MNIST Diffusion Project

Generated by Doxygen 1.9.1

Chapter 1

MNIST Diffusion Project

Software used to investigate the use of generative diffusion modelling to recreate realistic monochrome images of handwritten digits, using the classical MNIST dataset as training data. The software utilises PyTorch to implement the neural network models and the training procedure.

1.1 Modules

1.1.1 models

This module contains code used to instantiate standard image-to-image convolutional neural networks, as well as a DDPM class, with methods that enable training the image-to-image model according to the diffusion paradigm, and similarly performing diffusion sampling.

1.1.2 train

This module contains the ddpm_train function, which implements the diffusion training process, and saves the model parameters and samples periodically, according to specified training parameters.

Author

Created by J. Hughes on 06/12/2023.

Chapter 2

Namespace Index

2.1 Packages

Here are the packages with brief descriptions (if available):

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 diffusiontools Namespace Reference

Namespaces

- analysis
- models
- train

6.2 diffusiontools.analysis Namespace Reference

Functions

- def plot_learning_curve (List[float] losses_train, List[float] losses_val, str title, str filename)

  Plot changes in loss function across epochs for model training.
- def compute_image_diffusion (DDPM model, List[float] image_fractions, int n_sample, size, device)

  Save intermediate image generations at specified parts of the Gaussian diffusion process.
- torch.Tensor compute_image_diffusion_custom (DMCustom model, List[int] image_fractions, int n_sample, size, device, bool direct=False)

  Save intermediate image generations at specified parts of the custom diffusion process.
- def plot_image_diffusion (model, List[int] image_fractions, int n_sample, size, device, str title, str filename)

  Plot intermediate image generations at specified parts of the custom diffusion process.
- def plot_samples (np.ndarray samples, int n_row, int n_col, str filename)

  Produce matrix of synthetic image samples.
- def plot_gt_direct_diffusion (np.ndarray direct_sample, np.ndarray diffusion_sample, int n_plot, str filename)

  Compare direct and diffused reconstructions.
- def compute_variance (np.ndarray sample)

  Compute the total variance of a sample of vectors.
- def degradation_demo (DMCustom model, device, List[int] t_list, str title, str filename)

  Plot MNIST images of digits 1 to 4 with varying custom degradation levels.
- def compute_tsne_kl_div (List[np.ndarray] samples, List[str] labels)

  Map the given samples into a 2D t-SNE embedding alongside the MNIST test data, and compute the KL divergences.
- def plot_sample_tsne (np.ndarray density_gt, np.ndarray sample_1_fitted, np.ndarray sample_2_fitted, np.ndarray density_1, np.ndarray density_2, str label_1, str label_2, str filename)

  Plot the t-SNE embeddings of two samples alongside the MNIST test data fitted density.
- def plot_mnist_tsne (str filename)

  Plot the t-SNE embeddings of MNIST test data alone, digit classes indicated.

6.2.1 Function Documentation

6.2.1.1 compute_image_diffusion()

```
def diffusiontools.analysis.compute_image_diffusion (
             DDPM model,
             List[float] image_fractions,
             int n_sample,
              size,
              device )
```

Save intermediate image generations at specified parts of the Gaussian diffusion process.

Similar to the sample() method of the DDPM class, except intermediate reconstructions are saved.

Parameters

model	DDPM Class instance, which implements the Gaussian noising schedule
image_fractions	List of floats in [0, 1] specifying the stages of diffusion in which to save the current reconstruction
n_sample	Integer specifying number of samples to generate
size	Image size as (channels, height, width)
device	PyTorch device variable used to control processing unit

Returns

image_idx List of discrete integer time steps at which reconstructions were saved

images List of intermediate image reconstructions.

6.2.1.2 compute_image_diffusion_custom()

```
torch.Tensor diffusiontools.analysis.compute_image_diffusion_custom (
             DMCustom model,
             List[int] image_fractions,
             int n_sample,
              size,
              device,
             bool  direct = False )
```

Save intermediate image generations at specified parts of the custom diffusion process.

Generates an initial image by maximally degrading an MNIST test sample according to the model's parameters, and then performing the reverse diffusion process.

