EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Computer Science Department
Neuro-Cognitive Modeling

Summer Term 2024

Recurrent and Generative Neural Networks

Exercise Sheet 02

Release: December 16, 2024   Deadline: January 20, 2025 (12:00 pm)

General remarks:

- Download the file `exercisesheet05.zip` from the lecture site (ILIAS). This archive contains files (Python scripts), which are required for the exercises.
- All relevant equations can be found in the corresponding lecture slides. Ideally, the exercises should be completed in teams of two students. Larger teams are not allowed.
- **Add a brief documentation (pdf) to your submission**. The documentation should contain your **names**, **matriculation numbers**, and **email adresses** as well as protocols of your experiments, parameter choices, and a discussion of the results.
- Set up a Conda environment using the provided `environment.yml` file (included in the `exercisesheet05.zip` archive).
- Exercises that require GPU training include both `train.sbatch` and `train.sh` scripts for use on the TCML cluster. To run these, copy all source code files to your home directory on the cluster. Start training by executing:

  ```
  sbatch train.sbatch
  ```

  The data is already uploaded to the TCML cluster and will be handled by the sbatch script. Don't upload it again!
- When questions arise start a forum discussion in ILIAS or contact us via email:
  Manuel Traub (`manuel.traub@uni-tuebingen.de`)

Exercise 1: Latent Space and Noise Schedule [30 points]

In this exercise, we will learn how to create a suitable latent space for training a diffusion model. A well-constructed latent space ensures efficient and high-quality denoising during the diffusion process. Since our diffusion model operates solely within this latent space—without directly interacting with the image space—it is crucial to meet the following requirements:

Objectives for the Latent Space

- **Continuity**: The latent space should be continuous, without regions that are out of distribution for the decoder. This ensures the denoiser can smoothly navigate the latent space.
- **Strong Bottleneck**: A strong bottleneck reduces the latent dimensionality, saving computational resources during denoising.
- **Sufficient Capacity**: The bottleneck must still be wide enough to capture essential image features, enabling good reconstructions. The sampling quality depends heavily on this balance.
- **Effective Loss**: A loss function that promotes high-quality reconstructions is essential for achieving good latent representations.

You will work with an autoencoder implemented in `model/autoencoder/ffhq.py`, which can be trained using the trainer module in `model/lightning/vae_trainer.py`. The autoencoder is based on a **β-VAE** (beta variational autoencoder), with the β factor decaying over time to balance reconstruction and regularization. The training process is already set up for you and can be started using the following command:

```
python -m model.main -cfg config/ffhq-beta-vae.json --train-vae -n <run-number>
```

With the current settings, the autoencoder should converge after approximately 50,000 to 100,000 updates. Experiment with different hyperparameters, such as **latent-space size** and the **loss function** used in `model/autoencoder/ffhq.py`, to observe how they affect the reconstruction quality and latent space structure. To inspect the performace of the trained autoencoder use the following command:

```
python -m model.scripts.sample_vae -cfg config/ffhq-beta-vae.json \
       --load <path-to-vae-checkpoint>
```

(a) Train an Autoencoder [10 points]

Train the autoencoder and analyze its reconstruction quality.

Task:

1. Train the autoencoder using the default configuration and modify the settings (e.g., latent space size, β value) to experiment with different behaviors.
2. Provide **side-by-side visual comparisons** of input images versus their reconstructed outputs (encoded $\rightarrow$ decoded images).
3. Discuss your final hyperparameter choices and explain their impact on the latent space and reconstruction quality in your report.

Hints:

- Look at the trade-off between the size of the latent space (bottleneck) and the quality of reconstructions. Smaller latent spaces may lead to blurry reconstructions, while larger spaces might fail to enforce continuity.
- The β-VAE loss includes two terms: a reconstruction loss and a KL divergence term. Experiment with the final β factor and the decay period to balance these terms.

Deliverables:

- A few side-by-side comparison images (e.g., 3-5 examples) showing input images alongside their reconstructed outputs.
- A short explanation of the hyperparameters you used and the observed results.

(b) Noise Schedule [20 points]

The noise schedule defines how noise is added during the forward diffusion process and significantly impacts training and sampling [3]. In this exercise, we will compare two common noise schedules: **linear** and **cosine**. Your goal is to understand how these schedules influence the noise levels in the latent space.

