Towards an Online Empathetic Chatbot with Emotion Causes

Name - Mehuli Pal
Roll no - 1901CS78

BTP End Semester (7th Sem) Report

Computer Science and Engineering
Indian Institute of Technology Patna

Link to Research Paper: https://arxiv.org/pdf/2105.11903.pdf
Drive link for Code Implementation:
https://drive.google.com/drive/folders/1w0VIjS_2_2qq8Y26F-BDlJDo_-qAzHBb?usp=sharing
Link to Training Code:
https://colab.research.google.com/drive/14HiuNXjuj4YJ-FtXeuaIOgekYdWQT6MT?usp=shar
ing
Link to End Semester Presentation:
https://docs.google.com/presentation/d/1HeEkDh3AeGZNJv9DyayeG2BxE99qhooQ7VAxie7
yh0w/edit?usp=sharing
Link to Mid Semester Presentation:
https://docs.google.com/presentation/d/1_wtFO4W-BEne3mReNIAJQJYfplwHtFBqwPOUW
FtC64A/edit?usp=sharing
Link to English Dataset:
https://drive.google.com/file/d/1epM6283zJp70pNatrubRkP5iO3EmbdxT/view?usp=sharing
Link to Chinese Dataset: https://github.com/XiaoMi/emma/tree/master/data/raw
Link to English Dataset (used as reference):
https://github.com/declare-lab/RECCON/blob/main/data/original_annotation/dailydialog_tra
in.json
Link to Python Script for making English dataset from scratch and annotating reply labels:
https://colab.research.google.com/drive/10PZy4NSHa3CtM5VSa29tLSQ4OYQMldu5?usp=s
haring

Abstract

Existing emotion-aware conversational models usually focus on controlling the response contents to align with a specific emotion class. In contrast, empathy is the ability to understand and concern with the feelings and experiences of others. Since empathy plays a vital role in amicable social conversation and trustful social bonding, it is critical to endow social chatbots with the ability to empathize. Hence, it is critical to learn the causes that evoke the users' emotions for empathetic responding, a.k.a. emotion causes. We leverage counseling strategies to gather emotion causes in online environments and develop an empathetic chatbot to utilize causal emotion information. On a real-world online dataset, we verify the effectiveness of the proposed approach by comparing our chatbot with several SOTA methods using automatic metrics, expert-based human judgments, and user-based online evaluation.

Literature Review

- The first chatbot was ELIZA, constructed in 1966. Its ability to communicate was limited, but it was a source of inspiration for the subsequent development of other chatbots.
- In 1972, PARRY appeared. It is considered more advanced than ELIZA as it is supposed to have a "personality" and a better controlling structure.
- The term Chatterbot was first mentioned in 1991. It was a TINYMUD (multiplayer real-time virtual world) artificial player, whose primary function was to chat.
- In 2001, there was a real evolution in chatbot technology with the development of SmarterChild, which was available on Messengers like America Online and Microsoft.
- Apple Siri, IBM Watson, Google Assistant, Microsoft Cortana, and Amazon Alexa are the most popular voice assistants of today.

Introduction

EMMA is a shorthand representation for online "**Em**pathetic chatbot based on the user e**m**otion c**a**uses". It not only learns the ***class*** of users' emotions but also ***causes*** that evoke the users' emotions for empathetic responding, a.k.a. *emotion causes*. Apart from understanding what is being discussed, EMMA also acknowledges the implied feelings of the conversation and responds appropriately.

