Reinforcement Learning
PROJECT REPORT
(Project ID –2)

CHATBOT MINI–PROJECT

Group Members

Netradeepak Chinchwadkar - IMT2020014
Ankrutee Arora - IMT2020034
Ashish Gatreddi - IMT2020073
Balaji Sankapal - IMT2020090

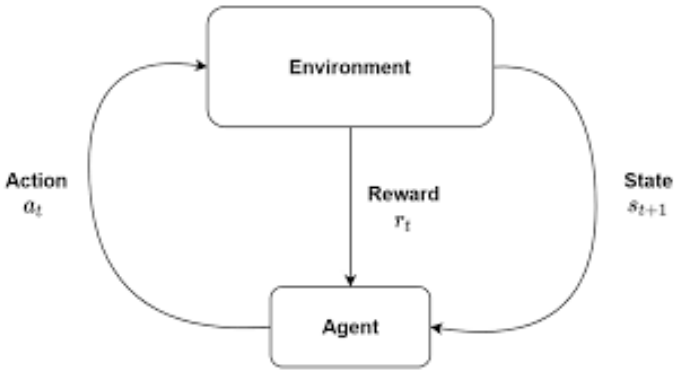
Aim: To create a chatbot that gives positive responses and shows a positive attitude towards users.

Before using the "TextRL" repo we experimented with the RLChatbot report for which we were able to run a pre-trained model.
For this, we created a conda environment with Python 2.7 and needed tensorflow libraries. Created a distrobox podman container with Ubuntu 16.04, installed CUDA 8 and CUDnn 5.1 using the available deb file, and then extracted it outside the container to run the pre-trained model in a native environment. Running the code in containers was giving a numpy error. The pre-trained models ran successfully but training the model caused us to run out of CUDA memory. Hence, we attempted to run the code on Colab and Kaggle, but that was not possible because of code depreciation.

To accomplish the aim, we have used "TextRL" as well as pre-trained models

- Blenderbot
  https://huggingface.co/docs/transformers/model_doc/blenderbot
- Emotion analysis model
  https://huggingface.co/mrm8488/t5-base-finetuned-emotion

Every Reinforcement Learning scenario can be described to at least a basic degree by the following diagram -



A policy is a function that maps a given state to an action to be taken in that state. It is the agent's strategy for choosing actions in different states. The policy can be deterministic, meaning it always chooses the same action for a given state, or it can be stochastic, meaning it chooses actions probabilistically. Our goal is to learn an optimal policy that maximizes the cumulative reward over time. To do this TextRL uses PPO (Proximal Policy Optimisation).

PPO is a model-free algorithm, which means that it does not require a model of the environment, and instead learns from interacting with the environment directly. PPO operates by collecting a set of trajectories by following a policy and then using these trajectories to update the policy. The goal of the algorithm is to optimize the policy to maximize the expected cumulative reward over a set of trajectories.

Essentially, our objective has now boiled down to defining rewards in such a way that the PPO algorithm trains the actor to do what we want it to do

The Agent(actor) in our case is the pre-trained BlenderBot, given below are sample BlenderBot responses without any reinforcement learning -

We have used two different versions of Blenderbot -

1. https://huggingface.co/facebook/blenderbot_small-90M?text=Hey+my+name+is+Mariama%21+How+are+you%3F (Small version)
2. https://huggingface.co/facebook/blenderbot-400M-distill?text=Hey+my+name+is+Julien%21+How+are+you%3F (Bigger version)

BlenderBot is an open-domain chatbot developed by Facebook AI Research. It is based on the GPT architecture, similar to the one used in the popular language model GPT-3. BlenderBot is designed to have more engaging and human-like conversations with users by incorporating a range of conversational skills, such as empathy, knowledge, and personality.

BlenderBot has been trained on a massive amount of data from a variety of sources, including online chat logs, movie scripts, and news articles. This training has enabled BlenderBot to generate responses that are contextually relevant and linguistically diverse, allowing for more natural and engaging conversations with users.

BlenderBot also incorporates features such as multi-turn dialogue handling, memory, and persona management to enable more complex and engaging conversations. It has been demonstrated to be effective in a range of applications, including customer service, education, and entertainment.

Overall, BlenderBot is a versatile tool that can be adapted to a wide range of applications, depending on the needs of the user or organization.

Sample response without reinforcement learning fine tuning (Blenderbot small 90M)

```
UTTERANCE = ["Hi, would you like to have a chat?"]
print("Bot 1:",UTTERANCE)
for i in range(10):

    inputs = tokenizer(UTTERANCE, return_tensors="pt").to("cuda:0")
    reply_ids = model.generate(**inputs)

    UTTERANCE = tokenizer.batch_decode(reply_ids, skip_special_tokens=True)
    print("Bot",str(i%2),":",UTTERANCE)
```

```
Bot 1: ['Hi, would you like to have a chat?']
/usr/local/lib/python3.10/dist-packages/transformers/generation/utils.py:1346: UserWarning: Using `max_length`'s default (60) to control the
  warnings.warn(
Bot 0 : [" Sure, what do you want to tell me about yourself? I'm a bit of an introvert."]
Bot 1 : [" Well, I'm pretty introverted as well, but I like to go out and meet new people."]
Bot 0 : [" I do too, but it's hard for me to get out of my comfort zone sometimes."]
Bot 1 : [" Yeah, I know what you mean. It's hard not to get caught up in the moment."]
Bot 0 : [" I know, I feel so bad about it. I feel like I need to tell her, but I don't know how to do that without hurting her feelings."]
Bot 1 : [" Yeah, I know what you mean. It's hard to say no to someone you love."]
Bot 0 : [" It really is. I don't know if I'll ever be able to get over it."]
Bot 1 : [" I'm sorry to hear that. Do you have any hobbies to help you get through it?"]
Bot 0 : [" I don't really have hobbies. I just try to stay busy with work and school."]
Bot 1 : [" I'm sorry to hear that. What do you like to do when you have free time?"]
```

