CS523: Deep Learning

ImagePrompt: Discrete Prompt Optimization for Image Generation using Policy Gradient Methods

Group Members:

Reana Naik

Sandesh Bharadwaj

Pruthvi Bharadwaj

Boston University

Overview

- Motivation & Problem Formulation
- Architecture
- Methods
- Results

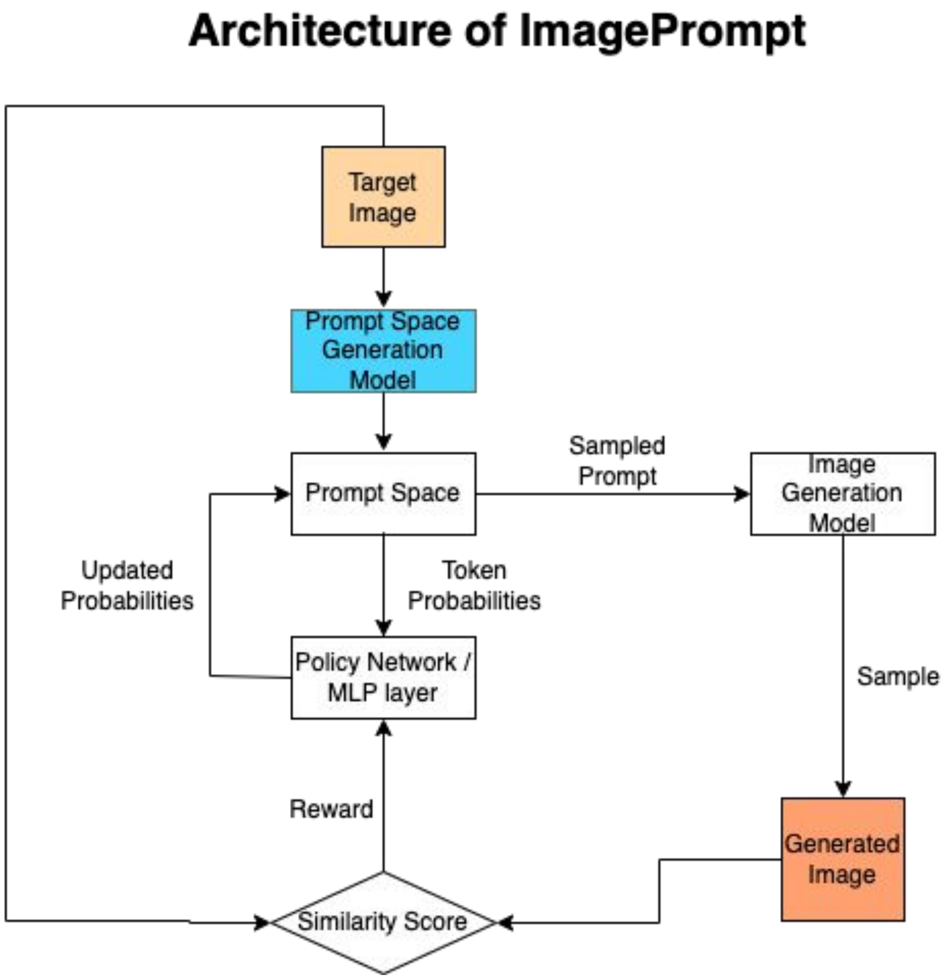
Motivation & Problem Formulation

- ➔ Forensic artists can be replaced with generative AI models
- ➔ Prompt engineering to get the generated image to match the target image
- ➔ Prompt Engineering - Generating desired prompts as required for text-to-image and language models.
  - ❖ Time-consuming
  - ❖ Labor-intensive
  - ❖ Size of optimal prompts for images can vary based on image.
- ➔ Image captioning - Generating a human-readable text description of a given image.