Turn	Utterance	Strategy & Cause
U1	I'm upset.	None
S1	Everything will be OK.	None

Fig - Existing approach: **EMOTION CLASS**

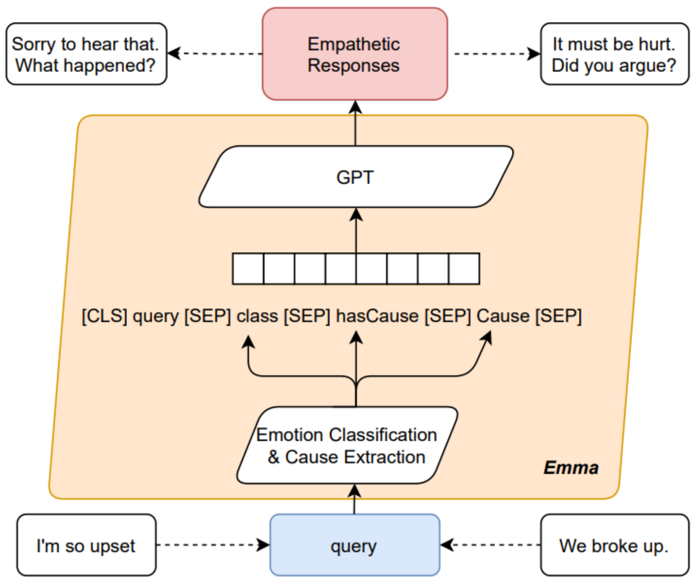
Turn	Utterance	Strategy & Cause
U1	I’m upset.	None
S1	Sorry to hear that. What happened?	Effective questioning
U2	We ***broke up***.	Emotion cause
S2	Oh dear, it must be hurt. Did you argue for something?	Active listening

Fig - EMMA: **EMOTION CLASS** + **EMOTION CAUSES**

Approach

1. When a user starts a conversation, our approach first detects user emotion class and recognizes the emotion causes.
2. If no emotion cause is detected, our chatbot Emma directs users to self-disclose more based on effective questioning and active listening, the two counseling strategies that are often used by psychologists to encourage further engagement and gather detailed information during counseling conversations.
3. Finally, Emma produces empathetic responses based on the conversation history, detected emotion class as well as the emotion causes.

Architecture


Sorry to hear that. What happened?
Empathetic Responses
It must be hurt. Did you argue?
GPT
[CLS] query [SEP] class [SEP] hasCause [SEP] Cause [SEP]
Emotion Classification & Cause Extraction
Emma
I'm so upset
query
We broke up.

Once emotion class and emotion cause have been extracted from user query, the formatted input **[CLS] query [SEP] class [SEP] hasCause [SEP] Cause [SEP]** is now passed through a GPT model (preferably, GPT-2). The output appears in the form of empathetic responses.

Mathematical Formula

Our task is to learn a response generation model via ***maximum likelihood estimation*** and, generate a response $Y = \{y1, \cdots, yT\}$ according to the user query $X = \{x1, \cdots, xN\}$ and history conversations H. To make the generated responses more empathetic, we also leverage emotional information, like emotion class label L and emotion causes C.

The conditional probability of generating a response Y can be formulated as:

$$\boxed{P_Y = \prod_{i=1}^{T} P(y_t \mid y_{0:t-1}, X, H, L, C)}$$

where,

Query: $X = \{x1, \cdots, xN\}$
H: History conversations
L: Emotion class label
C: Emotion causes
Response: $Y = \{y1, \cdots, yT\}$