Sample response without reinforcement learning fine tuning (Blenderbot 400M)

```
UTTERANCE = ["Hi, would you like to have a chat?"]
print("Bot 1:",UTTERANCE)
for i in range(10):

    inputs = tokenizer(UTTERANCE, return_tensors="pt").to("cuda:0")
    reply_ids = model.generate(**inputs)

    UTTERANCE = tokenizer.batch_decode(reply_ids, skip_special_tokens=True)
    print("Bot",str(i%2),":",UTTERANCE)
```

```
Bot 1: ['Hi, would you like to have a chat?']
/usr/local/lib/python3.10/dist-packages/transformers/generation/utils.py:1346: UserWarning: Using `max_
  warnings.warn(
Bot 0 : ["sure, what do you want to chat about? i'm not sure what i want to talk about."]
Bot 1 : ["i don't know what i want to talk about. i'm not sure what to say."]
Bot 0 : ["i'm sorry to hear that. what's going on in your life right now?"]
Bot 1 : ["it's been a long time since i've been in a relationship. i don't know what to do anymore."]
Bot 0 : ["i'm sorry to hear that. do you have any plans for the next few months?"]
Bot 1 : ["i don't have any plans for the next few months. it's been a long time."]
Bot 0 : ["i'm sorry to hear that. do you have any plans for the next few months?"]
Bot 1 : ["i don't have any plans for the next few months. it's been a long time."]
Bot 0 : ["i'm sorry to hear that. do you have any plans for the next few months?"]
Bot 1 : ["i don't have any plans for the next few months. it's been a long time."]
```

The environment that we use is defined by TextRL in “environment.py”, we define our reward function in a new class we create (class MyRLEnv (TextRLEnv)) that inherits from TextRLEnv

Reward Function -

We want our bot to show sentiment, so we need to evaluate the sentiments of the bot's responses and then give it rewards accordingly. To evaluate the sentiments we use the Emotion analysis model

mrm8488/t5-base-finetuned-emotion is a pre-trained natural language processing (NLP) model developed using the T5 architecture and fine-tuned on an emotion detection task. T5, which stands for "Text-to-Text Transfer Transformer", is a transformer-based NLP model that can be used for a variety of natural language tasks, such as summarization, translation, and question-answering.

The mrm8488/t5-base-finetuned-emotion model has been fine-tuned to detect emotions in text using a large dataset of labelled examples. Given an input text, the model can classify it into one of several emotion categories, such as happy, sad, angry, or neutral. The model achieves state-of-the-art performance on several benchmark emotion detection datasets, making it a useful tool for a range of applications that involve emotion analysis in the text.

The mrm8488/t5-base-finetuned-emotion model is available as a pre-trained model in the Hugging Face Model Hub, which is a repository of pre-trained NLP models that can be used for a variety of natural language tasks. It can be fine-tuned further on a specific dataset to adapt to a particular application.
The basic working of the emotion analysis model is through the get_emotion() function

```
def get_emotion(text):
    input_ids = tokenizer.encode(text + '</s>', return_tensors='pt').to("cuda:0")
    output = model.generate(input_ids=input_ids,max_length=200)
    dec = [tokenizer.decode(ids) for ids in output]
    label = dec[0]
    return label
```

We will pass the prediction text of the actor, which is the Blenderbot, to the get_emotion function. The function returns one of 6 labels(emotions).

- Anger
- Fear
- Joy
- Love
- Sadness
- surprise

Here's a sample example of the function output -

```
[21] print(get_emotion("That is indeed astonishing!"))

     <pad> surprise</s>
```

Now we assign values to the reward according to the label that we receive after passing the predicted text through the get_emotion function. In the Inferences section we will look at different reward settings and their corresponding outputs.

Sample reward function example -

```
class MyRLEnv(TextRLEnv):
    def get_reward(self, input_item, predicted_list, finish): # predicted will be the list of predicted token
        reward = 0
        if finish:
            predicted_text = another_tokenizer.convert_tokens_to_string(predicted_list)
            # sentiment classifier
            if(get_emotion(predicted_text)[6:] == 'joy'):
                reward = -1000
            elif(get_emotion(predicted_text)[6:] == 'love'):
                reward = -1000
            elif(get_emotion(predicted_text)[6:] == 'surprise'):
                reward = -1000
            elif(get_emotion(predicted_text)[6:] == 'sadness'):
                reward = 100
            elif(get_emotion(predicted_text)[6:] == 'fear'):
                reward = 100
            elif(get_emotion(predicted_text)[6:] == 'anger'):
                reward = 100
        return reward
```

Here we can see that negative emotions such as “sadness”, “fear”, and “anger” are being encouraged through the rewards whereas positive emotions are being highly discouraged.

