HOMEWORK 4
MULTI-MODAL FOUNDATION MODELS [*]

10-423/10-623 GENERATIVE AI
http://423.mlcourse.org

OUT: Mar. 13, 2024
DUE: Mar. 22, 2024
TAs: Asmita, Haoyang, Tiancheng

Instructions

- **Collaboration Policy**: Please read the collaboration policy in the syllabus.
- **Late Submission Policy:** See the late submission policy in the syllabus.
- **Submitting your work:** You will use Gradescope to submit answers to all questions and code.
  - **Written:** You will submit your completed homework as a PDF to Gradescope. Please use the provided template. Submissions can be handwritten, but must be clearly legible; otherwise, you will not be awarded marks. Alternatively, submissions can be written in LaTeX. Each answer should be within the box provided. If you do not follow the template, your assignment may not be graded correctly by our AI assisted grader and there will be a **2% penalty** (e.g., if the homework is out of 100 points, 2 points will be deducted from your final score).
  - **Programming:** You will submit your code for programming questions to Gradescope. There is no autograder. We will examine your code by hand and may award marks for its submission.
- **Materials:** The data that you will need in order to complete this assignment is posted along with the writeup and template on the course website.

Question	Points
Instruction Fine-Tuning & RLHF	9
Latent Diffusion Model (LDM)	6
Programming: Prompt2Prompt	31
Code Upload	0
Collaboration Questions	2
Total:	48

[*] Compiled on Wednesday 13th March, 2024 at 17:17

1 Instruction Fine-Tuning & RLHF (9 points)

1.1. (6 points) **Short answer:** Highlight the differences between in-context learning, unsupervised pre-training, supervised fine-tuning, and instruction fine-tuning by defining each one. Assume we are interested specifically in autoregressive large language models (LLMs) over text. Each definition must mention properties of the training examples and how they are used, and how learning affects the parameters of the model.

Definition: in-context learning

Definition: unsupervised pre-training

Definition: supervised fine-tuning

Definition: instruction fine-tuning

1.2. (3 points) **Ordering:** Consider a correctly defined reinforcement learning with human feedback (RLHF) pipeline. *Select the correct ordering of the items below to define such a pipeline by numbering them from 1 to N. If two items can occur simultaneously, number them identically. To exclude an item from the ordering, number it as* 0*.*

- ☐ Repeat the previous step many times.
- ☐ Repeat the following steps many times.
- ☐ From human labelers, collect rankings of samples from the language model.
- ☐ Collect instruction fine-tuning training examples from human labelers.
- ☐ Take a (stochastic) gradient step for a reinforcement learning objective.
- ☐ Sample a prompt/response pair from the language model.
- ☐ Collect prompt/response/reward tuples from human labelers.
- ☐ Perform supervised fine-tuning of the language model.
- ☐ Query the regression model for its score of an input.
- ☐ Perform supervised training of the regression model.
- ☐ Pre-train the language model.

2 Latent Diffusion Model (LDM) (6 points)

2.1. (2 points) **Short answer:** Why does a latent diffusion model run diffusion in a latent space instead of pixel space?

2.2. **Short answer:** Standard cross-attention for a diffusion-based text-to-image model defines the queries $\mathbf{Q}$ as a function of the pixels (or latent space) $\mathbf{Y} \in \mathbb{R}^{m \times d_y}$, and the keys $\mathbf{K}$ and values $\mathbf{V}$ as a function of the text encoder output $\mathbf{X} \in \mathbb{R}^{n \times d_x}$.

$$\mathbf{Q} = \mathbf{Y}\mathbf{W}_q, \qquad \mathbf{K} = \mathbf{X}\mathbf{W}_k, \qquad \mathbf{V} = \mathbf{X}\mathbf{W}_v$$

(where $\mathbf{W}_q \in \mathbb{R}^{d_y \times d}$ and $\mathbf{W}_k, \mathbf{W}_v \in \mathbb{R}^{d_x \times d}$) and then applies standard attention:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\mathbf{Q}\mathbf{K}^T/\sqrt{d})\mathbf{V}$$

Now, suppose you instead defined a new formulation where the values are a function of the pixels (or latent space): $\mathbf{V} = \mathbf{Y}\mathbf{W}_v$ where $\mathbf{W}_v \in \mathbb{R}^{d_y \times d}$.