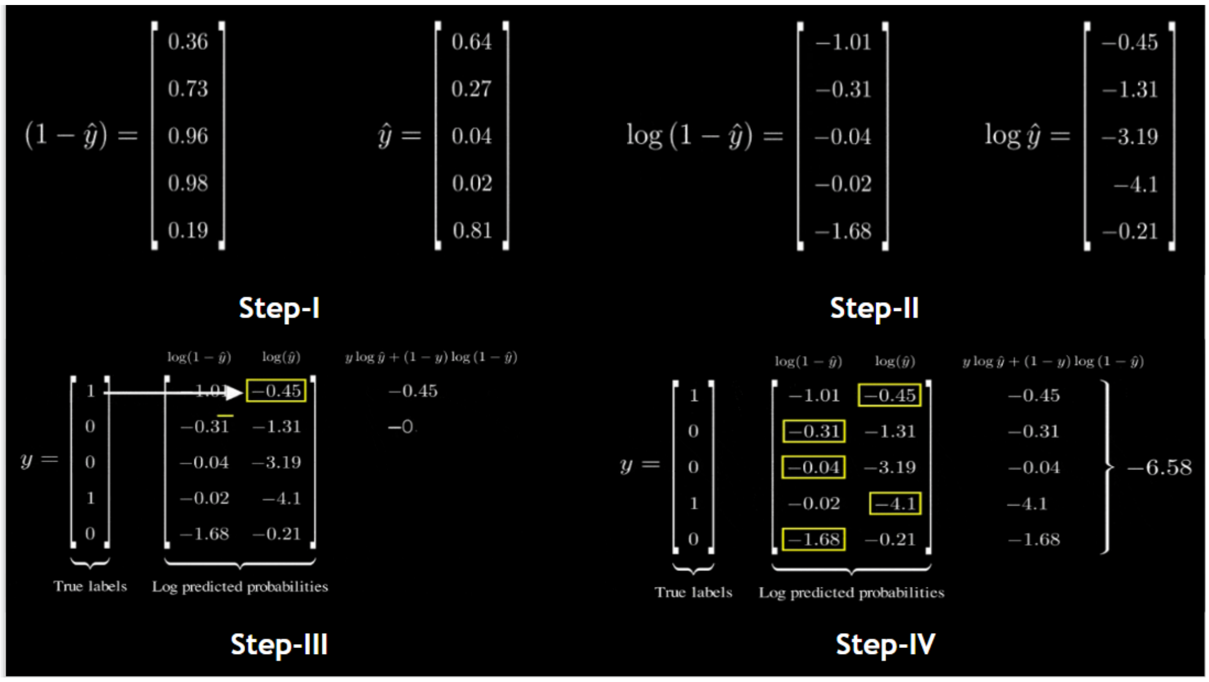
[CLS] [speaker1] query1 [speaker2] response1 [speaker1] query2 [SEP] label [SEP] hasCause [SEP] Cause [SEP] -> This is the input format that we are going to feed into our model.

Negative Log-Likelihood Loss

Negative log-likelihood minimization is a proxy problem to the problem of ***maximum likelihood estimation***. The loss function is given by ***L(ˆy, y)***, where **ˆ*y*** represents the ***predicted output*** and ***y*** represents the ***true output***. The training objective is then to minimize the loss across the different training examples. The formula for calculating NLL loss:

$$\boxed{L_{\text{cross-entropy}}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_i y_i \log(\hat{y}_i)}$$

For a binary class classification, NLL loss can be calculated as:


$(1-\hat{y}) = [0.36, 0.73, 0.96, 0.98, 0.19]$
$\hat{y} = [0.64, 0.27, 0.04, 0.02, 0.81]$
Step-I
$\log(1-\hat{y}) = [-1.01, -0.31, -0.04, -0.02, -1.68]$
$\log \hat{y} = [-0.45, -1.31, -3.19, -4.1, -0.21]$
Step-II
$y = [1, 0, 0, 1, 0]$
$\log(1-\hat{y})$
$\log(\hat{y})$
$y \log \hat{y} + (1-y)\log(1-\hat{y})$
True labels
Log predicted probabilities
Step-III
-6.58
Step-IV

Adam Optimizer

Adaptive Moment Estimation is an optimization technique for gradient descent. The method is really efficient when working with a large problem involving a lot of data or parameters. It requires less memory and is efficient.

Adam optimizer involves a combination of two gradient descent methodologies:

- Gradient Descent with Momentum
- Root Mean Square Propagation (RMSP)

Weights are updated using the following update rule in Adam optimizer:

$$w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$$

Implementation

1. **Setting Model Class**

   Select GPT2HeadModel if we train on GPT-2, otherwise select OpenAIGPTLMHeadModel.

```
'''
    If we are using GPT-2, model class is GPT2LMHeadModel.
    For other models like BERT, model class is OpenAIGPTLMHeadModel.
'''
model_class = OpenAIGPTLMHeadModel if not args.gpt2 else GPT2LMHeadModel
model = model_class.from_pretrained(args.model_checkpoint)
print("Model:", model_class)
```