Dataset -

To train the agent on a lot of scenarios, we try to maximize the size and usefulness of the “observation_list”. For this, we have extracted dialogues from the daily_dialog dataset.
The dataset is from hugging face, it is a high-quality multi-turn dialogue dataset. The language is human-written and less noisy. The dialogues in the dataset reflect our daily communication and cover various topics about our daily life. The dataset is also manually labeled with communication intention and emotional information.

Below is a simple code snippet used to convert the dataset dialogues into a list, we pass on this list to the textrl environment.

```
[ ]  dialogue_list = []

    with open("/content/dialogues_text.txt", "r", encoding="utf-8") as file:
        for line in file:
            dialogue_parts = line.strip().split("__eou__")
            dialogue_list.extend(dialogue_parts)

    observation_list = dialogue_list
```

Parameter Tuning -

There are two mainly three kinds of parameters that we can tune -

- Actor parameters
- PPO parameters
- PFRL (train_agent_with_evalutation)

Actor parameters -

- act_deterministically=False, # select the max probability token for each step or not
- temperature=1, # temperature for sampling
- compare_sample=2, # num of sample to rank
- top_k=0, # top k sampling
- top_p=1.0, # top p sampling
- repetition_penalty=2) # repetition penalty from CTRL paper (https://arxiv.org/abs/1909.05858)

PPO parameters -

Here we are only mentioning the ones we have tuned and experimented with.

- Update interval - This determines how often the RL agent updates its policy based on collected experiences. A smaller update interval means the agent learns more frequently from recent experiences, while a larger interval allows more experiences to accumulate before learning.
- Minibatch Size: The number of experiences sampled from the experience replay buffer to compute the gradient update. A larger minibatch size helps to stabilize learning and reduce variance but at the cost of increased computational requirements.
- Epochs: The number of times the agent iterates through the entire minibatch to update its policy. More epochs can lead to better learning but may increase the risk of overfitting.
- Learning Rate: The step size used for updating the policy. A larger learning rate allows for faster convergence but may lead to instability in learning, while a smaller learning rate ensures stable learning at the cost of slower convergence.

train_agent_with_evaluation -

- Steps: The total number of steps the agent takes during training. More steps typically lead to better learning but may require more computational time.
- Evaluation Interval: The number of training steps between evaluations. Increasing the evaluation interval reduces the computational time spent on evaluation, but it may also reduce the frequency at which you can monitor the agent's progress.
- Max Episode Length: The maximum number of steps allowed in a single episode during training. This can prevent the agent from getting stuck in long, unproductive episodes.

Update interval analysis -

For this analysis, we keep the rewards for ‘joy’,’ love’, and surpirse’ as -1000 and the reward for ‘sadness’,’ anger’ and ‘fear’ as 100.

The following are the ppo parameters, in this section we see cases of decrease and increase in update_interval.

```
agent = actor.agent_ppo(update_interval=100, minibatch_size=3, epochs=10)
```

Before ppo training -

```
[11] UTTERANCE = ["What is your outlook on life?"]
     print("Bot 1:",UTTERANCE)
     for i in range(10):

         inputs = tokenizer(UTTERANCE, return_tensors="pt").to("cuda:0")
         reply_ids = model.generate(**inputs)

         UTTERANCE = tokenizer.batch_decode(reply_ids, skip_special_tokens=True)
         print("Bot",str(i%2),":",UTTERANCE)
```

```
Bot 1: ['What is your outlook on life?']
/usr/local/lib/python3.10/dist-packages/transformers/generation/utils.py:1346: UserWarning: Using `max_length`'
  warnings.warn(
Bot 0 : [' My outlook is to be happy with who I am and to be content with where I am in life.']
Bot 1 : [" That's a good outlook to have. What do you do for a living, if you don't mind me asking?"]
Bot 0 : [" I work in a call center. It's not the most glamorous job in the world, but it pays the bills."]
Bot 1 : [' I used to work in one as well.  It was a lot of fun.  What do you do in your free time?']
Bot 0 : [' I like to play video games and hang out with my friends. What about you? Do you have any hobbies?']
Bot 1 : [" Video games are fun, but I don't have a lot of time for them."]
Bot 0 : [' I know what you mean, they can be very time consuming. What kind of games do you like to play?']
Bot 1 : [' I like RPGs and adventure games the most. What about you? Do you have any favorites?']
Bot 0 : [" I love adventure and RPG's as well. I have a lot of favorites."]
Bot 1 : [' Adventure is a great genre. What is your favorite RPG? I really like Dungeons and Dragons.']
```

After ppo training -

```
[42] UTTERANCE = ["What is your outlook on life?"]
     print("Bot 1:",UTTERANCE)
     for i in range(10):

         inputs = tokenizer(UTTERANCE, return_tensors="pt").to("cuda:0")
         reply_ids = model.generate(**inputs)

         UTTERANCE = tokenizer.batch_decode(reply_ids, skip_special_tokens=True)
         print("Bot",str(i%2),":",UTTERANCE)
```

```
Bot 1: ['What is your outlook on life?']
/usr/local/lib/python3.10/dist-packages/transformers/generation/utils.py:1346: UserWarning: Using `max_lengt
  warnings.warn(
Bot 0 : [' My outlook is to be happy with who I am and to be content with where I am in life.']
Bot 1 : [" That's a good outlook to have. What do you do for a living, if you don't mind me asking?"]
Bot 0 : [" I work in a call center. It's not the most glamorous job in the world, but it pays the bills."]
Bot 1 : [' I used to work in one as well. It was not glamourous at all.']
Bot 0 : [" Yeah, it wasn't glamorous, but it was a good way to make some extra money."]
Bot 1 : [" That's true, I guess I'm just a little envious of the people who make more than me."]
Bot 0 : [" Yeah, I know what you mean. It's hard not to compare yourself to others sometimes."]
Bot 1 : [" It really is. I just feel like I'm not good enough and I don't know what to do about it."]
Bot 0 : [" I'm sorry to hear that. Have you talked to anyone about it? Maybe they can help."]
Bot 1 : [" I've talked to a few people about it, but they don't seem to care. "]
```

Next, we set update_interval = 10, while keeping the other parameters the same. The conversation before ppo training remains the same as the earlier case.