Parameters

model	DMCustom Class instance, which implements the custom noising schedule

Parameters

image_fractions	List of floats in [0, 1] specifying the stages of diffusion in which to save the current reconstruction
n_sample	Integer specifying number of samples to generate
size	Image size as (channels, height, width)
device	PyTorch device variable used to control processing unit
direct	Determines whether the very first reconstruction is saved (without a degradation)

Returns

image_idx List of discrete integer time steps at which reconstructions were saved

images List of intermediate image reconstructions.

direct_img Stack of direct image recosntructions if this was specified

6.2.1.3 compute_tsne_kl_div()

```
def diffusiontools.analysis.compute_tsne_kl_div (
            List[np.ndarray] samples,
            List[str] labels )
```

Map the given samples into a 2D t-SNE embedding alongside the MNIST test data, and compute the KL divergences.

Parameters

samples	List of different generated image stacks
labels	Labels for the sample to use in the output

Returns

density_gt Fitted GMM density values for the GT data

samples_fitted t-SNE embeddings for each of the samples

samples_densities Fitted GMM density maps for each sample

6.2.1.4 compute_variance()

```
def diffusiontools.analysis.compute_variance (
            np.ndarray sample )
```

Compute the total variance of a sample of vectors.

Parameters

sample	NumPy array whose first dimension is the image index

Returns

total_variance The computed total variance of the vectors

6.2.1.5 degradation_demo()

```
def diffusiontools.analysis.degradation_demo (
            DMCustom model,
             device,
            List[int] t_list,
            str title,
            str  filename )
```

Plot MNIST images of digits 1 to 4 with varying custom degradation levels.

Parameters

model	DMCustom class instance
device	PyTorch device variable used to control processing unit
t_list	List of time steps at which to demonstrate the degradation
title	String for title to be included in the figure
filename	String determining saved filename for figure

6.2.1.6 plot_gt_direct_diffusion()

```
def diffusiontools.analysis.plot_gt_direct_diffusion (
            np.ndarray direct_sample,
            np.ndarray diffusion_sample,
            int n_plot,
            str filename )
```

Compare direct and diffused reconstructions.

Produce a 3x[n_plot] grid of images comparing corresponding ground truth vs. direct vs. diffusion image samples, and then compute MSE statistics for the full samples.

Parameters

direct_sample	NumPy array whose first dimension is the image index
diffusion_sample	NumPy array whose first dimension is the image index
filename	String determining saved filename for figure

6.2.1.7 plot_image_diffusion()

```
def diffusiontools.analysis.plot_image_diffusion (
              model,
             List[int] image_fractions,
             int n_sample,
              size,
              device,
             str title,
             str filename )
```

Plot intermediate image generations at specified parts of the custom diffusion process.

Parameters

model	DDPM or DMCustom class instance
image_fractions	List of floats in [0, 1] specifying the stages of diffusion in which to save the current reconstruction
n_sample	Integer specifying number of samples to generate
size	Image size as (channels, height, width)
device	PyTorch device variable used to control processing unit
title	String for title to be included in the figure
filename	String determining saved filename for figure

6.2.1.8 plot_learning_curve()

```
def diffusiontools.analysis.plot_learning_curve (
             List[float] losses_train,
             List[float] losses_val,
             str title,
             str filename )
```

Plot changes in loss function across epochs for model training.

Parameters

losses_train	List of training losses per epoch
losses_val	List of validation losses per epoch
title	String of title to be displayed in plot
filename	String of filename to save to

6.2.1.9 plot_mnist_tsne()

```
def diffusiontools.analysis.plot_mnist_tsne (
```

```
            str filename )
```

Plot the t-SNE embeddings of MNIST test data alone, digit classes indicated.

Parameters

filename	String determining saved filename for figure

6.2.1.10 plot_sample_tsne()

```
def diffusiontools.analysis.plot_sample_tsne (
            np.ndarray density_gt,
            np.ndarray sample_1_fitted,
            np.ndarray sample_2_fitted,
            np.ndarray density_1,
            np.ndarray density_2,
            str label_1,
            str label_2,
            str filename )
```

Plot the t-SNE embeddings of two samples alongside the MNIST test data fitted density.

Parameters

density_gt	Mesh of fitted GMM density for the 2D ground truth embedding
sample_1_fitted	2D embeddings for sample 1
sample_2_fitted	2D embeddings for sample 2
density_1	Mesh of fitted GMM density for sample 1
density_2	Mesh of fitted GMM density for sample 2
label_1	Legend label for sample 1
label_2	Legend label for sample 2
filename	String determining saved filename for figure

6.2.1.11 plot_samples()

```
def diffusiontools.analysis.plot_samples (
            np.ndarray samples,
            int n_row,
            int n_col,
            str filename )
```

Produce matrix of synthetic image samples.

