Homework #4

Report

- 간단한 Conditional Difussion Model 이해하기-

전공 컴퓨터공학전공

이름 한사랑

학번 2271064

1. 과제 및 모델 개요

본 코드에서는 CIFAR-10 이미지 생성을 위한 조건부(Conditional) Diffusion Model을 PyTorch로 구현하고, 전체 파이프라인의 구조와 동작 원리를 요약합니다.
특히, 데이터 준비, 확산(Diffusion) 과정 정의, 조건부 U-Net 백본 설계, 모델 학습 및 이미지 생성(샘플링)까지의 전 과정을 다룹니다.

구현된 주요 요소는 다음과 같습니다:

- CIFAR-10 데이터셋 로딩 및 정규화, DataLoader 구성
- 코사인 스케줄 기반의 Diffusion Process 정의 및 노이즈 추가 함수 구현
- 시간 및 클래스 임베딩을 활용한 조건부 U-Net(Conditional U-Net) 백본 설계
- MSE 손실 기반의 학습 루프 및 EMA(Exponential Moving Average) 적용
- 조건부 이미지 생성(샘플링) 함수 및 각 클래스별 생성 이미지 시각화

실험에는 CIFAR-10 공식 데이터셋(32x32 컬러 이미지, 10개 클래스)이 사용되었으며, 구현은 PyTorch 프레임워크와 GPU 환경에서 수행되었습니다.

2. 데이터 준비

1. Importing Necessary Libraries and GPU Configuration

2. Preparing the Dataset

- 코드에서는 CIFAR-10 이미지 데이터셋을 사용합니다. 이미지는 `transforms.ToTensor()`로 텐서로 변환되고, `transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))`로 [-1, 1] 범위로 정규화됩니다. 이렇게 정규화하면 디퓨전 모델이 노이즈를 더 잘 예측할 수 있습니다.
- PyTorch의 `torchvision.datasets.CIFAR10` 클래스를 이용해 데이터를 불러오고, DataLoader로 배치 단위로 모델에 전달합니다.
- `worker_init_fn`을 통해 랜덤 시드를 고정하여 실험의 재현성을 높였습니다.
- 필요에 따라 특정 클래스만 골라 쓸 수 있도록 `BinaryCIFAR10` 클래스도 정의되어 있습니다.
이렇게 준비된 데이터는 모델이 각 클래스별 이미지를 생성하는 데 사용됩니다.

3. Diffusion Process 설명

3. Defining Diffusion Process

- 이 코드에서는 디퓨전 프로세스를 `DiffusionProcess` 클래스로 구현합니다. 디퓨전 프로세스란, 원본 이미지에 점진적으로 노이즈를 추가해 완전히 무작위한 이미지로 만들고, 이를 다시 복원하는 과정을 의미합니다.
- 먼저, 코사인 스케줄을 이용해 각 단계별 노이즈 비율(`beta`)을 계산합니다.
  이 값들을 바탕으로 노이즈가 추가되는 정도와, 반대로 이미지를 복원할 때 필요한 파라미터들이 미리 계산됩니다.
- `add_noise` 함수는 주어진 이미지에 랜덤 노이즈를 섞어, 디퓨전의 forward(노이즈 추가) 과정을 구현합니다.
  이렇게 만들어진 노이즈 이미지는 모델 학습에 사용되며, 모델은 주어진 노이즈 이미지에서 원래의 노이즈를 예측하도록 훈련됩니다.

4. Conditional U-Net 구조

4. Implementing Conditional U-Net

이 코드에서는 CIFAR-10 이미지 생성을 위해 Conditional U-Net 구조를 사용합니다.
U-Net은 인코더(다운샘플링)와 디코더(업샘플링)로 구성되어 있고, 각 단계에서 skip connection을 통해 정보를 전달합니다. 특징적으로, 시간 정보와 클래스 정보를 모두 임베딩하여 모델에 입력합니다.

- 시간 임베딩은 `SinusoidalPositionEmbeddings` 클래스를 통해 각 디퓨전 단계별로 고유한 벡터로 변환됩니다.
- 클래스 임베딩은 CIFAR-10의 10개 클래스를 임베딩 벡터로 변환합니다.
- 두 임베딩을 합쳐서 각 블록에 전달함으로써, 모델이 "언제(몇 번째 디퓨전 단계)"와 "무엇(어떤 클래스)"을 생성해야 하는지 알 수 있습니다.