2.2.a. (2 points) What goes wrong mathematically in the new formulation?

2.2.b. (2 points) Intuitively, why doesn't the new formulation make sense? Briefly begin with an explanation of what the original formulation of cross-attention is trying to accomplish for a single query vector, and why this new formulation fails to accomplish that.

3 Programming: Prompt2Prompt (31 points)

Introduction

In this section, we explore an innovative approach to image editing. Editing techniques aim to retain the majority of the original image's content while making certain changes. However, current text-to-image models often produce completely different images when only a minor change to the prompt is made. State-of-the-art methods typically require a spatial mask to indicate the modification area, which ignores the original image's structure and content in that region, resulting in significant information loss.

In contrast, the Prompt2Prompt framework by Hertz et al. (2022) facilitates edits using only text, striving to preserve original image elements while allowing for changes in specific areas.

Cross-attention maps, which are high-dimensional tensors binding pixels with prompt text tokens, hold rich semantic relationships crucial to image generation. The key idea is to edit the image by injecting these maps into the diffusion process. This method controls which pixels relate to which particular prompt text tokens throughout the diffusion steps, allowing for targeted image modifications.

You'll explore modifying token values to change scene elements (e.g. a "dog" riding a bicycle $\rightarrow$ a "cat" riding a bicycle) while maintaining the original cross-attention maps to keep the scene's layout intact.

HuggingFace Diffusers

In this assignment, we will be using HuggingFace's diffusers, a library created for easily using well-known state-of-the-art diffusion models, including creating the model classes, loading pre-trained weights, and calling specific parts of the models for inference. Specifically, we will be using the API for the class `DiffusionPipeline` and methods from its subclass `StableDiffusionPipeline` for loading the pre-trained LDM model.

You are required to read the API for StableDiffusionPipeline:

`https://huggingface.co/docs/diffusers/en/api/pipelines/stable_diffusion/text2img`

You will be implementing the model loading and calling individual components of StableDiffusion-Pipeline in this assignment.

Starter Code

The files are organized as follows:

```
hw4/
   run_in_colab.ipynb
   prompt2prompt.py
   ptp_utils.py
   seq_aligner.py
   requirements.txt
```

Here is what you will find in each file:

1. `run_in_colab.ipynb`: This is where you can run inference and see the visualization of your implemented methods.

2. `prompt2prompt.py`: Contains the `text2image_ldm(...)` method that generates images from text prompts by controlling the diffusion process with attention mechanisms in HuggingFace's latent diffusion model, and contains the `AttentionReplace` class. The class contains the forward process and methods to replace attention. You will implement all these. (Note: Locations in the code where changes ought to be made are marked with a TODO.)
3. `ptp_utils.py`: Contains a set of helper functions that will be useful to you for filling in the `text2image_ldm(...)` method. Carefully read through the file to understand what these functions are.
4. `seq_aligner.py`: Contains a set of helper functions that are used to initialize `AttentionReplace`'s class variables. You will need to implement `get_replacement_mapper_(...)` (Note: Locations in the code where changes ought to be made are marked with a TODO.)
5. `requirements.txt`: A list of packages that need to be installed for this homework.

Command Line

We recommend conducting your final experiments for this homework on Colab. Colab provides a free T4 GPU for code execution.

```
(Run the run_in_colab.ipynb for visualization.)
```

You may find it easier to implement/debug locally. We have also included a very simple example of visualization that you can run on the command line:

```
python prompt2prompt.py
```

Prompt2Prompt

In this problem, you will implement Prompt2Prompt in the file `prompt2prompt.py`.