Parameters

samples	NumPy array whose first dimension is the image index

Parameters

image_fractions	List of floats in [0, 1] specifying the stages of diffusion in which to save the current reconstruction
n_row	Integer specifying number of rows
n_col	Integer specifying number of columns
filename	String determining saved filename for figure

6.3 diffusiontools.models Namespace Reference

Classes

- class CNNBlock

  Class instantiating a convolutional block with layer normalisation and non-linear activation.
- class CNN

  Defines CNN class used to reconstruct images during diffusion.
- class DDPM

  Defines Gaussian diffusion model class.
- class DMCustom

  Class defining custom diffusion model.

Functions

- Dict[str, torch.Tensor] ddpm_schedules (float beta1, float beta2, int T)

  Returns pre-computed schedules for DDPM sampling with a linear noise schedule.

6.3.1 Function Documentation

6.3.1.1 ddpm_schedules()

```
Dict[str, torch.Tensor] diffusiontools.models.ddpm_schedules (
            float beta1,
            float beta2,
            int   T )
```

Returns pre-computed schedules for DDPM sampling with a linear noise schedule.

Parameters

beta1	Minimal noise parameter
beta2	Taximal noise parameter
T	Total number of discrete diffusion time steps

Returns

Dict[str,torch.Tensor] Dictionary storing alpha and beta noise parameter vectors

6.4 diffusiontools.train Namespace Reference

Functions

- def train_model (nn.Module ddpm, torch.optim.Optimizer optim, DataLoader dataloader_train, DataLoader dataloader_val, Accelerator accelerator, int n_epoch, int save_interval, str sample_path, str checkpoint_path, str config_id)

  Handles the training process for a diffusion model.

6.4.1 Function Documentation

6.4.1.1 train_model()

```
def diffusiontools.train.train_model (
            nn.Module ddpm,
            torch.optim.Optimizer optim,
            DataLoader dataloader_train,
            DataLoader dataloader_val,
            Accelerator accelerator,
            int n_epoch,
            int save_interval,
            str sample_path,
            str checkpoint_path,
            str config_id )
```

Handles the training process for a diffusion model.

Parameters

ddpm	Diffusion model object; either DDPM or DMCustom class instance
dataloader_train	Iterable that yields pairs of matched ground truth and degraded image samples for training
dataloader_val	Iterable that yields pairs of matched ground truth and degraded image samples for model validation
accelerator	accelerate package object which simplifies the management of CPU/GPU processing devices
n_epoch	Total number of epochs to train for before stopping
save_interval	Number of epochs to wait for between saving models
sample_path	Specifies where to save model samples (deprecated)
checkpoint_path	Specifies where to save model checkpoint files
config_id	Specifies configuration id to use to label model checkpoint files

6.5 final_analysis Namespace Reference

Variables

- device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
- gt_1
- model_1 = DDPM(gt=gt_1, betas=(0.0001, 0.02), n_T=1000).to(device)
- ckpt_0001_0490
- gt_2
- model_2 = DDPM(gt=gt_2, betas=(0.001, 0.2), n_T=100).to(device)
- ckpt_0002_0490
- gt_9
- model_9
- ckpt_0009_0080
- gt_10
- model_10
- ckpt_0010_0180
- ckpt_0001_0120
- ckpt_0002_0080
- ckpt_0009_0020
- ckpt_0010_0140
- int SAMPLE_SIZE = 200
- _
- sample_1
- sample_2
- sample_9
- sample_9_direct
- direct
- sample_10
- sample_10_direct
- total_var_1 = compute_variance(sample_1[0])
- total_var_2 = compute_variance(sample_2[0])
- total_var_9 = compute_variance(sample_9[0])
- total_var_10 = compute_variance(sample_10[0])
- density_gt
- samples_fitted
- samples_densities

6.5.1 Variable Documentation

6.5.1.1 _

`final_analysis._ [private]`

6.5.1.2 ckpt_0001_0120

final_analysis.ckpt_0001_0120

Initial value:

```
1 =  torch.load(
2     "data/DDPM/checkpoint/ddpm_checkpoint_0001_0120.pt", map_location=device
3 )
```