각 `ConvBlock`은 입력 이미지와 임베딩 정보를 함께 받아서 처리하며, 인코더와 디코더 사이에는 skip connection이 적용되어 세부 정보를 보존합니다. 최종적으로, 이 구조 덕분에 모델은 원하는 클래스의 이미지를 효과적으로 생성할 수 있습니다.

5. 학습 과정

5. Training Loop

- 학습 과정에서는 모델이 노이즈가 추가된 이미지를 입력받아, 원래 추가된 노이즈를 예측하도록 훈련됩니다.
- 각 배치마다 무작위로 디퓨전 단계(timestep)를 선택하고, 해당 단계만큼 노이즈가 섞인 이미지를 생성합니다.
- 모델은 이 이미지를 보고 실제로 추가된 노이즈를 예측하며, 예측값과 실제 노이즈의 차이(MSE 손실)를 최소화하도록 학습합니다.
- 최적화에는 `AdamW` 옵티마이저와 `OneCycleLR` 스케줄러가 사용되며, 학습 안정성을 위해 gradient clipping과 EMA(Exponential Moving Average) 모델이 적용됩니다.
  EMA 모델은 학습 중 모델 파라미터의 이동평균을 저장해, 더 안정적인 이미지 생성을 돕습니다.

6. 샘플링(이미지 생성) 과정

6. Conditional Sampling

- 이미지 생성(샘플링) 과정에서는, 먼저 완전히 무작위한 노이즈 이미지를 만듭니다.
  그 다음, 디퓨전 과정을 거꾸로 진행하면서, 각 단계마다 모델이 예측한 노이즈를 점진적으로 제거합니다. 이때 원하는 클래스 정보를 함께 입력하면, 해당 클래스에 맞는 이미지를 생성할 수 있습니다.
- 최종적으로, 모든 디퓨전 단계를 거치면 노이즈가 제거된 새로운 이미지가 생성됩니다.
  이렇게 생성된 이미지는 [0, 1] 범위로 변환되어 시각화에 사용됩니다.

7. 시각화 및 결과

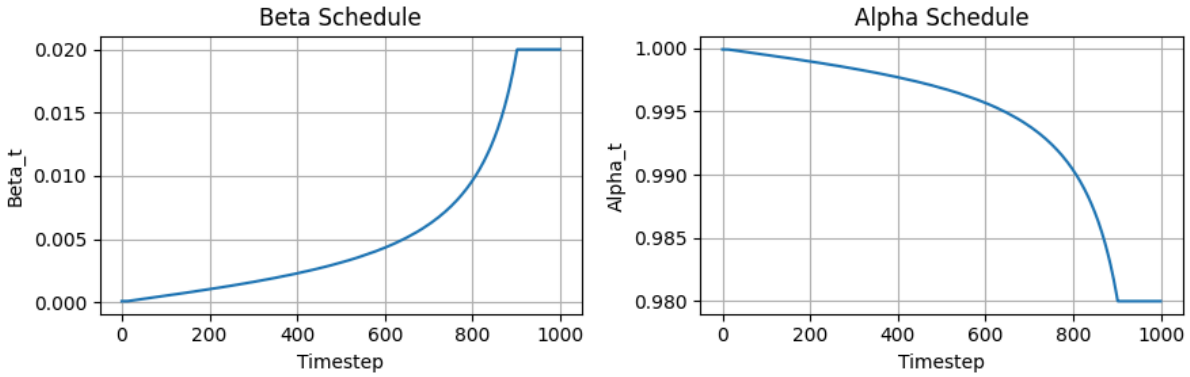
7. Visualizaion

8. Execution scripts

모델 학습 및 평가 과정에서는 다양한 시각화 결과를 통해 디퓨전 모델의 동작과 성능을 확인할 수 있습니다.