Task 1: Implement a Cosine Noise Schedule

1. Open the file `model/base.py` and complete the implementation of the `cosine_beta_schedule` function.
2. The function should compute a cosine-based schedule for the noise variances β_t. Use the following formula:
$$\alpha_{\text{cum}}(t) = \cos^2\left(\frac{(t/T + s)}{1 + s} \cdot \frac{\pi}{2}\right),$$
where:
    - T is the total number of timesteps,
    - s is a small offset to ensure stability ($s = 0.008$ by default),

   The β_t values can then be derived as:
$$\beta_t = 1 - \frac{\alpha_{\text{cum}}(t+1)}{\alpha_{\text{cum}}(t)}.$$
3. Normalize the cumulative product α_{cum} such that $\alpha_{\text{cum}}(0) = 1$:
$$\alpha_{\text{cum}} = \frac{\alpha_{\text{cum}}}{\alpha_{\text{cum}}(0)}.$$

Task 2: Visualize Noise Levels

1. Use your trained autoencoder to encode an image into the latent space.
2. Add noise to the latent representation using the **cosine** and **linear** schedules for β_t, with $T = 10$ timesteps. Use the following formula to generate the noisy latent states:
$$z_t = \sqrt{\bar{\alpha}_t} \cdot z_0 + \sqrt{1 - \bar{\alpha}_t} \cdot \epsilon,$$
where:
    - z_0 is the latent representation of the input image,
    - $\bar{\alpha}_t = \prod_{i=1}^{t}(1 - \beta_i)$,
    - $\epsilon \sim \mathcal{N}(0, I)$ is Gaussian noise.
3. Decode the noisy latent states back into the image space for both schedules and compare the results.

Task 3: Analyze the Results

1. Visualize the outputs for each schedule at different timesteps (e.g., $t = [0, 1, ..., 9]$).

2. Discuss the differences in noise distribution and visual appearance between the two schedules.
3. Explain how the noise schedule might affect the denoiser's training and performance.

Deliverables:

- A plot or table of β_t and $\bar{\alpha}_t$ values for both schedules.
- Visualizations of noisy reconstructions at various timesteps for both schedules.
- A short explanation comparing the two schedules and their impact on training and sampling.

Exercise 2: Training the Denoiser [20 Points]

In this exercise, we will focus on the **denoiser** component of the diffusion model, which is trained to reverse the forward diffusion process. The objective is to minimize the loss between the predicted and true latent representations at each timestep, facilitating high-quality image reconstruction during sampling.

Theoretical Background

Our diffusion model employs the **v-parametrization** introduced in *Salimans and Ho, 2022* [1]. This parametrization reformulates the prediction task in terms of v, a linear combination of the original latent (z_0) and noise (ϵ), enhancing training stability and sample quality.

The v-parametrization is defined as:

$$v_t = \sqrt{\bar{\alpha}_t} \cdot \epsilon - \sqrt{1 - \bar{\alpha}_t} \cdot z_0,$$

where:

- $\bar{\alpha}_t = \prod_{i=1}^{t}(1 - \beta_i)$ is the cumulative product of the noise schedule terms.
- ϵ is the Gaussian noise added at timestep t.
- z_0 is the original (clean) latent representation.

Your Task

You will complete the implementation of the diffusion loss function and train the denoiser to predict v.

(a) Implement the Loss Function [10 Points]

1. **Goal**: Implement the missing components in the `compute_loss` function of the `AbstractLatentDiffusionModel` class (file: `model/base.py`):
   a) Compute the noisy latent z_t using the formula:

   $$z_t = \sqrt{\bar{\alpha}_t} \cdot z_0 + \sqrt{1 - \bar{\alpha}_t} \cdot \epsilon$$

   b) Compute the target v using:

   $$v_t = \sqrt{\bar{\alpha}_t} \cdot \epsilon - \sqrt{1 - \bar{\alpha}_t} \cdot z_0$$

2. Ensure the loss is calculated as the mean squared error (MSE) between the predicted v and the ground truth v:

$$\text{Loss} = \mathbb{E}_{t,z_0,\epsilon}\left[\|v_t - \text{Denoiser}(z_t, t)\|^2\right]$$

(b) Train the Denoiser [10 Points]

1. **Goal**: Train the denoiser by minimizing the v-parametrized loss function implemented in part (a). Use the provided ConvNeXt U-Net architecture for the denoiser, and the pre-trained VAE from Exercise 1 for encoding and decoding.
2. **Steps**:
   - Start the training process using the following command:

     ```
     python -m model.main -cfg config/ffhq-denoiser.json --train-denoiser \
            -n <run-number> --load-vae <path-to-the-vae-checkpoint>
     ```

     Replace `<path-to-the-vae-checkpoint>` with the location of the VAE checkpoint saved during Exercise 1.
   - Monitor the training logs for the average loss per batch. The loss should decrease over time as the denoiser learns to predict v_t more accurately. If training is unstable, try adjusting the learning rate or batch size in `config/ffhq-denoiser.json`. You can also use the `--load` flag to resume training from a saved checkpoint.
   - First generated images should become recognizable after around 100,000 updates. However, full convergence may require up to 500,000 updates. To improve stability and convergence, consider increasing the batch size during training or decreasing the learning rate as needed.
3. **Deliverables**:
   - A plot of the training loss over the number of updates.
   - A short explanation of the observed trends in the loss plot. For example:
     - Does the loss converge smoothly?
     - How does the choice of hyperparameters (e.g., learning rate, batch size) affect the training process?