6.5.1.3 ckpt_0001_0490

final_analysis.ckpt_0001_0490

Initial value:

```
1 =  torch.load(
2     "data/DDPM/checkpoint/ddpm_checkpoint_0001_0490.pt", map_location=device
3 )
```

6.5.1.4 ckpt_0002_0080

final_analysis.ckpt_0002_0080

Initial value:

```
1 =  torch.load(
2     "data/DDPM/checkpoint/ddpm_checkpoint_0002_0080.pt", map_location=device
3 )
```

6.5.1.5 ckpt_0002_0490

final_analysis.ckpt_0002_0490

Initial value:

```
1 =  torch.load(
2     "data/DDPM/checkpoint/ddpm_checkpoint_0002_0490.pt", map_location=device
3 )
```

6.5.1.6 ckpt_0009_0020

final_analysis.ckpt_0009_0020

Initial value:

```
1 =  torch.load(
2     "data/DDPM/checkpoint/ddpm_checkpoint_0009_0020.pt", map_location=device
3 )
```

6.5.1.7 ckpt_0009_0080

final_analysis.ckpt_0009_0080

Initial value:

```
1 =  torch.load(
2     "data/DDPM/checkpoint/ddpm_checkpoint_0009_0080.pt", map_location=device
3 )
```

6.5.1.8 ckpt_0010_0140

final_analysis.ckpt_0010_0140

Initial value:

```
1 =  torch.load(
2     "data/DDPM/checkpoint/ddpm_checkpoint_0010_0140.pt", map_location=device
3 )
```

6.5.1.9 ckpt_0010_0180

final_analysis.ckpt_0010_0180

Initial value:

```
1 =  torch.load(
2     "data/DDPM/checkpoint/ddpm_checkpoint_0010_0180.pt", map_location=device
3 )
```

6.5.1.10 density_gt

final_analysis.density_gt

6.5.1.11 device

```
final_analysis.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

6.5.1.12 direct

final_analysis.direct

6.5.1.13 gt_1

final_analysis.gt_1

Initial value:

```
1 =  CNN(
2     in_channels=1,
3     expected_shape=(28, 28),
4     n_hidden=(16, 32, 32, 16),
5     act=nn.GELU,
6 ).to(device)
```

6.5.1.14 gt_10

final_analysis.gt_10

Initial value:

```
1 =  CNN(
2     in_channels=1,
3     expected_shape=(28, 28),
4     n_hidden=(16, 32, 32, 16),
5     act=nn.GELU,
6 ).to(device)
```

6.5.1.15 gt_2

final_analysis.gt_2

Initial value:

```
1 =  CNN(
2     in_channels=1,
3     expected_shape=(28, 28),
4     n_hidden=(16, 32, 32, 16),
5     act=nn.GELU,
6 ).to(device)
```

6.5.1.16 gt_9

final_analysis.gt_9

Initial value:

```
1 =  CNN(
2     in_channels=1,
3     expected_shape=(28, 28),
4     n_hidden=(16, 32, 32, 16),
5     act=nn.GELU,
6 ).to(device)
```

6.5.1.17 model_1

final_analysis.model_1 = DDPM(gt=gt_1, betas=(0.0001, 0.02), n_T=1000).to(device)

6.5.1.18 model_10

```
final_analysis.model_10
```

Initial value:

```
1 =  DMCustom(
2     gt=gt_10, alphas=(0.035, 0.2), n_T=20, size=(1, 28, 28)
3 ).to(device)
```

6.5.1.19 model_2

```
final_analysis.model_2 = DDPM(gt=gt_2, betas=(0.001, 0.2), n_T=100).to(device)
```

6.5.1.20 model_9

```
final_analysis.model_9
```

Initial value:

```
1 =  DMCustom(
2     gt=gt_9,
3     alphas=(0.035, 0.15),
4     n_T=100,
5     size=(1, 28, 28),
6     criterion=nn.L1Loss(),
7 ).to(device)
```

6.5.1.21 sample_1