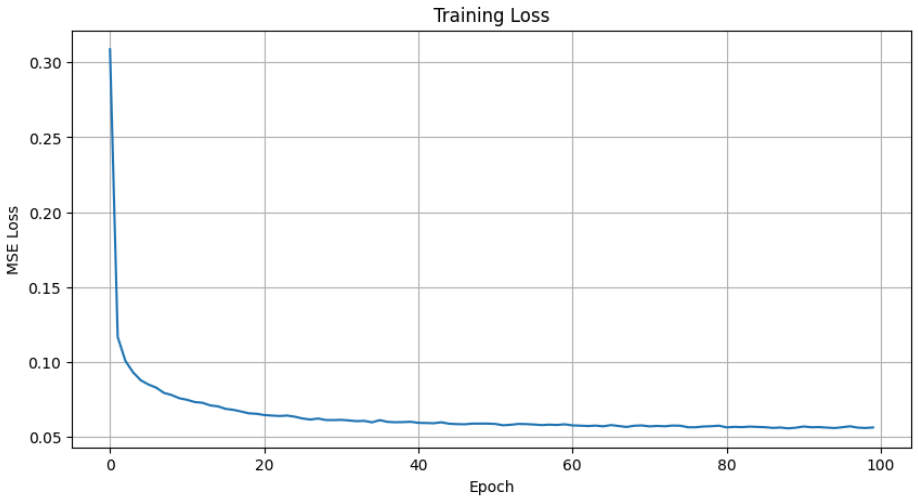
- **베타/알파 스케줄 시각화**

  `plot_beta_schedule` 함수를 통해 디퓨전 과정에서 사용되는 베타(`beta`)와 알파(`alpha`) 값의 변화를 그래프로 확인할 수 있습니다.이 그래프는 각 타임스텝마다 노이즈가 얼마나 추가되는지, 그리고 그에 따라 이미지가 어떻게 변하는지 이해하는 데 도움이 됩니다.



- **학습 손실 곡선**

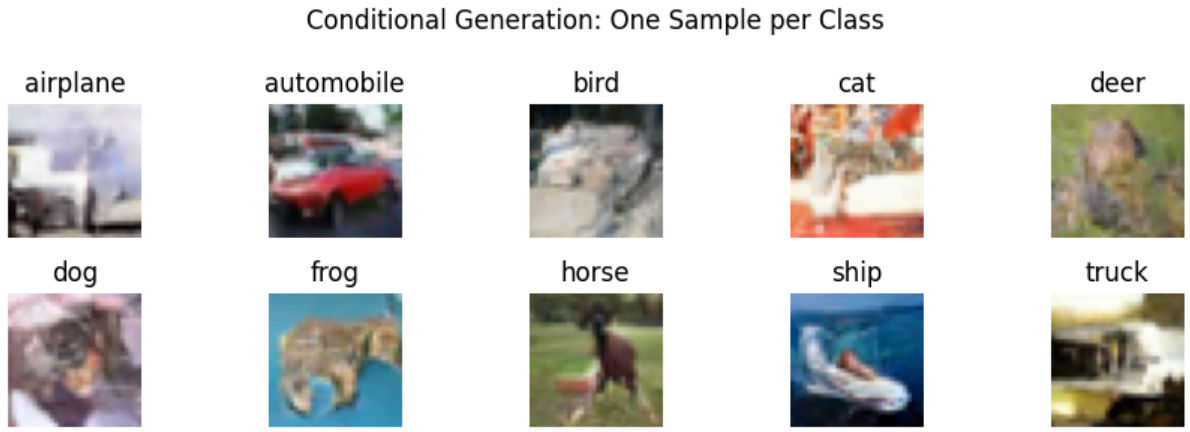
  학습 과정에서의 손실(MSE Loss) 변화를 그래프로 나타내어, 모델이 점차적으로 노이즈 예측을 잘하게 되는 과정을 확인할 수 있습니다.



- **클래스별 생성 이미지**

  `visualize_conditional_generation` 함수를 이용해, CIFAR-10의 10개 클래스

각각에 대해 조건부로 이미지를 생성하고 시각화합니다. 이를 통해 모델이 각 클래스의 특징을 잘 학습했는지 확인할 수 있습니다.



- **단일 클래스(고양이) 이미지 생성**

  `sample_conditional_images` 함수를 사용해 고양이 클래스(예: label=3) 이미지를 여러 장 생성하여, 같은 클래스 내에서도 다양한 이미지를 만들어낼 수 있음을 보여줍니다.

이처럼, Conditional Diffusion Model은 원하는 클래스의 이미지를 다양하게 생성할 수 있으며, 시각화를 통해 모델의 성능과 생성 결과를 직관적으로 확인할 수 있습니다.