Exercise 3: DDPM Sampling [20 Points]

In this exercise, we will focus on implementing the **Denoising Diffusion Probabilistic Model (DDPM) sampler** [4]. This involves deriving the predicted noise ϵ_{pred} and the mean of the posterior μ_t, then using them to reconstruct latent representations step-by-step during the reverse diffusion process.

Theoretical Background

The DDPM sampler performs reverse diffusion, reconstructing the clean latent representation z_0 from a noisy latent z_T. At each step t, the reverse diffusion is parameterized by the mean μ_t and variance σ_t^2 of the posterior $q(z_{t-1}|z_t, z_0)$.

Posterior Mean μ_t: The posterior mean is derived as:

$$\mu_t = \frac{1}{\sqrt{\alpha_t}} \left(z_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \cdot \epsilon_{\text{pred}} \right),$$

where:

- $\alpha_t = 1 - \beta_t$,
- $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$,
- β_t is the noise variance at step t,
- ϵ_{pred} is the predicted noise, which the denoiser predicts from z_t.

Predicted Noise ϵ_{pred}: Using the v-parametrization, we can derive ϵ_{pred} from the denoiser output (v_{pred}) as:

$$\epsilon_{\text{pred}} = \sqrt{1-\bar{\alpha}_t} \cdot z_t + \sqrt{\bar{\alpha}_t} \cdot v_{\text{pred}}.$$

Sampling z_{t-1}: To compute z_{t-1}, we sample from the posterior:

$$z_{t-1} \sim \mathcal{N}(\mu_t, \sigma_t^2 I),$$

where:

$$\sigma_t^2 = \beta_t \cdot \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}.$$

For $t = 0$, no additional noise is added, and we set $z_0 = \mu_t$.

Your Task

Complete the implementation of the `ddpm_step` function in `model/base.py` to perform a single reverse diffusion step. Your implementation should compute the required components, including the posterior mean μ_t and variance σ_t^2, and use them to sample z_{t-1} from the posterior distribution $q(z_{t-1}|z_t, z_0)$. For $t = 0$, ensure no additional noise is added.

Once implemented, test your sampler by generating image samples using the following command:

```
python -m model.scripts.sample -cfg config/ffhq-denoiser.json \
       --load <path-to-denoiser-checkpoint> \
       --num_samples 25 --seed 42 --clip_limit 100
```

Replace `<path-to-denoiser-checkpoint>` with the location of your trained denoiser from Exercise 2. Experiment with different seeds (`--seed`) and clipping limits (`--clip_limit`, suggested range: 1 to 3) to analyze the impact of noise control on sampling quality.

Deliverables

- **Generated Images:**
  - Provide a set of 5 images generated by your DDPM sampler using the trained denoiser.
  - Show the reverse diffusion process for a single example by visualizing the decoded images at key timesteps ($t = [T, T-n, T-2n, \dots, 0]$).

- **Analysis:**
  - Include a short explanation of your observations:
    - How do the generated images improve as $t \to 0$?
    - Are there any visual artifacts or signs of instability?

Exercise 4: DDIM Sampling [30 Points]

In this exercise, you will implement the **Denoising Diffusion Implicit Models (DDIM) sampler** [**2**], which provides a more efficient alternative to the DDPM sampler. Unlike DDPM, which requires the same number of timesteps for sampling as used during training, DDIM allows for fewer sampling steps without retraining the denoiser, making it particularly suitable for faster inference.

Theoretical Background

While DDPM adds stochastic noise at each reverse diffusion step, DDIM eliminates this randomness and instead deterministically updates the latent variables. DDIM achieves this by modeling the reverse process as an implicit transformation. This deterministic approach enables skipping steps in the diffusion process while preserving sample quality.