2. **Setting Tokenizer Class & adding Special Tokens**

   Special tokens include [CLS], [SEP], [None].

```
'''
    If we are using GPT-2, tokenizer class is GPT2Tokenizer.
    For other models like BERT, tokenizer class is BertTokenizer.
'''
tokenizer_class = BertTokenizer if not args.gpt2 else GPT2Tokenizer
tokenizer = tokenizer_class.from_pretrained(args.model_checkpoint, do_lower_case=True)
print("Tokenizer:", tokenizer_class)

# Add special tokens to vocabulary
special_tokens_dict = {'additional_special_tokens': ['[CLS]', '[SEP]', '[None]']}
num_added_toks = tokenizer.add_special_tokens(special_tokens_dict)
model.resize_token_embeddings(len(tokenizer))
```

3. **Adam Optimizer**

```
'''
    We use AdamW (Adaptive Moment Estimation) optimizer.
    It is an optimization technique for gradient descent.
    Adam optimizer involves a combination of two gradient descent methodologies:
    (a) Momentum (b) Root Mean Square Propagation (RMSP)
'''
optimizer = torch.optim.AdamW(model.parameters(), lr=args.lr)
print('\nOptimizer:', optimizer)
```

4. **Create Data Loaders**

   Create training and validation data loaders to load data in batches. Batch size is taken as user input.

```
# Create training and validation data loaders to load the data in batches
# train_batch_size and valid_batch_size are taken as user inputs via arguments
loader_class = data_loaders
logger = logging.getLogger(__file__)
train_loader, val_loader = loader_class(args, tokenizer, logger)
```

5. **Extracting input_ids, token_type_ids, lm_labels for TRAINING**

```
'''
    input_ids:        Indices of input sequence tokens in the vocabulary.
                      For eg, [CLS] My name is Mehuli. [SEP] I am in BTech. [SEP],
                      which can be interpreted as
                      [101, 8, 5, 6, 7, 3, 102, 9, 2, 1, 4, 10, 102]

    token_type_ids:   Same as the segment ids, which differentiates sentence 1 and 2
                      in 2-sentence tasks. In [0, 0, 0, 1, 1, 1, 1], '0' represent
                      sentence 1 & '1' represents sentence 2.

    lm_labels:        Language modeling labels. Indices are selected in [-100, 0,
                      ..., config.vocab_size]. All labels set to -100 are ignored
                      (masked), the loss is only computed for labels in [0, ...,
                      config.vocab_size]
'''
input_ids, token_type_ids, lm_labels = tuple(input_tensor.to(args.device) for
input_tensor in batch)
```

6. **Calculating Error, Error Derivative for Backpropagation & Gradient Clipping**

   Backpropagation is a widely used algorithm for training neural networks. It computes the gradient of the loss function with respect to the network weights and is very efficient, rather than naively directly computing the gradient with respect to each individual weight.

   Gradient Clipping is a method where the error derivative is changed or clipped to threshold during backpropagation through the network, and the clipped gradients are used to update the weights.

```
# Loss function / Cost function / Error function
lm_loss = model(input_ids, labels=lm_labels, token_type_ids=token_type_ids).loss
loss = lm_loss / args.gradient_accumulation_steps
'''
    backward() method in Pytorch is used to calculate the error derivative/
    gradient during backpropagation in a neural network.
'''
loss.backward()
'''
    Gradient Clipping is a method where the error derivative is changed or clipped
    to a threshold during backpropagation through the network, and the clipped
    gradients are used to update the weights.
'''
torch.nn.utils.clip_grad_norm_(model.parameters(), args.max_norm) # gradient clipping
```

