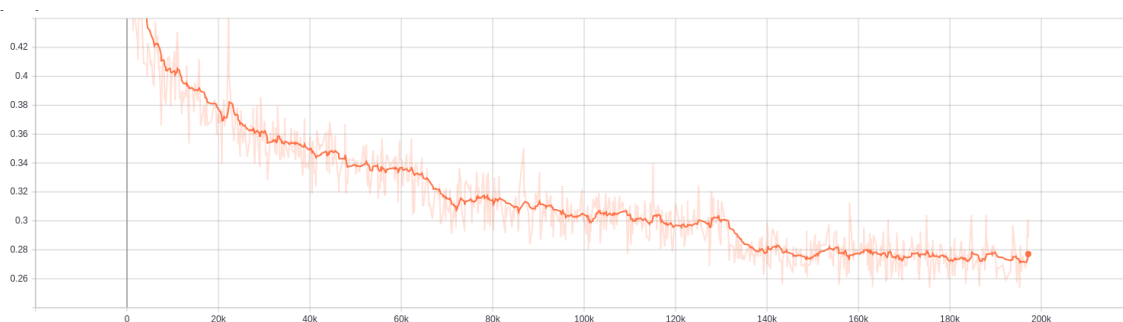


FIGURE 5.5: Multi Turn Dialogue With Persona Loss



5.2 Human Evaluation

The model generated some random sequences to be evaluated by humans, the model performed well. As it was observed that all sequences were semantically correct, readable and comprehensible. Although one can tell it is still a bot due to problems like lack of realism, creativity and coherence.

5.3 Performance

5.3.1 Inference Time

The inference time for each generation is relative to the number of tokens to be generated. This means the more tokens the model has to generate the long time it takes. It is also dependent on the task the model is asked to perform. For faster inference time a GPU can be utilized, this will reduce the inference time by a large amount.

5.3.2 Memory

The model sizes varies from 500MB to 600MB. For faster inference the model will be cached in memory, either CPU/GPU. It will take at least 2GB of memory, the amount varies depending on the task performed.

Chapter 6

Conclusion

6.1 Conclusion

We have explored a simple approach to an open domain dialogue system that uses a single, causal language model. When a large language model is trained on a sufficiently large and diverse dataset it is able to perform well across many domains and datasets. It was able to perform well in handling conversations in a meaningful and sensible manner with all sentences being semantically correct, arguably imitating human conversation. This model can also be fine tuned to perform other task like Question Answering, Summarization, Common Sense Reasoning, Reading Comprehension and Natural Language Inference.

6.2 Further Improvements

For additional enhancement of this model, the obvious tasks are

- Increase the model size from 117M to a larger model, the performance will increase exponentially as there are more parameters the model will have enough room to learn and store more information.
- Use a character based language model, a character based model might achieve a better performance than the current model in some tasks(next word prediction).
- More finetuning data, a larger finetuning dataset size will also enhance the model especially the multi-turn model, the finetuning data for multi-turn was 32.23MB in comparison with one-turn with 300MB. Although obtaining multi-turn Japanese data might be strenuous, an increase in the data will yield great improvements.
- Better data quality for the dialogue models, some of the finetuning data were translated to Japanese using google translate, which was prevalent in the multi-turn model. Getting better data quality should improve the performance.

- Tweaking the training and finetuning parameter, most of the parameter chosen where tested but some where assumed. Tweaking the parameters might give better result but not an enormous one.

Bibliography

- [1] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.
- [2] A. Graves. Generating sequences with recurrent neural networks, 2013.
- [3] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.
- [4] K. ichi Funahashi and Y. Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, 6(6):801 – 806, 1993.
- [5] F. Jelinek, R. L. Mercer, L. R. Bahl, and J. K. Baker. Perplexity—a measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America*, 62(S1):S63–S63, 1977.
- [6] T. Mikolov and G. Zweig. Context dependent recurrent neural network language model. In *2012 IEEE Spoken Language Technology Workshop (SLT)*, pages 234–239, 2012.
- [7] P. J. Ortiz Suárez, B. Sagot, and L. Romary. Asynchronous Pipeline for Processing Huge Corpora on Medium to Low Resource Infrastructures. In P. Bański, A. Barbaresi, H. Biber, E. Breiteneder, S. Clematide, M. Kupietz, H. Lungen, and C. Iliadi, editors, *7th Workshop on the Challenges in the Management of Large Corpora (CMLC-7)*, Cardiff, United Kingdom, July 2019. Leibniz-Institut für Deutsche Sprache.
- [8] K. Papineni, S. Roukos, T. Ward, and W. J. Zhu. Bleu: a method for automatic evaluation of machine translation. 10 2002.
- [9] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks, 2012.
- [10] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2018.
- [11] A. M. Rush, S. Chopra, and J. Weston. A neural attention model for abstractive sentence summarization. *CoRR*, abs/1509.00685, 2015.

- [12] R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units, 2015.
- [13] A. Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, Mar 2020.
- [14] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014.
- [15] T. H. Toshinori Sato and M. Okumura. Implementation of a word segmentation dictionary called mecab-ipadic-neologd and study on how to use it effectively for information retrieval (in japanese). In *Proceedings of the Twenty-three Annual Meeting of the Association for Natural Language Processing*, pages NLP2017–B6–1. The Association for Natural Language Processing, 2017.
- [16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.
- [17] T. Wolf, V. Sanh, J. Chaumond, and C. Delangue. Transfertransfo: A transfer learning approach for neural network based conversational agents. *CoRR*, abs/1901.08149, 2019.