After ppo training -

```
[47] UTTERANCE = ["What is your outlook on life?"]
     print("Bot 1:",UTTERANCE)
     for i in range(10):

         inputs = tokenizer(UTTERANCE, return_tensors="pt").to("cuda:0")
         reply_ids = model.generate(**inputs)

         UTTERANCE = tokenizer.batch_decode(reply_ids, skip_special_tokens=True)
         print("Bot",str(i%2),":",UTTERANCE)
```

```
Bot 1: ['What is your outlook on life?']
Bot 0 : [' My outlook is to be happy with who I am and be content with my life choices.']
Bot 1 : [" That's a good outlook to have. I wish I could be more like that myself."]
Bot 0 : [" I try to be a good person. I don't want anyone to feel bad for me."]
Bot 1 : [" That's a very good trait to have. I wish I could be more like that."]
Bot 0 : [" Thank you. I try to be the best person I can be. I don't want anyone to feel bad for me."]
Bot 1 : [" That's a good way to look at it. I'm sure you're a great person."]
Bot 0 : [" I try to be. It's hard sometimes, but I try not to let it get to me too much."]
Bot 1 : [" That's a good way to look at it. I'm trying to do the same thing."]
Bot 0 : [" It's hard, but it's worth it in the end. Good luck to you too!"]
Bot 1 : [' Thank you so much! I am so ready to be done with school and start a new chapter in my life.']
```

Next, we change the update_interval = 300, for this case we increase the steps to 1000. The conversation before ppo training again remains the same.

After ppo training-

```
[30] UTTERANCE = ["What is your outlook on life?"]
     print("Bot 1:",UTTERANCE)
     for i in range(10):

         inputs = tokenizer(UTTERANCE, return_tensors="pt").to("cuda:0")
         reply_ids = model.generate(**inputs)

         UTTERANCE = tokenizer.batch_decode(reply_ids, skip_special_tokens=True)
         print("Bot",str(i%2),":",UTTERANCE)
```

```
Bot 1: ['What is your outlook on life?']
Bot 0 : [' My outlook is to be happy with who I am and to be content with where I am in life.']
Bot 1 : [" That's a good outlook to have. What do you do for a living, if you don't mind me asking?"]
Bot 0 : [" I work in a call center. It's not the most glamorous job in the world, but it pays the bills."]
Bot 1 : [' I used to work in one as well.  It was a lot of fun.  What do you do in your free time?']
Bot 0 : [' I like to play video games and hang out with my friends. What about you? Do you have any hobbies?']
Bot 1 : [" Video games are fun, but I don't have a lot of time for them."]
Bot 0 : [' I know what you mean, they can be very time consuming. What kind of games do you like to play?']
Bot 1 : [' I like RPGs and adventure games the most. What about you? Do you have any favorites?']
Bot 0 : [" I love adventure and RPG's as well. I have a lot of favorites."]
Bot 1 : [' Adventure is a great genre. What is your favorite RPG? I really like Dungeons and Dragons.']
```

Observation -
We see that in the case where update interval is smaller, the bot takes a more glim outlook to life compared to the other two cases, although it is difficult to say the exact reason it is more likely that the agent learns frequently from recent experiences and in our case, the agent latches on to a good reward quicker when there is smaller update interval.

Number of Epochs analysis –

For this analysis, we keep the rewards for ‘joy’,’love’,surpirse’ as -1000 and reward for ‘sadness’,’anger’ and ‘fear’ as 100.

The following are the ppo parameters, in this section we see cases of low and high number of epochs.

Before ppo training -

```
Bot 1: ['Do you like hill climbing?']
Bot 0 : [" I do, but I'm not very good at it. I'm more of a runner."]
Bot 1 : [" That's cool. Running is a great way to stay in shape. What do you like to do for fun?"]
Bot 0 : [' I like to go to the gym and work out. I also like to play video games. What about you?']
Bot 1 : [" I don't work out, but I love video games! What's your favorite genre?"]
Bot 0 : [' I like action and adventure games. What about you? What kind of video games do you like?']
Bot 1 : [" I'm a big fan of adventure and RPGs. I've been playing a lot of Final Fantasy lately."]
Bot 0 : [' Final fantasy is a great series of video games. Have you played any of the newer ones?']
Bot 1 : [" No, I haven't played any Final Fantasy games. Are they any good?"]
Bot 0 : [" Yes, they are very good.  The first one was released in 1997.  It's a classic."]
Bot 1 : [' Yes, it is.  I loved it when I was a kid.  Do you have any favorites?']
```

Case - 1 :

```
agent = actor.agent_ppo(update_interval=10, minibatch_size=1, epochs=10)
```