7. **Extracting lm_logits and lm_labels for VALIDATION**

   Logits layer means the layer that feeds into softmax (or other such normalization). The output of the softmax gives the probabilities for the classification task and its input is logits layer. The logits layer typically produces values from -infinity to +infinity and the softmax layer transforms it to values from 0 to 1.

```
'''
    In context of deep learning the logits layer means the layer that feeds into
    softmax (or other such normalization). The output of the softmax gives the
    probabilities for the classification task and its input is logits layer. The
    logits layer typically produces values from -infinity to +infinity and the
    softmax layer transforms it to values from 0 to 1.
'''
lm_logits = model(input_ids, token_type_ids=token_type_ids).logits
lm_logits_flat_shifted = lm_logits[..., :-1,:].contiguous().view(-1, lm_logits.size(-1))
lm_labels_flat_shifted = lm_labels[..., 1:].contiguous().view(-1)
return lm_logits_flat_shifted, lm_labels_flat_shifted
```

8. **Setting PiecewiseLinear Scheduler**

   Initially, the learning rate is set to user-inputted learning rate, which is linearly decreased over epochs, until the learning rate becomes 0.

```
'''
    Initially, sets the learning rate to args.lr (i.e., user inputted learning rate)
    in the 0th iteration, then decreases linearly to 0.0 until the end of the
    iterations, given by (args.n_epochs * len(train_loader)
'''
scheduler = PiecewiseLinear(optimizer, "lr", [(0, args.lr),
                                          (args.n_epochs * len(train_loader), 0.0)])
```

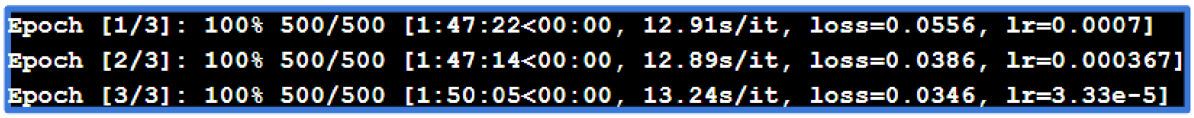
9. **Calculating Negative Log-Likelihood (NLL) Loss**

```
'''
    NLL Loss: Negative log-likelihood minimization is a proxy problem to the
    problem of maximum likelihood estimation. Cross-entropy and negative
    log-likelihood are closely related mathematical formulations. The essential
    part of computing the negative log-likelihood is to "sum up the correct log
    probabilities". After converting logits into probabilities by passing
    through a softmax layer, the purpose of NLL is to take the output
    probabilities and measure the distance from the truth values.
'''
metrics = {"negative_log_likelihood_loss":
Loss(torch.nn.CrossEntropyLoss(ignore_index=-1), output_transform=lambda x: (x[0], x[1]))}
```

Results & Plots

1. **XiaoMi EMMA Chinese Dataset**

   Language: Chinese
   Number of records: 80,000
   Model: bert-base-uncased
   Learning rate: 0.001
   Number of epochs: 3
   Training batch size: 16
   Validation batch size: 16

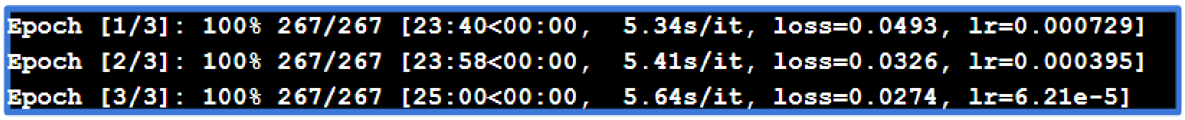


```
Epoch [1/3]: 100% 500/500 [1:47:22<00:00, 12.91s/it, loss=0.0556, lr=0.0007]
Epoch [2/3]: 100% 500/500 [1:47:14<00:00, 12.89s/it, loss=0.0386, lr=0.000367]
Epoch [3/3]: 100% 500/500 [1:50:05<00:00, 13.24s/it, loss=0.0346, lr=3.33e-5]
```

2. **Self-prepared and self-annotated English Dataset**
   a. **Dataset preparation**: Using an existing *dailydialog.json dataset* at https://github.com/declare-lab/RECCON/blob/main/data/original_annotation/dailydialog_train.json, I have prepared an English dataset for EMMA using annotation.py python script.
   b. **Dataset annotation:** Emotion classes (reply_labels) have been annotated using one of the 11 emotion labels.