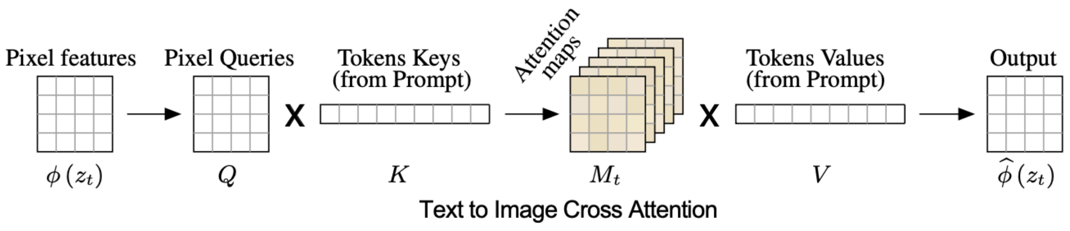

Figure 1: Visual and textual embedding are fused using cross-attention layers that produce attention maps for each textual token. Figure source: Hertz et al. (2022)

Latent Diffusion Model:

You will implement the `text2image_ldm` method. In that method, we provided some suggested structure by giving you the left-hand side of the initializations.

Implementing this method requires you to have already read the HuggingFace Diffusers API. See above. You will be working with the `DiffusionPipeline` type, but the line

`DiffusionPipeline.from_pretrained(model_id)` is actually loading a class of type `StableDiffusionPipeline`.

Here is an overview of the key steps this method performs:

- Attention Control Registration: The function begins by registering an attention control mechanism within the model using the provided controller.
- Tokenization and Embedding of Prompts: The model's tokenizer converts both an empty string (to represent the unconditional generation case) and the actual text prompts into tokenized inputs. These tokenized inputs are then passed through a BERT-like model to obtain embeddings. The embeddings for the unconditional inputs and the text prompts are concatenated to serve as the context for the diffusion process.

  (Important note: the particular text encoder we are using has a maximum length of 77 tokens. You will notice this `max_len` is fixed to 77 in the starter code.)
- Latent Space Initialization: It initializes a latent space with the specified dimensions. This space will evolve into the final image through the diffusion process.
- Diffusion Process: The core of the image generation happens here. For each timestep defined by num_inference_steps, the function performs a diffusion step. This involves manipulating the latent space towards the desired outcome based on the context and the current timestep, under the guidance of the specified scale. The controller plays a role here in directing the attention mechanism during these steps.
- Image Generation: After completing the diffusion steps, the final latent representation is converted into an image using the model's VQ-VAE (Vector Quantized Variational AutoEncoder).

Hint: Some of these steps can be performed simply by utilising the necessary methods from `ptp_utils.py`.

Cross Attention:

The LDM utilizes text prompts to influence the noise prediction at each diffusion step through cross-attention layers. Essentially, at each step t, the model predicts noise ϵ based on a noisy image z_t and the text prompt's embedding $\psi(P)$ using a U-net architecture, leading to the final image $I = z_0$. The key interaction between image and text occurs in the noise prediction phase, where visual and textual embeddings are integrated via cross-attention layers. As illustrated in Fig. 1, these layers generate spatial attention maps for textual tokens by projecting the image's deep features and text embedding into query (Q), key (K), and value (V) matrices through learned projections ℓ_Q, ℓ_K, ℓ_V. The attention mechanism is formulated as:

$$M = \text{Softmax}\left(\frac{QK^T}{\sqrt{d}}\right), \tag{1}$$

where M_{ij} represents the influence of the j-th token's value on the i-th pixel, with d_k being the dimensionality of the keys and queries. The output from cross-attention, $\phi_b(z_t) = MV$, updates the image features $\phi(z_t)$. Intuitively, MV is a weighted average of V based on the attention maps M, which are correlated to the similarity between Q and K. This process leverages multi-head attention to enhance expressiveness, concatenating the outcomes from parallel heads and refining them through an additional linear layer for the final output.