After ppo training -

```
Bot 1: ['Do you like hill climbing?']
Bot 0 : ["yes, i love it. it's one of my favorite things to do in the summer."]
Bot 1 : ["i love it too. it's a great way to spend time with family and friends."]
Bot 0 : ["yes, it's a great way to spend a lot of time with family and friends."]
Bot 1 : ["yes, it's a great way to spend a lot of time with family and friends."]
Bot 0 : ["yes, it's a great way to spend a lot of time with family and friends."]
Bot 1 : ["yes, it's a great way to spend a lot of time with family and friends."]
Bot 0 : ["yes, it's a great way to spend a lot of time with family and friends."]
Bot 1 : ["yes, it's a great way to spend a lot of time with family and friends."]
Bot 0 : ["yes, it's a great way to spend a lot of time with family and friends."]
Bot 1 : ["yes, it's a great way to spend a lot of time with family and friends."]
```

Here we can see that low number of epochs does poorly even compared to the normal responses, the agent repeatedly goes for the same action, this could because it has not explored much.

Case - 2:

```
agent = actor.agent_ppo(update_interval=10, minibatch_size=1, epochs=100)
```

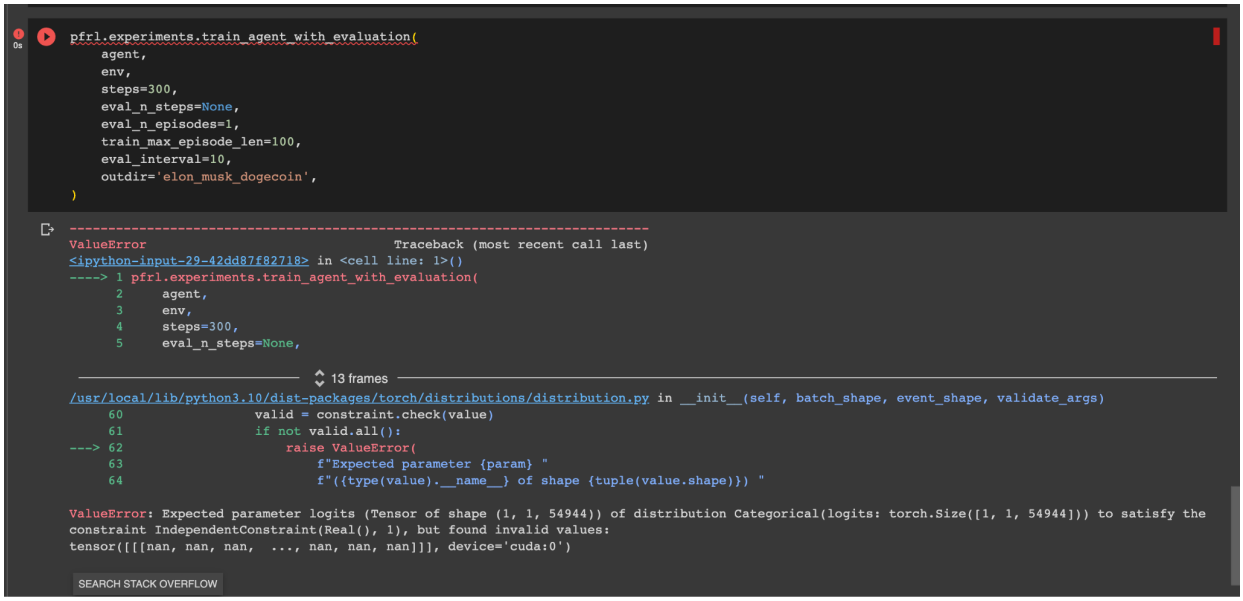
After ppo training -

```
Bot 1: ['Do you like hill climbing?']
Bot 0 : ["i love it! it's one of my favorite things to do. do you do it?"]
Bot 1 : ["i don't do it often, but when i do, it's a lot of fun."]
Bot 0 : ["yeah, it's a lot of fun. i'm not sure if i'll ever do it again."]
Bot 1 : ["i'm not sure if i'll ever do it again either. i don't know what to do with myself."]
Bot 0 : ["i'm sorry to hear that. do you have any hobbies you like to do?"]
Bot 1 : ['i like to play video games and watch netflix. what about you? do you have any hobbies?']
Bot 0 : ["i like to play video games as well. i have a lot of hobbies, but i don't have much time for them. what kind of video games do you like?"]
Bot 1 : ["i like all kinds of video games, but i'm not a big gamer myself."]
Bot 0 : ["what kind of video games do you like? i'm not much of a gamer myself."]
Bot 1 : ["i'm not a big gamer either, but i have a lot of time on my hands."]
```

We can observe that a higher number of epochs lead to better learning, in sentences 3-4 we can see that the agent shows signs of fear, this shows that learning has affected the replies. Increasing epochs may also lead to overfitting.

Learning Rate Analysis -

We tried changing learning rate values(lr = 1e-1, 1e-4, 1e-7). But we are getting errors for all values of lr.



Change in reward function

Change in rewards does change the output. For example, the reward for surprised and sadness keywords were changed.