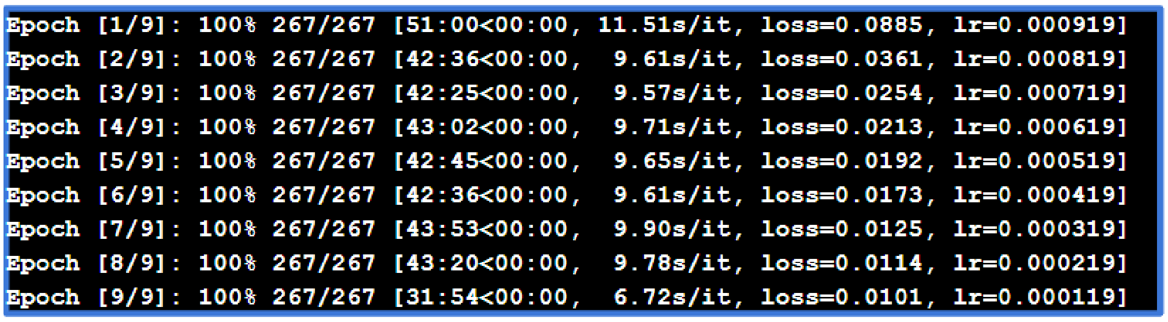
```
consoling
sympathizing
acknowledging
sharing own thoughts/opinion
wishing
appreciating
agreeing
questioning
suggesting
expressing care or concern
encouraging
```

   c. **Language**: Chinese
   Number of records: 4,269
   Model: bert-base-uncased
   Learning rate: 0.001
   Number of epochs: 3
   Training batch size: 16
   Validation batch size: 16



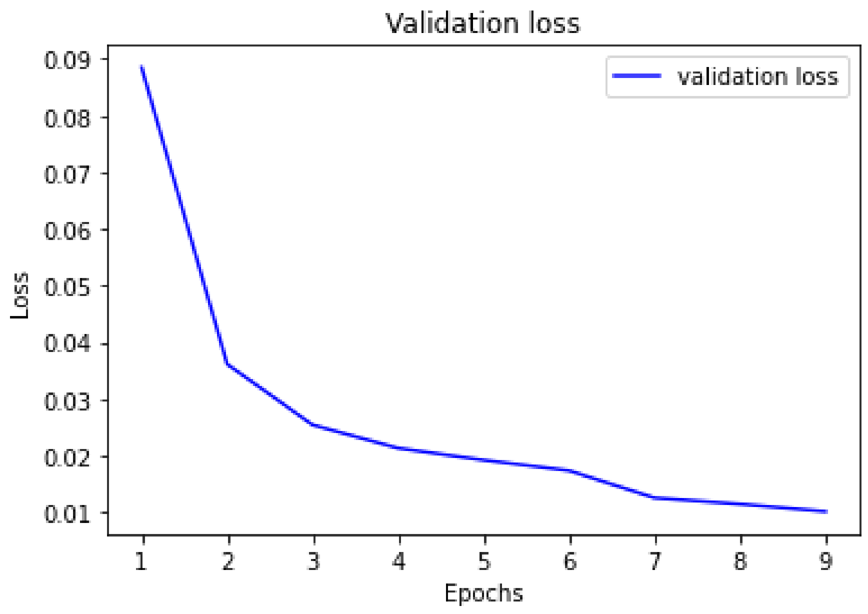
```
Epoch [1/3]: 100% 267/267 [23:40<00:00,  5.34s/it, loss=0.0493, lr=0.000729]
Epoch [2/3]: 100% 267/267 [23:58<00:00,  5.41s/it, loss=0.0326, lr=0.000395]
Epoch [3/3]: 100% 267/267 [25:00<00:00,  5.64s/it, loss=0.0274, lr=6.21e-5]
```

d. **Language**: Chinese
   Number of records: 4,269
   Model: gpt2
   Learning rate: 0.001
   Number of epochs: 9
   Training batch size: 16
   Validation batch size: 16