The core idea is to directly predict $z_{t_{\text{next}}}$ using a deterministic update based on z_t, z_0, and ϵ:

$$z_{t_{\text{next}}} = \begin{cases} z_0 & \text{if } t_{\text{next}} = 0, \\ \sqrt{\bar{\alpha}_{t_{\text{next}}}} \cdot z_0 + \sqrt{1 - \bar{\alpha}_{t_{\text{next}}}} \cdot \epsilon & \text{otherwise.} \end{cases}$$

To implement this update rule, you must recover z_0 and ϵ from the following equations:

$$v_t = \sqrt{\bar{\alpha}_t} \cdot \epsilon - \sqrt{1 - \bar{\alpha}_t} \cdot z_0.$$

$$z_t = \sqrt{\bar{\alpha}_t} \cdot z_0 + \sqrt{1 - \bar{\alpha}_t} \cdot \epsilon,$$

Your Task

You will derive and implement the deterministic DDIM update rule for $z_{t_{\text{next}}}$. Specifically, you need to:

1. Derive the equations for z_0 and ϵ from the given v-parametrization and include your derivations in your report.
2. Implement the missing components in the `ddim_step` function in `model/base.py`:
   - Compute z_0 and ϵ based on the current latent z_t and predicted v_t (from the denoiser).
   - Implement the DDIM update rule to compute $z_{t_{\text{next}}}$.
3. Test the DDIM sampler by generating samples from random Gaussian noise.

Run the DDIM Sampler

After completing the implementation, use the following command to test your DDIM sampler:

```
python -m model.scripts.sample -cfg config/ffhq-ddim.json \
       --load <path-to-denoiser-checkpoint> \
       --num_samples 25 --seed 42 --clip_limit 1.5 --ddim
```

Replace `<path-to-denoiser-checkpoint>` with the location of your trained denoiser from Exercise 2. Experiment with the number of timesteps (`--steps`), seeds (`--seed`), and clipping limits (`--clip_limit`) to evaluate the quality and diversity of the generated samples.

Deliverables

- Your derivations for z_0 and ϵ, clearly explaining the steps based on the v-parametrization.
- A working implementation of the `ddim_step` function.
- Generated image samples using the DDIM sampler, with a brief analysis of the results, including:
  - How does the quality of the samples change with the number of timesteps?
  - How does the deterministic nature of DDIM affect sample diversity compared to DDPM?

Bonus Exercise: Reimagining Images [20 Points]

In this bonus exercise, you will explore the creative potential of diffusion models by starting from a partly noised image and letting the model reimagine it. You will compare the results produced by the DDPM and DDIM samplers at different noise strengths, analyzing their behavior and capabilities.

Your Task

1. **Noising an Input Image:**
   - Select an input image and encode it into the latent space using the pre-trained VAE from Exercise 1.
   - Add noise to the latent representation using the forward diffusion process:

   $$z_t = \sqrt{\bar{\alpha}_t} \cdot z_0 + \sqrt{1 - \bar{\alpha}_t} \cdot \epsilon,$$

   - Experiment with different levels of noise strength $t \in \{T/4, T/2, 3T/4, T\}$, where T is the total number of timesteps.
2. **Reimagining the Image:**
   - Use both the DDPM and DDIM samplers to perform reverse diffusion, reconstructing the image from the noised latent z_t.
   - For DDIM, try sampling with a reduced number of steps (e.g., $T \in [10, .., 100]$ to evaluate its efficiency and quality at lower sampling budgets.
3. **Visualize and Compare:**
   - For each noise strength t, visualize the reconstructed images from both DDPM and DDIM samplers side-by-side.
   - Compare the reconstructions in terms of fidelity, diversity, and artifacts. Discuss how the samplers handle the reverse diffusion process at different noise levels.

Deliverables

- **Visualization:**
  - Provide a grid of images showing the original input, the noised image at different noise levels, and the reconstructions from DDPM and DDIM for each noise level.
  - Include intermediate reconstructions (e.g., $t = T, T - n, T - 2n, \dots, 0$) for at least one noise level to illustrate the reverse diffusion process.
- **Analysis:**
  - Discuss how the reconstructions differ between DDPM and DDIM at varying noise strengths.
  - Reflect on how the deterministic nature of DDIM affects its ability to reimagine the image compared to the stochastic DDPM process.
  - Comment on the quality-speed trade-off when using fewer steps in DDIM sampling.

References

1 Salimans, Tim, and Jonathan Ho. `Progressive distillation for fast sampling of diffusion models.` arXiv preprint arXiv:2202.00512 (2022).

2 Song, Jiaming, Chenlin Meng, and Stefano Ermon. `Denoising diffusion implicit models.` arXiv preprint arXiv:2010.02502 (2020).

3 Chen, Ting. `On the importance of noise scheduling for diffusion models.` arXiv preprint arXiv:2301.10972 (2023).

4 Ho, Jonathan, Ajay Jain, and Pieter Abbeel. `Denoising diffusion probabilistic models.` Advances in neural information processing systems 33 (2020): 6840-6851.