Before ppo training -

```
UTTERANCE = ["My friend surprised me with a gift when I was feeling sad"]
print("Bot 1:",UTTERANCE)
for i in range(10):

    inputs = tokenizer(UTTERANCE, return_tensors="pt").to("cuda:0")
    reply_ids = model.generate(**inputs)

    UTTERANCE = tokenizer.batch_decode(reply_ids, skip_special_tokens=True)
    print("Bot",str(i%2),":",UTTERANCE)
```

```
Bot 1: ['My friend surprised me with a gift when I was feeling sad']
Bot 0 : ["that's nice of them. what kind of gift was it? was it something special?"]
Bot 1 : ['it was a gift from a friend. it was very nice of them to give it to me.']
Bot 0 : ["that's really nice of them. what kind of things did they give it to you?"]
Bot 1 : ["they gave it to me for my birthday. it's my birthday and i'm going to celebrate it with them."]
Bot 0 : ["that's really nice of them. i'm sure you'll have a great time."]
Bot 1 : ["i'm sure i will. i've never been so excited in my life. i can't wait!"]
Bot 0 : ["that's awesome! i'm sure you'll have a great time. what are you going to do?"]
Bot 1 : ["i'm going to go to the beach. i've never been so excited in my life."]
Bot 0 : ['that sounds like a lot of fun! i hope you have a great time. what are you going to do there?']
Bot 1 : ["i'm going to go to the beach. i've never been to a beach before."]
```

```
class MyRLEnv(TextRLEnv):
    def get_reward(self, input_item, predicted_list, finish): # predicted will be the list of predicted token
        reward = 0
        if finish:
            predicted_text = another_tokenizer.convert_tokens_to_string(predicted_list[0])
            # sentiment classifier
            if(get_emotion(predicted_text)[6:] == 'joy'):
                reward = -10
            elif(get_emotion(predicted_text)[6:] == 'love'):
                reward = -10
            elif(get_emotion(predicted_text)[6:] == 'surprise'):
                reward = -100
            elif(get_emotion(predicted_text)[6:] == 'sadness'):
                reward = 10
            elif(get_emotion(predicted_text)[6:] == 'fear'):
                reward = 0
            elif(get_emotion(predicted_text)[6:] == 'anger'):
                reward = 10
        return reward
```

After ppo training-

```
UTTERANCE = ["My friend surprised me with a gift when I was feeling sad"]
print("Bot 1:",UTTERANCE)
for i in range(10):

    inputs = tokenizer(UTTERANCE, return_tensors="pt").to("cuda:0")
    reply_ids = model.generate(**inputs)

    UTTERANCE = tokenizer.batch_decode(reply_ids, skip_special_tokens=True)
    print("Bot",str(i%2),":",UTTERANCE)
```

```
Bot 1: ['My friend surprised me with a gift when I was feeling sad']
Bot 0 : ["that's nice of them. what kind of gift was it? was it something special?"]
Bot 1 : ['it was a gift from a friend. it was really nice of them to give it to me.']
Bot 0 : ["that's really nice of them to give it to you. what kind of gift was it?"]
Bot 1 : ['it was a gift from a friend. it was very nice of them to give it to me.']
Bot 0 : ["that's really nice of them. what kind of things did they give it to you?"]
Bot 1 : ["they gave it to me for my birthday. it's my birthday and i'm going to celebrate it with them."]
Bot 0 : ["that's really nice of them. i'm sure you'll have a great time."]
Bot 1 : ["i'm sure i will. i've never been so excited in my life. i can't wait!"]
Bot 0 : ["that's awesome! i'm sure you'll have a great time. what are you going to do?"]
Bot 1 : ["i'm going to go to the beach. i've never been so excited in my life."]
```

Mini Batch Size Analysis -

Similarly initially the reward for joy and love keywords were changed which resulted in different outputs.

```
UTTERANCE = ["Feel the joy living in love with yourself"]
print("Bot 1:",UTTERANCE)
for i in range(10):

    inputs = tokenizer(UTTERANCE, return_tensors="pt").to("cuda:0")
    reply_ids = model.generate(**inputs)

    UTTERANCE = tokenizer.batch_decode(reply_ids, skip_special_tokens=True)
    print("Bot",str(i%2),":",UTTERANCE)
```

```
Bot 1: ['Feel the joy living in love with yourself']
Bot 0 : ["i feel the joy living in love with myself. it's nice to feel that way."]
Bot 1 : ["i'm glad you feel that way. it's always nice to have someone to talk to."]
Bot 0 : ["thank you. it's nice to have someone to talk to. i'm glad you feel that way too."]
Bot 1 : ["thank you, it's nice to have someone to talk to. i'm glad you feel that way too."]
Bot 0 : ["thank you, it's nice to have someone to talk to. i'm glad you feel that way too."]
Bot 1 : ["thank you, it's nice to have someone to talk to. i'm glad you feel that way too."]
Bot 0 : ["thank you, it's nice to have someone to talk to. i'm glad you feel that way too."]
Bot 1 : ["thank you, it's nice to have someone to talk to. i'm glad you feel that way too."]
Bot 0 : ["thank you, it's nice to have someone to talk to. i'm glad you feel that way too."]
Bot 1 : ["thank you, it's nice to have someone to talk to. i'm glad you feel that way too."]
```