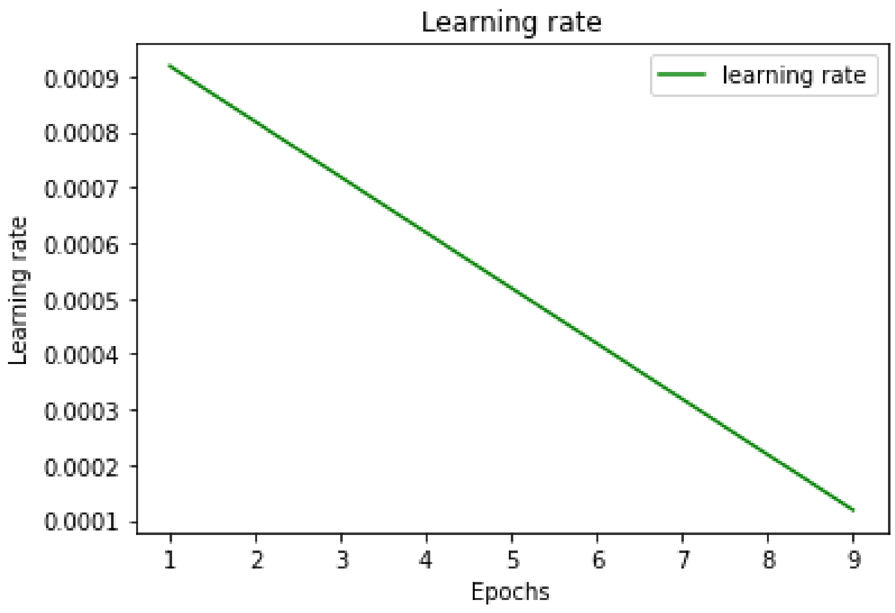


```
Epoch [1/9]: 100% 267/267 [51:00<00:00, 11.51s/it, loss=0.0885, lr=0.000919]
Epoch [2/9]: 100% 267/267 [42:36<00:00,  9.61s/it, loss=0.0361, lr=0.000819]
Epoch [3/9]: 100% 267/267 [42:25<00:00,  9.57s/it, loss=0.0254, lr=0.000719]
Epoch [4/9]: 100% 267/267 [43:02<00:00,  9.71s/it, loss=0.0213, lr=0.000619]
Epoch [5/9]: 100% 267/267 [42:45<00:00,  9.65s/it, loss=0.0192, lr=0.000519]
Epoch [6/9]: 100% 267/267 [42:36<00:00,  9.61s/it, loss=0.0173, lr=0.000419]
Epoch [7/9]: 100% 267/267 [43:53<00:00,  9.90s/it, loss=0.0125, lr=0.000319]
Epoch [8/9]: 100% 267/267 [43:20<00:00,  9.78s/it, loss=0.0114, lr=0.000219]
Epoch [9/9]: 100% 267/267 [31:54<00:00,  6.72s/it, loss=0.0101, lr=0.000119]
```

e. **Validation Loss vs Epochs Graph**



f. Learning Rate vs Epocha Graph



References

- https://towardsdatascience.com/openai-gpt-2-understanding-language-generation-through-visualization-8252f683b2f8
- https://towardsdatascience.com/how-to-fine-tune-gpt-2-for-text-generation-ae2ea53bc272
- https://huggingface.co/bert-base-uncased
- https://huggingface.co/docs/transformers/model_doc/bert#transformers.BertForMaskedLM.forward
- https://github.com/XiaoMi/emma/tree/master/data/raw
- https://github.com/declare-lab/RECCON/blob/main/data/original_annotation/dailydialog_train.json
- https://towardsdatascience.com/cross-entropy-negative-log-likelihood-and-all-that-jazz-47a95bd2e81
- https://www.geeksforgeeks.org/intuition-of-adam-optimizer/
- https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c
- https://www.scss.tcd.ie/~koidlk/cs4062/Loss-Functions.pdf
- https://jalammar.github.io/illustrated-gpt2/