```
final_analysis.sample_1
```

6.5.1.22 sample_10

```
final_analysis.sample_10
```

6.5.1.23 sample_10_direct

```
final_analysis.sample_10_direct
```

6.5.1.24 sample_2

```
final_analysis.sample_2
```

6.5.1.25 sample_9

```
final_analysis.sample_9
```

6.5.1.26 sample_9_direct

```
final_analysis.sample_9_direct
```

6.5.1.27 SAMPLE_SIZE

```
int final_analysis.SAMPLE_SIZE = 200
```

6.5.1.28 samples_densities

```
final_analysis.samples_densities
```

6.5.1.29 samples_fitted

```
final_analysis.samples_fitted
```

6.5.1.30 total_var_1

```
final_analysis.total_var_1 = compute_variance(sample_1[0])
```

6.5.1.31 total_var_10

```
final_analysis.total_var_10 = compute_variance(sample_10[0])
```

6.5.1.32 total_var_2

```
final_analysis.total_var_2 = compute_variance(sample_2[0])
```

6.5.1.33 total_var_9

```
final_analysis.total_var_9 = compute_variance(sample_9[0])
```

6.6 intermediate_analysis Namespace Reference

Variables

- device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
- ckpt_0001
- gt_1
- model_1 = DDPM(gt=gt_1, betas=(0.0001, 0.02), n_T=1000).to(device)
- ckpt_0002
- gt_2
- model_2 = DDPM(gt=gt_2, betas=(0.0001, 0.02), n_T=1000).to(device)
- ckpt_0009
- gt_9
- model_9
- ckpt_0010
- gt_10
- model_10

6.6.1 Variable Documentation

6.6.1.1 ckpt_0001

```
intermediate_analysis.ckpt_0001
```

Initial value:

```
1 =  torch.load(
2         f"data/DDPM/checkpoint/ddpm_checkpoint_0001_{epoch:04d}.pt",
3         map_location=device,
4     )
```

6.6.1.2 ckpt_0002

intermediate_analysis.ckpt_0002

Initial value:

```
1 =  torch.load(
2         f"data/DDPM/checkpoint/ddpm_checkpoint_0002_{epoch:04d}.pt",
3         map_location=device,
4     )
```

6.6.1.3 ckpt_0009

intermediate_analysis.ckpt_0009

Initial value:

```
1 =  torch.load(
2             f"data/DDPM/checkpoint/ddpm_checkpoint_0009_{epoch:04d}.pt",
3             map_location=device,
4         )
```

6.6.1.4 ckpt_0010

intermediate_analysis.ckpt_0010

Initial value:

```
1 =  torch.load(
2         f"data/DDPM/checkpoint/ddpm_checkpoint_0010_{epoch:04d}.pt",
3         map_location=device,
4     )
```

6.6.1.5 device

intermediate_analysis.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

6.6.1.6 gt_1

intermediate_analysis.gt_1

Initial value:

```
1 =  CNN(
2         in_channels=1,
3         expected_shape=(28, 28),
4         n_hidden=(16, 32, 32, 16),
5         act=nn.GELU,
6     ).to(device)
```

6.6.1.7 gt_10

intermediate_analysis.gt_10

Initial value:

```
1 =  CNN(
2          in_channels=1,
3          expected_shape=(28, 28),
4          n_hidden=(16, 32, 32, 16),
5          act=nn.GELU,
6      ).to(device)
```

6.6.1.8 gt_2

intermediate_analysis.gt_2

Initial value:

```
1 =  CNN(
2          in_channels=1,
3          expected_shape=(28, 28),
4          n_hidden=(16, 32, 32, 16),
5          act=nn.GELU,
6      ).to(device)
```

6.6.1.9 gt_9

intermediate_analysis.gt_9

Initial value:

```
1 =  CNN(
2              in_channels=1,
3              expected_shape=(28, 28),
4              n_hidden=(16, 32, 32, 16),
5              act=nn.GELU,
6          ).to(device)
```

6.6.1.10 model_1

```
intermediate_analysis.model_1 = DDPM(gt=gt_1, betas=(0.0001, 0.02), n_T=1000).to(device)
```

6.6.1.11 model_10

intermediate_analysis.model_10

Initial value:

```
1 =  DMCustom(
2          gt=gt_10, alphas=(0.035, 0.2), n_T=20, size=(1, 28, 28)
3      ).to(device)
```

6.6.1.12 model_2

```
intermediate_analysis.model_2 = DDPM(gt=gt_2, betas=(0.0001, 0.02), n_T=1000).to(device)
```

6.6.1.13 model_9