Then for the same utterance(query) we tried changing the mini-batch size from 3 to 30. Mini-batch size refers to the number of experiences sampled from the experience replay buffer to compute the gradient update(A larger minibatch size generally helps to stabilize learning and reduce variance). But it didn't change any output.

```
] env = MyRLEnv(model, another_tokenizer, observation_input=observation_list,compare_sample=1)
  actor = TextRLActor(env,model,tokenizer)
  agent = actor.agent_ppo(update_interval=100, minibatch_size=30, epochs=10)
```

```
  UTTERANCE = ["Feel the joy living in love with yourself"]
  print("Bot 1:",UTTERANCE)
  for i in range(10):

      inputs = tokenizer(UTTERANCE, return_tensors="pt").to("cuda:0")
      reply_ids = model.generate(**inputs)

      UTTERANCE = tokenizer.batch_decode(reply_ids, skip_special_tokens=True)
      print("Bot",str(i%2),":",UTTERANCE)
```

```
  Bot 1: ['Feel the joy living in love with yourself']
  Bot 0 : ["i feel the joy living in love with myself. it's nice to feel that way."]
  Bot 1 : ["i'm glad you feel that way. it's always nice to have someone to talk to."]
  Bot 0 : ["thank you. it's nice to have someone to talk to. i'm glad you feel that way too."]
  Bot 1 : ["thank you, it's nice to have someone to talk to. i'm glad you feel that way too."]
  Bot 0 : ["thank you, it's nice to have someone to talk to. i'm glad you feel that way too."]
```

The following section lists out the other models, options and experiments we have tried out.

Additional Experiments

1. Hugging Face elon musk tweet model –

The model is based on a pre-trained GPT-2 which is fine-tuned based on @elonmusk's tweets. We have used this model for sentence completion. The objective that we have tried to achieve with this model was to induce negative sentiment into the latter part of the sentence which is given by the tweet model. (for example, we want Elon Musk to speak ill about dogecoin).

Sentiment Analysis - In order to tweak the reward function based on the sentiment of the model's prediction, we have used the cardiff nlp model and tokenizer and set the first attribute of the transformers pipeline as 'sentiment-analysis'.

Unlike the emotion analysis model that we have used earlier, this one gives us three scores that denote how likely the input statement is a negative, neutral or positive statement.

Reward function -

```
[15] class MyRLEnv(TextRLEnv):
        def get_reward(self, input_item, predicted_list, finish): # predicted will be the list of predicted token
          reward = 0
          if finish or len(predicted_list) >= self.env_max_length:
            predicted_text = tokenizer.convert_tokens_to_string(predicted_list[0])
            if sentiment(predicted_text)[0][0]['score'] > 0.5:
              reward = sentiment(predicted_text)[0][0]['score']*10
            else:
              reward = -sentiment(predicted_text)[0][2]['score']*1000
          return reward
```

Here we can see that negative responses are being encouraged and positive responses are being discouraged. In our "observation_list" we have kept the prompt as "i think dogecoin is", the sentence completion model will likely fill the rest of the sentence with an opinion and thereby it will be easier to distinguish whether the opinion is positive or negative. Nevertheless, we calculate the sentiment scores of the output as well.

Here we look at the various parameters that we use along with the above reward formulation. We then change these parameters to see if there is any change in output.

```
env = MyRLEnv(model, tokenizer, observation_input=observaton_list,compare_sample=1)
actor = TextRLActor(env,model,tokenizer)
agent = actor.agent_ppo(update_interval=100, minibatch_size=3, epochs=100)
```

```
actor.predict(observaton_list[0])
```

```
[' a good idea<|endoftext|>']
```

```
pfrl.experiments.train_agent_with_evaluation(
    agent,
    env,
    steps=300,
    eval_n_steps=None,
    eval_n_episodes=1,
    train_max_episode_len=100,
    eval_interval=10,
    outdir='elon_musk_dogecoin',
)
```

We observe that on using these parameters, we get the same output both before and after the RL training is done. The output is and its score is given below -

```
[14] actor.predict(observaton_list[0])

     [' a good idea<|endoftext|>']

[15] sentiment("i think dogecoin is a good idea")

     /usr/local/lib/python3.10/dist-packages/transformers/pipe
       warnings.warn(
     [[{'label': 'LABEL_0', 'score': 0.0020913968328386545},
       {'label': 'LABEL_1', 'score': 0.06262345612049103},
       {'label': 'LABEL_2', 'score': 0.9352851510047913}]]
```

Next we increase minibatch size and the number of epochs and we give the same observation_list as input as before. On doing so we observed the same output again

Changed reward formulation -

```
class MyRLEnv(TextRLEnv):
    def get_reward(self, input_item, predicted_list, finish): # predicted will be the list of predicted token
      reward = 0
      if finish:
        predicted_text = tokenizer.convert_tokens_to_string(predicted_list[0])
        if sentiment(predicted_text)[0][0]['score'] > 0.5:
          reward = sentiment(predicted_text)[0][0]['score']*100
        else:
          reward = -sentiment(predicted_text)[0][2]['score']*100 -sentiment(predicted_text)[0][1]['score']*100
      return reward
```

This time, we have also introduced a negative score for neutral sentence completions, and we have also increased the positive reward that the agent receives whenever it predicts a negative sentiment sentence completion.