Controlling Cross Attention:

Pixels are more attracted (correlated) to the words that describe them (you will visualize this when you run the notebook). Building on the insight that cross-attention maps dictate the spatial layout and relationship between pixels and their corresponding descriptive words, Prompt2Prompt proposes a method to edit images while maintaining their original structure. By reusing attention maps M from an initial generation with prompt P in a subsequent generation with an altered prompt P^*, we can create an edited image I^* that respects the original image's layout I.

We can define $DM(z_t, P, t, s)$ as the function for a single diffusion step t, outputting a noisy image z_{t-1} and optionally an attention map M_t. We denote $DM(z_t, P, t, s)\{M \leftarrow \hat{M}\}$ to indicate the diffusion step with an externally supplied attention map $\hat{M}$ overriding the attention map M, while maintaining the value matrix V from P. The attention map generated with the edited prompt P^* is M_t^*. The function Edit(M_t, M_t^*, t) represents an editing operation on the attention maps of the original and edited prompts at step t. This general algo is written out in Fig. 2.

```
Algorithm 1: Prompt-to-Prompt image editing
1  Input: A source prompt $\mathcal{P}$, a target prompt $\mathcal{P}^*$, and a random seed $s$.
2  Optional for local editing: $w$ and $w^*$, words in $\mathcal{P}$ and $\mathcal{P}^*$, specifying the editing region.
3  Output: A source image $x_{src}$ and an edited image $x_{dst}$.
4  $z_T \sim N(0, I)$ a unit Gaussian random variable with random seed $s$;
5  $z_T^* \leftarrow z_T$;
6  for $t = T, T-1, \ldots, 1$ do
7      $z_{t-1}, M_t \leftarrow DM(z_t, \mathcal{P}, t, s)$;
8      $M_t^* \leftarrow DM(z_t^*, \mathcal{P}^*, t, s)$;
9      $\widehat{M}_t \leftarrow Edit(M_t, M_t^*, t)$;
10     $z_{t-1}^* \leftarrow DM(z_t^*, \mathcal{P}^*, t, s)\{M \leftarrow \widehat{M}_t\}$;
11     if local then
12         $\alpha \leftarrow B(\overline{M}_{t,w}) \cup B(\overline{M}^*_{t,w^*})$;
13         $z_{t-1}^* \leftarrow (1-\alpha) \odot z_{t-1} + \alpha \odot z_{t-1}^*$;
14     end
15 end
16 Return $(z_0, z_0^*)$
```

Figure 2: Algorithm: Prompt-to-Prompt image editing. Source: Hertz et al. (2022). Note that *local* is always False in our implementation.

Word Swap:

While Prompt-to-Prompt can be used for various different types of edit operations on the prompt, we will focus exclusively on word swapping, e.g., P = "a big bicycle" to P^*= "a big car".

For word swapping, we inject the attention maps of the source image into the generation by the modified prompt. We work with the `AttentionReplace` class, where you will initialize a mapper tensor as `self.mapper`. It is designed to facilitate the replacement of tokens in the cross-attention map and should be used to reassign attention from the old tokens to the new ones (dive into the code base to see what exactly it does and also refer to the section on Replacement Mapper). You will implement:

- `replace_self_attention`: Responsible for replacing the self-attention map of the current step with the base attention map `attn_base` or keeping it unchanged based on the size of the attention map to be replaced `att_replace`. This decision is made by comparing the size of the `att_replace` with a predefined threshold (in this case, 16 ** 2). If the size is smaller, it expands the `attn_base` to match the dimensions of `att_replace`; otherwise, it simply returns `att_replace`.

- `replace_cross_attention`: The cross-attention replacement involves a computation that maps the base attention `attn_base` through a transformation `self.mapper` to produce a new attention map. This transformation aligns the attention from the source domain (tokens from the original prompt) to the target domain (edited image features).

  (Hint: You can accomplish this through careful use of `einsum`!)