```
intermediate_analysis.model_9
```

Initial value:

```
1 =  DMCustom(
2             gt=gt_9,
3             alphas=(0.035, 0.15),
4             n_T=100,
5             size=(1, 28, 28),
6             criterion=nn.L1Loss(),
7         ).to(device)
```

6.7 run Namespace Reference

Variables

- config = cfg.ConfigParser()
- input_file = sys.argv[1]
- noise_min = config.getfloat("model", "noise_min", fallback=1e-4)
- noise_max = config.getfloat("model", "noise_max", fallback=0.02)
- n_T = config.getint("model", "n_T", fallback=1000)
- degradation = config.get("model", "degradation", fallback="gaussian")
- n_hidden = config.get("model", "n_hidden", fallback="16 32 32 16")
- loss_fn = config.get("model", "loss_fn", fallback="L2")
- n_epoch = config.getint("training", "n_epoch", fallback=100)
- batch_size = config.getint("training", "batch_size", fallback=128)
- lr_initial = config.getfloat("training", "lr_initial", fallback=2e-4)
- checkpoint_path
- sample_path
- save_interval = config.getint("output", "save_interval", fallback=10)
- config_id = config.getint("output", "config_id", fallback=1234)
- tf
- dataset = MNIST("./data", train=True, download=True, transform=tf)
- dataset_train
- dataset_val
- dataloader_train
- dataloader_val
- gt
- criterion = nn.MSELoss()
- model
- optim = torch.optim.Adam(model.parameters(), lr=lr_initial)
- accelerator = Accelerator()

6.7.1 Variable Documentation

6.7.1.1 accelerator

```
run.accelerator = Accelerator()
```

6.7.1.2 batch_size

```
run.batch_size = config.getint("training", "batch_size", fallback=128)
```

6.7.1.3 checkpoint_path

```
run.checkpoint_path
```

Initial value:

```
1 =  config.get(
2     "output", "checkpoint_path", fallback="./data/model/checkpoint/"
3 )
```

6.7.1.4 config

```
run.config = cfg.ConfigParser()
```

6.7.1.5 config_id

```
run.config_id = config.getint("output", "config_id", fallback=1234)
```

6.7.1.6 criterion

```
run.criterion = nn.MSELoss()
```

6.7.1.7 dataloader_train

```
run.dataloader_train
```

Initial value:

```
1 =  DataLoader(
2     dataset_train,
3     batch_size=batch_size,
4     shuffle=True,
5     num_workers=4,
6     drop_last=True,
7 )
```

6.7.1.8 dataloader_val

```
run.dataloader_val
```

Initial value:

```
1 =  DataLoader(
2     dataset_val,
3     batch_size=batch_size,
4     shuffle=True,
5     num_workers=4,
6     drop_last=True,
7 )
```

6.7.1.9 dataset

```
run.dataset = MNIST("./data", train=True, download=True, transform=tf)
```

6.7.1.10 dataset_train

```
run.dataset_train
```

6.7.1.11 dataset_val

```
run.dataset_val
```

6.7.1.12 degradation

```
run.degradation = config.get("model", "degradation", fallback="gaussian")
```

6.7.1.13 gt

```
run.gt
```

Initial value:

```
1 =  CNN(
2     in_channels=1, expected_shape=(28, 28), n_hidden=n_hidden, act=nn.GELU
3 )
```

6.7.1.14 input_file

```
run.input_file = sys.argv[1]
```

6.7.1.15 loss_fn

```
run.loss_fn = config.get("model", "loss_fn", fallback="L2")
```

6.7.1.16 lr_initial

```
run.lr_initial = config.getfloat("training", "lr_initial", fallback=2e-4)
```

6.7.1.17 model

```
run.model
```

Initial value:

```
1 =  DDPM(
2         gt=gt, betas=(noise_min, noise_max), n_T=n_T, criterion=criterion
3     )
```

6.7.1.18 n_epoch

```
run.n_epoch = config.getint("training", "n_epoch", fallback=100)
```

6.7.1.19 n_hidden

```
run.n_hidden = config.get("model", "n_hidden", fallback="16 32 32 16")
```

6.7.1.20 n_T

```
run.n_T = config.getint("model", "n_T", fallback=1000)
```

6.7.1.21 noise_max