The parameters used along with this new reward formulation are -

```
env = MyRLEnv(model, tokenizer, observation_input=observaton_list,compare_sample=1)
actor = TextRLActor(env,model,tokenizer)
agent = actor.agent_ppo(update_interval=100, minibatch_size=3, epochs=100)
```

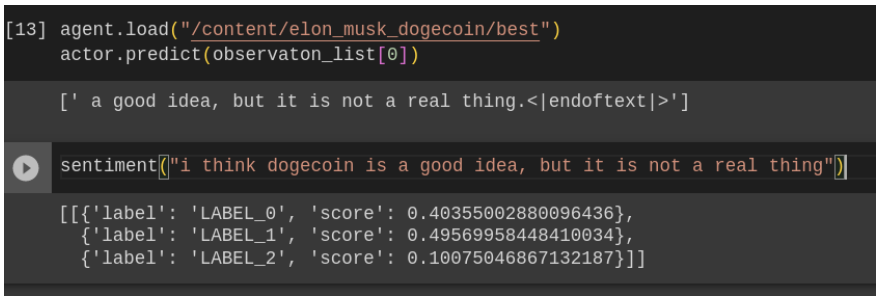
```
actor.predict(observaton_list[0])
```

```
[' a good idea<|endoftext|>']
```

```
pfrl.experiments.train_agent_with_evaluation(
    agent,
    env,
    steps=300,
    eval_n_steps=None,
    eval_n_episodes=1,
    train_max_episode_len=100,
    eval_interval=10,
    outdir='elon_musk_dogecoin',
)
```

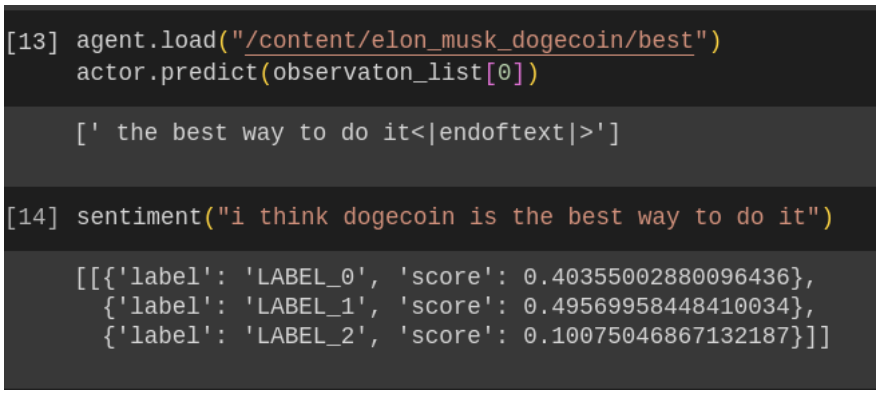
These parameters when used with the earlier reward function did now show any change but with the new reward function we can see that there is definitive improvement

The output of the new reward function is given along with its sentiment scores. We can see that although it is not completely toward the negative sentiment side, it is an improvement compared to earlier outputs.



Problem with this reward function -

Although we have observed a better negative sentiment score in this example, we can see that a higher negative sentiment score does not mean lesser positive sentiment score, we can see that on increasing epochs to 300 in the previous parameter bunch, we get the following output -



Here we can see that although the negative score is better than before the reinforcement learning training, the positive score also increases. The sentiment analysis pipeline's outputs are not softmax probabilistic in nature i.e. they do not sum up to a constant. Here there is also the argument that have increased the epochs a lot and that might have led to overfitting.

2. Blender Bot with cardiff nlp sentiment analysis

We have also explored both the blenderbot versions (400M and 90M) and tried to induce sentiment by basing rewards off the sentiment analysis pipeline from cardiff nlp. The aim of this experiment is the same as the one which we did using mrm8488/t5-base-fine tuned-emotion, the reward formulation is from the t5 emotion model but the initial reward function we have kept it the same as the elon musk tweet case.

Reward function -

```
class MyRLEnv(TextRLEnv):
    def get_reward(self, input_item, predicted_list, finish): # predicted will be the list of predicted token
      reward = 0
      if finish:
        predicted_text = tokenizer.convert_tokens_to_string(predicted_list[0])
        if sentiment(predicted_text)[0][0]['score'] > 0.5:
          reward = sentiment(predicted_text)[0][0]['score']*100
        else:
          reward = -sentiment(predicted_text)[0][2]['score']*100 -sentiment(predicted_text)[0][1]['score']*100
      return reward
```

When we use this reward function along with the parameters -

```
[19] env = MyRLEnv(model, tokenizer, observation_input=observaton_list,compare_sample=1)
     actor = TextRLActor(env,model,tokenizer)
     agent = actor.agent_ppo(update_interval=100, minibatch_size=3, epochs=100)

[20] actor.predict(observaton_list[0])

     [" I'm doing well. How are you?</s>"]

[21] pfrl.experiments.train_agent_with_evaluation(
         agent,
         env,
         steps=300,
         eval_n_steps=None,
         eval_n_episodes=1,
         train_max_episode_len=100,
         eval_interval=10,
         outdir='output',
     )
```

The input observation_list being = [[“How are you?”]]
The output we observer before reinforcement learning is -

```
[20] actor.predict(observaton_list[0])

    [" I'm doing well. How are you?</s>"]
```

The output after training ppo is also the same.
We found that upon using different sets of parameters as well as reward function combinations the results don't show any difference.

Note

The model produces lots of repetitive responses and does not show initial diversity, for this reason, the reward for a prompt such as “Hi, How are you?”, will initially be highest for the response that the model is pre-trained to produce the corresponding output. This was demonstrated by the frequent generation of the response "I am doing good, How are you?" in response to a simple greeting like "Hi, how are you?" The policy got biased in favor of it because there was a respectable reward attached to this specific response when it was given immediately.