- `forward` method: Algorithm is indicated below:

Algorithm 1 Forward method of AttentionReplace class

```
1: if the layer is cross attention layer or the current step is subject to be edited then
2:     Calculate the number of heads $h$
3:     Reshape attn to be the correct shape
4:     Split attn to attn_base and attn_replace
       (attn_base is the attention for reference example and attn_replace is the attention for the remaining examples)
5:     if the layer is cross attention layer then
6:         Edit attn[1:] with replace cross attention method according to the current step's $\alpha$ (indicating whether to replace the attention for that word) of each individual word
7:     else
8:         Edit attn[1:] with replace self attention method
9:     Reshape attn to be the correct shape
   return attn
```

(Hint: To see some examples of how the alphas are constructed, you can run the main at the bottom of `ptp_utils.py`, e.g. `python ptp_utils.py`.)

Replacement Mapper:

In the function `get_replacement_mapper`, we return the stacked PyTorch tensor containing all the mapping matrices, where each matrix corresponds to the mapping from the first prompt to one of the subsequent prompts. It calls upon `get_replacement_mapper_` (which you will implement) that splits both input strings `x` and `y` into words and constructs a mapping matrix of size `max_len` $\times$ `max_len`, with values in $[0, 1]$ indicating the matching between the changing word in the input prompt and the corresponding word in the modification prompt.

(Hint: For most things in PyTorch we avoid for loops, but you needn't do so here. Since this method is only called once during initialization, for loops are fine.)

(Hint: Use the main at the bottom of `seq_aligner.py` to check that your implementation of `get_replacement_mapper_` is behaving as expected, e.g. `python seq_aligner.py`.)

Evaluation:

We ask you to run the notebook to get the visualizations once you complete filling in the needed functions. You will be visualizing replacement edit and local editing results.

Empirical Questions

The questions below refer directly to the section headers of the Colab notebook in `run_in_colab.ipynb`.

3.1. (4 points) Paste the results from the section 'Baseline: Cross-Attention Visualization'

[Expected runtime on Colab T4: 10s]

3.2. (3 points) Briefly explain what the greyscale cross-attention visualization reveals to you about the behavior of the model.

3.3. (4 points) Paste the results from the section ‘Baseline: No Attention Controller’

[Expected runtime on Colab T4: 30s]

3.4. (4 points) Paste the results from the section ‘Prompt-to-Prompt: Word-swap’

[Expected runtime on Colab T4: 30s]

3.5. (1 point) Briefly explain how your results from Question 3.3 differ from your results in Question 3.4?

3.6. (4 points) Paste the results from the section 'Prompt-to-Prompt: Modify Cross-Attention injection'

[Expected runtime on Colab T4: 30s]

3.7. (2 points) How do you your results in Question 3.6 vary as you change the word-specific cross attention parameters?

3.8. (4 points) Paste the results from the section 'Prompt-to-Prompt: Local Edit'

[Expected runtime on Colab T4: 30s]

3.9. (2 points) Intuitively, what do we accomplish by setting `"default_": 1.` and a word specific attention parameter to a much smaller value, e.g. `"lasagne": .2`, in Question 3.8?

3.10. Define your own base prompt and three prompt edits (i.e. something other than the examples provided in the `.ipynb`) and run them through Prompt-to-Prompt.

3.10.a. (1 point) Report the prompts and any hyperparameters that you used.

3.10.b. (2 points) Paste the resulting images below.

4 Code Upload (0 points)

4.1. (0 points) Did you upload your code to the appropriate programming slot on Gradescope?
Hint: The correct answer is 'yes'.

◯ Yes

◯ No

For this homework, you should upload all the code files that contain your new and/or changed code. Files of type `.py` and `.ipynb` are both fine.

5 Collaboration Questions (2 points)

After you have completed all other components of this assignment, report your answers to these questions regarding the collaboration policy. Details of the policy can be found in the syllabus.

5.1. (1 point) Did you collaborate with anyone on this assignment? If so, list their name or Andrew ID and which problems you worked together on.

5.2. (1 point) Did you find or come across code that implements any part of this assignment? If so, include full details.