```
run.noise_max = config.getfloat("model", "noise_max", fallback=0.02)
```

6.7.1.22 noise_min

```
run.noise_min = config.getfloat("model", "noise_min", fallback=1e-4)
```

6.7.1.23 optim

```
run.optim = torch.optim.Adam(model.parameters(), lr=lr_initial)
```

6.7.1.24 sample_path

```
run.sample_path
```

Initial value:

```
1 =  config.get(
2     "output", "sample_path", fallback="./data/model/sample/"
3 )
```

6.7.1.25 save_interval

```
run.save_interval = config.getint("output", "save_interval", fallback=10)
```

6.7.1.26 tf

```
run.tf
```

Initial value:

```
1 =  transforms.Compose(
2     [transforms.ToTensor(), transforms.Normalize((0.5,), (1.0))]
3 )
```

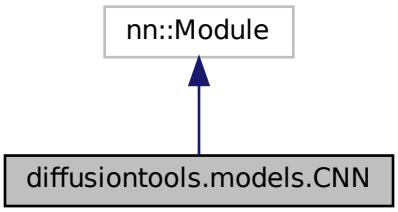
Chapter 7

Class Documentation

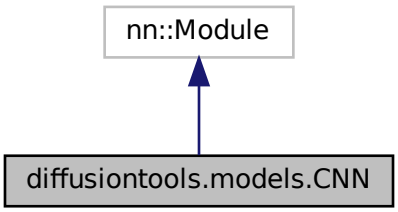
7.1 diffusiontools.models.CNN Class Reference

Defines CNN class used to reconstruct images during diffusion.

Inheritance diagram for diffusiontools.models.CNN:



Collaboration diagram for diffusiontools.models.CNN:



Public Member Functions

- None __init__ (self, in_channels, expected_shape=(28, 28), n_hidden=(64, 128, 64), kernel_size=7, last_↩ kernel_size=3, time_embeddings=16, act=nn.GELU)

  Initialisation for CNN class.
- torch.Tensor time_encoding (self, torch.Tensor t)

  Produces embedding vectors for the time steps passed during the input to the network.
- torch.Tensor forward (self, torch.Tensor x, torch.Tensor t)

  Defines the forward pass for the model.

Public Attributes

- blocks
- time_embed

7.1.1 Detailed Description

Defines CNN class used to reconstruct images during diffusion.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 __init__()

```
None diffusiontools.models.CNN.__init__ (
              self,
              in_channels,
              expected_shape = (28, 28),
              n_hidden = (64, 128, 64),
              kernel_size = 7,
              last_kernel_size = 3,
              time_embeddings = 16,
              act = nn.GELU )
```

Initialisation for CNN class.

Parameters

in_channels	Number of input channels
expected_shape	image shape as (height, width)
n_hidden	Tuple containing number of hidden channels per hidden layer
kernel_size	Convolution kernel size
last_kernel_size	Used to specify different kernel size in final layer
time_embedding	Determines dimensionality of temporal embeddings for diffusion (which is later doubled before combining with the network)
act	Non-linear activation function to use in the hidden neurons

7.1.3 Member Function Documentation

7.1.3.1 forward()

```
torch.Tensor diffusiontools.models.CNN.forward (
              self,
             torch.Tensor x,
             torch.Tensor t )
```

Defines the forward pass for the model.

Parameters

x	Image stack with shape (batch, chan, height, width)
t	Time step stack with shape (batch,)

Returns

embed Final output of the model

7.1.3.2 time_encoding()

```
torch.Tensor diffusiontools.models.CNN.time_encoding (
              self,
             torch.Tensor t )
```

Produces embedding vectors for the time steps passed during the input to the network.

Parameters

t	Tensor of time steps, one for each batch member

Returns

torch.Tensor Embedded vectors that have been passed through trigonometric functions and then learned linear network

7.1.4 Member Data Documentation

7.1.4.1 blocks