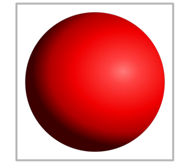
Objective:

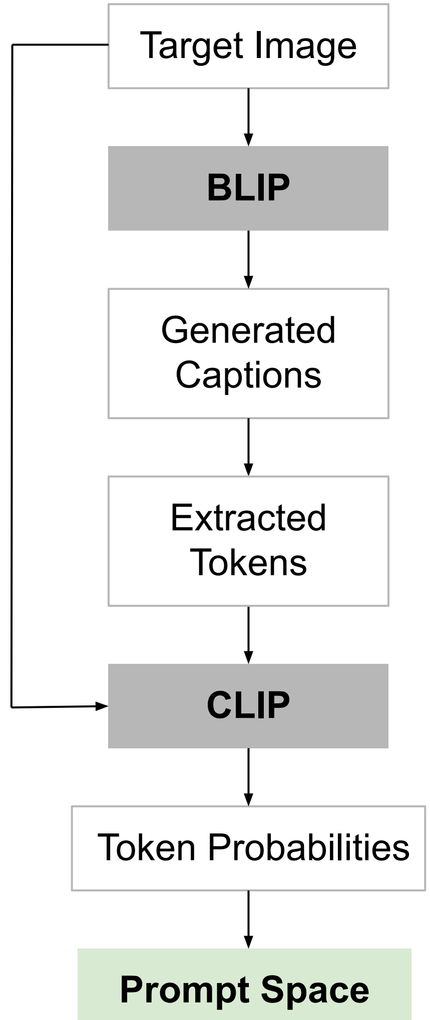
- ➔ Generate an optimal prompt from captions generated for an input image using reinforcement learning, such that the prompt generates an image with maximum similarity to the input image.

Methods: *ImagePrompt Architecture*



Methods: *Generating the Prompt Space*





BLIP generates captions for input image.

Extract unique tokens from all generated captions.

"a 3D rendering of a sphere",
"a red ball",
"an egg, which is red"

'a', '3D', 'rendering', 'of', 'sphere', 'red', 'ball', 'an', 'egg', 'which', 'is'

CLIP extracts token and image features for every token and input image.

Calculate similarity between image and token features to get token probabilities

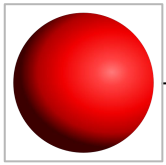
Choose set number of tokens with highest probabilities.

'sphere', 'red', 'ball', 'egg'

Methods: *Policy Optimization Task*

Prompt Space

'sphere', 'red', 'ball', 'egg'



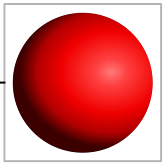
CLIP

Policy, π

P('sphere'), P('red'), P('ball'), P('egg')

SPHERE	**RED**	**BALL**	**EGG**
P = 0.238	P = 0.262	P = 0.261	P = 0.238

Prompt: **RED** **BALL**



68% match!

(Is this value maximized?)

We need to optimize the TOKEN PROBABILITIES!

Methods: *Policy-Gradient Method (REINFORCE)*

REINFORCE is a policy-gradient method that learns the optimal policy directly (on-policy). The parameter being updated are the weights θ of the neural network.

Instead of calculating the reward after every episode, we generate a *trajectory*:

$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \ldots, a_k, r_{k+1}, s_{k+1}, \ldots)$$

A trajectory consists of all the transitional states, actions and rewards in each episode.

We calculate the *return for a trajectory* τ as *R(*τ*)*:

$$R(\tau) = G_1 + G_2 + \ldots + G_k$$

where G_k is the future return at time step k for a transition episode k

$$(s_k, a_k, r_{k+1})$$

We optimize θ by maximizing the expected return U(θ) defined as:

$$U(\theta) = \sum_\tau P(\tau;\theta) \cdot R(\tau)$$

where P(τ; θ) is the probability of each possible trajectory.

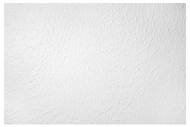
Methods: *Collecting Trajectory Data*

1	2	3	4
SPHERE	**RED**	**BALL**	**EGG**
P = 0.238	P = 0.262	P = 0.261	P = 0.238

Prompt: RED ________ ________

Token History:

Reward History:

State History: 

$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \ldots, a_k, r_{k+1}, s_{k+1}, \ldots)$$

Methods: *Collecting Trajectory Data*

1	2	3	4
SPHERE	**RED**	**BALL**	**EGG**
P = 0.238	P = 0.262	P = 0.261	P = 0.238

Prompt: RED ______ SPHERE ______

Token History: [2]

Reward History: [0.49]

State History:  

$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \ldots, a_k, r_{k+1}, s_{k+1}, \ldots)$$

