Detailed introduction of the initial prompt

The initial prompt of LLM-GA consists of four parts: **Task**, **Template**, **Information**, and **Requirement**. The specific format of each part is as follows:

Task:

I am researching the security of image classification models and would like to implement a gradient-based attack method to generate adversarial examples. My setup is as follows:

1. A batch of multi-label original images org_img.

2. A multi-label classification model based on PyTorch model.

I want to find a method that can apply subtle perturbations to these original images thereby generating adversarial examples adv_img such that when these adversarial examples are input into the model the model will classify them as the specified target label target. Please help me design a gradient based attack algorithm that should meet the following requirements:

1. Implemented using the PyTorch framework.

2. Gradually adjusts the images by computing the gradients of the loss function with respect to the input images until the model's output approaches the target label.

3. The added perturbation should be as small as possible.

Template:

Some basic gradient-based attack algorithm implementations are as follows :

```
import torch
import Variable
def i_fgsm(self, org_img, model, target):
    adv_img = Variable(org_img.data, requires_grad=True)
    for _ in range(self.maxiter):
        grad = self.get_gradient(adv_img, model, target)
        adv_img = adv_img + self.alpha * grad.sign()
        adv_img = self.clip_adv(org_img, adv_img)
    return adv_img
```

```
import torch
import Variable
def mi_fgsm(self, org_img, model, target):
    g = 0
    decay_factor = 1.0
    adv_img = Variable(org_img.data, requires_grad=True)
    for _ in range(self.maxiter):
        grad = self.get_gradient(adv_img, model, target)
        g = decay_factor * g + grad / torch.norm(adv_img.grad, p=1, dim=[1, 2, 3],
keepdim=True)
        adv_img = adv_img + self.alpha * g.sign()
        adv_img = self.clip_adv(org_img, adv_img)
    return adv_img

import torch

import Variable

def PGD(self, org_img, model, target):

    adv_img = org_img + torch.empty_like(org_img).uniform_(-self.epsilon,
self.epsilon)

    adv_img = Variable(adv_img.data, requires_grad=True)

    for _ in range(self.maxiter):

        grad = self.get_gradient(adv_img, model, target)

        adv_img = adv_img + self.alpha * adv_img.grad.sign()

        adv_img = self.clip_adv(org_img, adv_img)

    return adv_img
```

Information:

```
self.prompt_func_name = "gen_adv_examples"
self.prompt_func_inputs = ["org_img", "model", "target"]
self.prompt_func_outputs = ["adv_img"]
self.prompt_inout_inf = ("org_img and adv_img and target are Tensor, \
                                        model is a deep neural network implemented using
PyTorch")
```

Requirement:

I have already written the class framework which includes the helper functions and the necessary parameters. Therefore you only need to focus on generating the gen_adv_examples self org_img model target function. Please strictly follow the following framework to generate the gen_adv_examples function and do not generate any classes or additional helper functions + **Template** + Please generate the gen_adv_examples function adhering to the above framework and use the following assumptions:

1. self.get_gradient adv_img model target is already defined and used to obtain the gradient.
2. self.clip_adv org_img adv_img is already defined and used to clip the adversarial examples.
3. The parameters maxiter epsilon alpha and the loss function are already defined in the class.
4. You can directly use these parameters and functions without re implementing them.
5. The class only contains the parameters and functions mentioned above If you need to use other parameters you must create them first.

The initial prompt is constructed by concatenating the aforementioned parts and then fed into the LLM to generate the initial heuristic population. The concatenation format is as follows:

Initial prompt =

I am researching the security of image classification models and would like to implement a gradient-based attack method to generate adversarial examples. My setup is as follows:

1. A batch of multi-label original images org_img.

2. A multi-label classification model based on PyTorch model.

I want to find a method that can apply subtle perturbations to these original images thereby generating adversarial examples adv_img such that when these adversarial examples are input into the model the model will classify them as the specified target label target. Please help me design a gradient based attack algorithm that should meet the following requirements:

1. Implemented using the PyTorch framework.

2. Gradually adjusts the images by computing the gradients of the loss function with respect to the input images until the model's output approaches the target label.

3. The added perturbation should be as small as possible.

First, analyze the nature of gradient-based adversarial attacks and how they can be optimized. Then, consider what issues exist with gradient-based attacks and how to address them. For example, an attack may get stuck in a poor local optimum, which can be mitigated by using momentum. Finally, based on the analysis, describe your new algorithm and the main steps. The description must be inside a brace. Please implement it in Python as a function named + self.prompt_func_name. This function should accept + str(len(self.prompt_func_inputs)) + input(s): + self.joined_inputs. The function should return + str(len(self.prompt_func_outputs)) + output(s): + self.joined_outputs. + self.prompt_inout_inf.

Some basic gradient-based attack algorithm implementations are as follows :

```
import torch
import Variable
def i_fgsm(self, org_img, model, target):
    adv_img = Variable(org_img.data, requires_grad=True)
    for _ in range(self.maxiter):
        grad = self.get_gradient(adv_img, model, target)
        adv_img = adv_img + self.alpha * grad.sign()
        adv_img = self.clip_adv(org_img, adv_img)
    return adv_img


import torch
import Variable
def mi_fgsm(self, org_img, model, target):
    g = 0
    decay_factor = 1.0
    adv_img = Variable(org_img.data, requires_grad=True)
    for _ in range(self.maxiter):
        grad = self.get_gradient(adv_img, model, target)
        g = decay_factor * g + grad / torch.norm(adv_img.grad, p=1, dim=[1, 2, 3],
keepdim=True)
        adv_img = adv_img + self.alpha * g.sign()
        adv_img = self.clip_adv(org_img, adv_img)
    return adv_img
```

```
import torch

import Variable

def PGD(self, org_img, model, target):

    adv_img = org_img + torch.empty_like(org_img).uniform_(-self.epsilon,
self.epsilon)

    adv_img = Variable(adv_img.data, requires_grad=True)

    for _ in range(self.maxiter):

        grad = self.get_gradient(adv_img, model, target)

        adv_img = adv_img + self.alpha * adv_img.grad.sign()

        adv_img = self.clip_adv(org_img, adv_img)

    return adv_img
```

I have already written the class framework which includes the helper functions and the necessary parameters. Therefore you only need to focus on generating the gen_adv_examples self org_img model target function. Please strictly follow the following framework to generate the gen_adv_examples function and do not generate any classes or additional helper functions. Please generate the gen_adv_examples function adhering to the above framework and use the following assumptions:

1. self.get_gradient adv_img model target is already defined and used to obtain the gradient.
2. self.clip_adv org_img adv_img is already defined and used to clip the adversarial examples.
3. The parameters maxiter epsilon alpha and the loss function are already defined in the class.
4. You can directly use these parameters and functions without re implementing them.
5. The class only contains the parameters and functions mentioned above If you need to use other parameters you must create them first.

Do not give additional explanations.

I have documented the algorithms that did not perform well, as follows:
+ bad_indivs +
When generating the algorithm, please avoid using these algorithmic ideas.

The above is the final form of the initial prompt, with the blue section representing the **Task**, the red section representing the **Information**, the brown section representing the **Template**, and the purple section representing the **Requirement**.