Methods: *Collecting Trajectory Data*

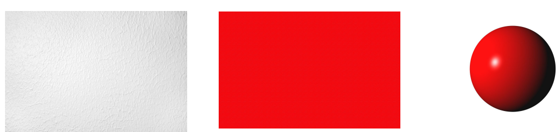
1	2	3	4
SPHERE	**RED**	**BALL**	**EGG**
P = 0.238	P = 0.262	P = 0.261	P = 0.238

Prompt: RED SPHERE

Token History: [2, 1]

Reward History: [0.49, 0.89]

State History:



$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \ldots, a_k, r_{k+1}, s_{k+1}, \ldots)$$

Methods: *Collecting Trajectory Data*

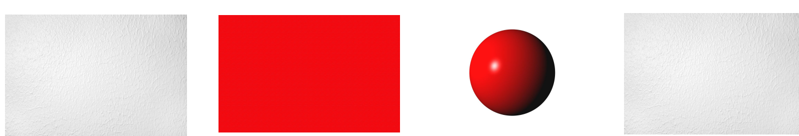
1	2	3	4
SPHERE	**RED**	**BALL**	**EGG**
P = 0.238	P = 0.262	P = 0.261	P = 0.238

Prompt: EGG ______ ______

Token History: [2, 1]

Reward History: [0.49, 0.89]

State History:



$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \ldots, a_k, r_{k+1}, s_{k+1}, \ldots)$$

Methods: *Collecting Trajectory Data*

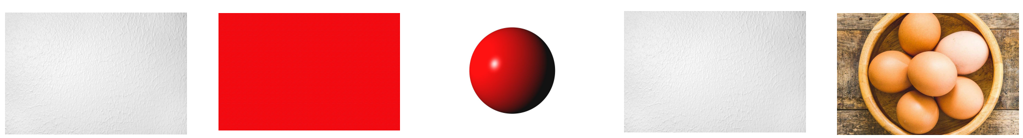
1	2	3	4
SPHERE	**RED**	**BALL**	**EGG**
P = 0.238	P = 0.262	P = 0.261	P = 0.238

Prompt: EGG ______ RED ______

Token History: [2, 1, 4]

Reward History: [0.49, 0.89, 0.43,]

State History:



$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \ldots, a_k, r_{k+1}, s_{k+1}, \ldots)$$

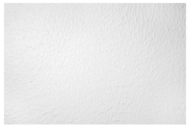
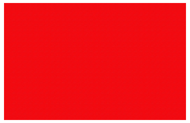
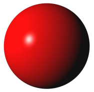
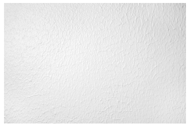
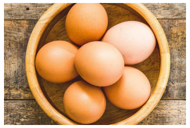
Methods: *Collecting Trajectory Data*

1	2	3	4
SPHERE	**RED**	**BALL**	**EGG**
P = 0.238	P = 0.262	P = 0.261	P = 0.238

Prompt: EGG RED

Token History: [2, 1, 4, 2]

Reward History: [0.49, 0.89, 0.43, 0.91]

State History:      

$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \ldots, a_k, r_{k+1}, s_{k+1}, \ldots)$$

Methods: *Collecting Trajectory Data*

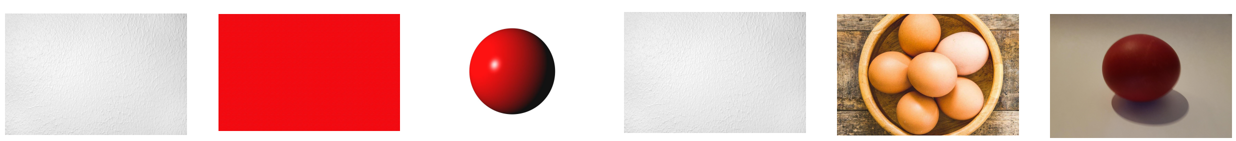
1	2	3	4
SPHERE	**RED**	**BALL**	**EGG**
P = 0.238	P = 0.262	P = 0.261	P = 0.238

Prompt: EGG ____ RED ____

Token History: [2, 1, 4, 2]

Reward History: [0.49, 0.89, 0.43, 0.91]

State History:



Future Returns: [0.49, 1.38, 0.92, 1.34]

$$R(\tau) = G_1 + G_2 + \ldots + G_k$$

$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \ldots, a_k, r_{k+1}, s_{k+1}, \ldots)$$

Methods: *Policy Network Training*

```
model = nn.Linear(512, 4)
optimizer = torch.optim.Adam(net_parameters(), lr=lr)

for _ in range(n_iterations):

observations, actions, future_returns = …

logits = net(observations)
policy_distributions =
torch.distributions.Categorical(logits=logits)
log_probs = policy_distributions.log_probs(actions)

mean = future_returns.mean()
std = future_returns.std().clamp_min(1e-12)
normalized_future_returns = (future_returns -
mean)/std

loss = -(log_probs * normalized_returns).mean()

net.zero_grad()
loss.backward()
optimizer.step()
```

Initialize the policy network
Choose optimizer as Adam

Collect trajectory data for multiple episodes

Compute the policy distribution for each observation and the log probabilities of each action as determined by the policy distribution.

Normalize future returns to reduce variance

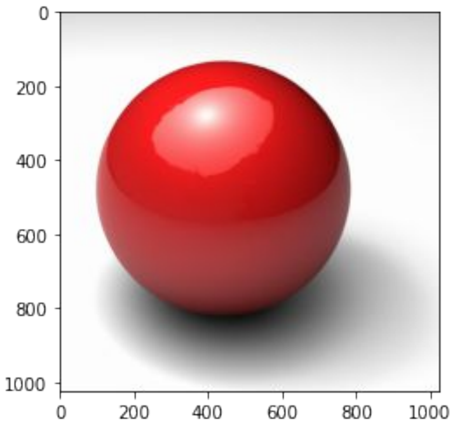
Loss minimization

Zero the gradients, propagate the error backwards and update θ

$$U(\theta) = \sum_\tau P(\tau;\theta) \cdot R(\tau)$$

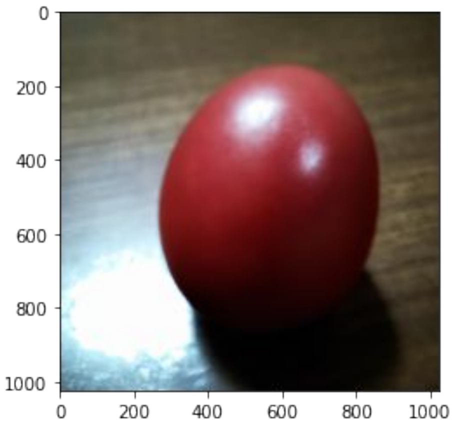
Results: *Prompt Generation with Target Image Only (Policy Gradient Method)*

```
# Hyperparameters
n_captions = 5, n_tokens = 4
input_size = 512, output_size = 4, lr = 0.001
max_episodes = 10, n_iterations = 40
prompt_length = 2


Prompt Space:  ['sphere', 'egg', 'ball', 'red']
Prompt Space Probabilities:  [0.23805307 0.23905936 0.2610057
0.26188195]

Generated Prompt: red sphere
Similarity Score:  0.93471456
```



Results: *Prompt Completion with Target Image and Partial Prompt (Policy Gradient Method)*

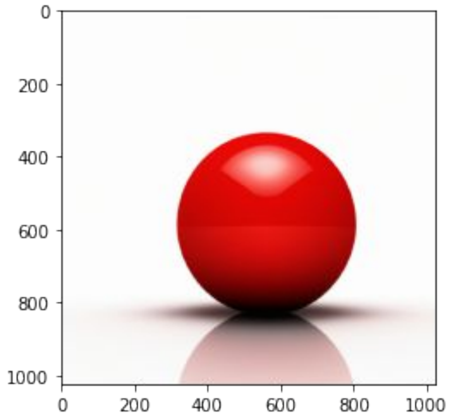
```
# Hyperparameters
n_captions = 5, n_tokens = 4
input_size = 512, output_size = 4, lr = 0.001
max_episodes = 20, n_iterations = 20
prompt_length = 2, init_prompt = 'a red'


Prompt Space:  ['sphere', 'egg', 'ball', 'red']
Prompt Space Probabilities:  [0.23805307 0.23905936 0.2610057
0.26188195]

Generated Prompt: a red egg
Similarity Score:  0.8069409
```



Results: *Prompt Generation with Initial CLIP Probabilities*

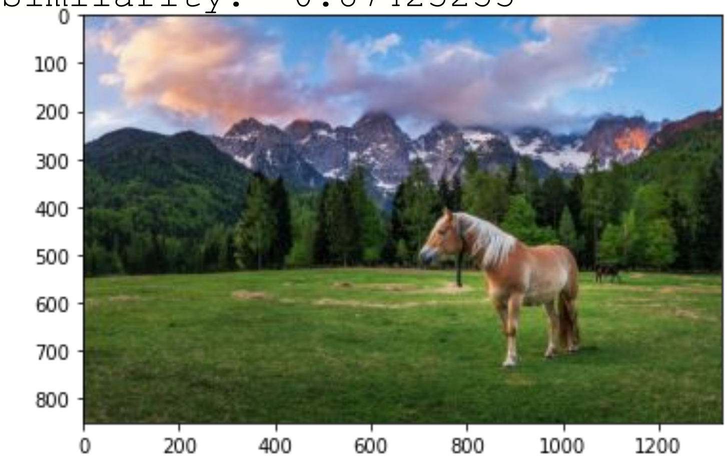
```
# Hyperparameters
n_captions = 5, n_tokens = 4

prompt_length = 2
```

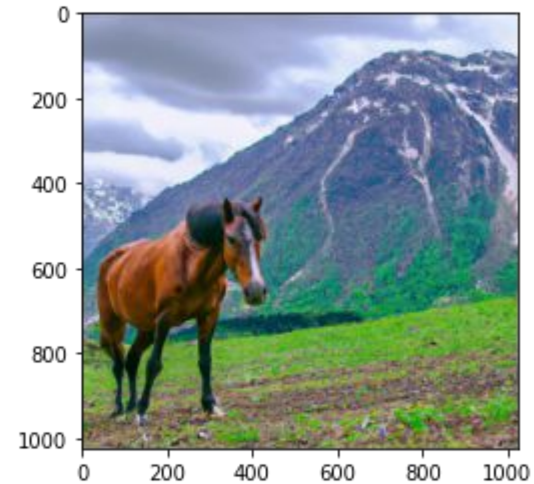
```
Prompt Space:  ['sphere', 'egg', 'ball', 'red']
Prompt Space Probabilities:  [0.23805307 0.23905936 0.2610057
0.26188195]

Generated Prompt: red ball
Similarity Score: 0.9255184
```



Results: *Prompt Generation with Target Image Only (Policy Gradient Method)*

```
# Hyperparameters
n_captions = 8, n_tokens = 7, max_episodes = 10, n_iterations = 20
prompt_length = 4
Prompt Space:
['the','peaks','mountain','spain','horse','mountains','horses']
Prompt Space Probabilities:
[0.1266768 0.12865311 0.1397564 0.1431053 0.15863107 0.14128713 0.16189025]

Best Prompt:  mountain horse mountains field
Similarity:  0.87423253
```

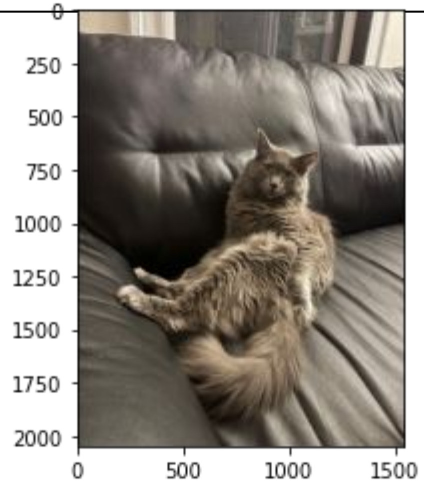




Boston University

Results: *Prompt Generation with Target Image Only (Policy Gradient Method)*

```
# Hyperparameters
n_captions = 7, n_tokens = 8, max_episodes = 10, n_iterations = 20
prompt_length = 4
Prompt Space:  ['of', 'is', 'sitting', 'sofa', 'couch', 'grey', 'long', 'cat']
Prompt Space Probabilities: [0.11573587 0.11826485 0.1274537 0.13060148 0.13342977
0.11866169 0.12265412 0.13319854]

Best prompt:
Similarity:
```



Limitations of ImagePrompt

- Relies on the performance and computation speed of other pre-trained models.
- Server on which DALL-E runs is unstable at times and crashes during training.
- DALL-E image generation is slow so training is slow.
- The model would have to be trained for every brand new image.

Future Works

- Train the model for more iterations and evaluate performance.
- Testing larger prompt sizes with more complex images.

Thank you!

Questions?