```
diffusiontools.models.CNN.blocks
```

7.1.4.2 time_embed

`diffusiontools.models.CNN.time_embed`

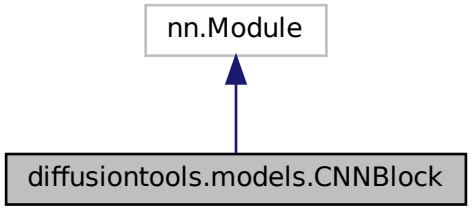
The documentation for this class was generated from the following file:

- /home/jhughes2712/projects/m2_assessment/jh2284/src/diffusiontools/models.py

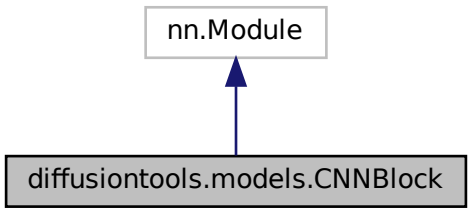
7.2 diffusiontools.models.CNNBlock Class Reference

Class instantiating a convolutional block with layer normalisation and non-linear activation.

Inheritance diagram for diffusiontools.models.CNNBlock:



Collaboration diagram for diffusiontools.models.CNNBlock:



Public Member Functions

- def __init__ (self, in_channels, out_channels, ∗expected_shape, act=nn.GELU, kernel_size=7)

  Initialisation for CNNBlock class.
- def forward (self, x)

  Forward pass function for CNNBlock.

Public Attributes

- net

7.2.1 Detailed Description

Class instantiating a convolutional block with layer normalisation and non-linear activation.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 __init__()

```
def diffusiontools.models.CNNBlock.__init__ (
              self,
              in_channels,
              out_channels,
             * expected_shape,
              act = nn.GELU,
              kernel_size = 7 )
```

Initialisation for CNNBlock class.

Parameters

in_channels	Number of input channels
out_channels	Number of output channels
expected_shape	Image shape (height, width)
act	Non-linear activation function
kernel_size	Size of convolutional kernel to use

7.2.3 Member Function Documentation

7.2.3.1 forward()

```
def diffusiontools.models.CNNBlock.forward (
              self,
              x )
```

Forward pass function for CNNBlock.

Parameters

x	Input from previous hidden layer

Returns

torch.tensor Output to pass to next hidden layer

7.2.4 Member Data Documentation

7.2.4.1 net

`diffusiontools.models.CNNBlock.net`

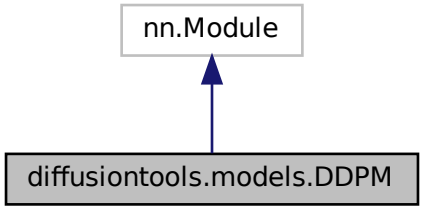
The documentation for this class was generated from the following file:

- /home/jhughes2712/projects/m2_assessment/jh2284/src/diffusiontools/models.py

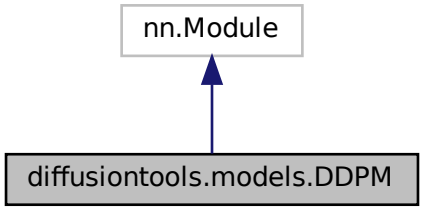
7.3 diffusiontools.models.DDPM Class Reference

Defines Gaussian diffusion model class.

Inheritance diagram for diffusiontools.models.DDPM:



Collaboration diagram for diffusiontools.models.DDPM:



Public Member Functions

- None __init__ (self, gt, Tuple[float, float] betas, int n_T, nn.Module criterion=nn.MSELoss())

  Initialisation for DDPM class.
- torch.Tensor forward (self, torch.Tensor x)

  Forward pass for the DDPM model.
- torch.Tensor sample (self, int n_sample, size, device)

  Reverse diffusion sampling for the DDPM model.

Public Attributes

- gt
- n_T
- criterion

7.3.1 Detailed Description

Defines Gaussian diffusion model class.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 __init__()

```
None diffusiontools.models.DDPM.__init__ (
              self,
              gt,
             Tuple[float, float] betas,
             int n_T,
             nn.Module  criterion = nn.MSELoss() )
```

Initialisation for DDPM class.

Parameters

gt	Trainable model used to approximate the reconstruction process
betas	Determines the extremes of the noise levels along the diffusion process
n_T	Number of discrete diffusion time steps
criterion	Loss function to use for the denoising process

7.3.3 Member Function Documentation

7.3.3.1 forward()

```
torch.Tensor diffusiontools.models.DDPM.forward (
              self,
             torch.Tensor x )
```

Forward pass for the DDPM model.

Implements Algorithm 18.1 in Understanding Deep Learning, found at http://udlbook.com. Note that unusually, this forward method returns the loss function of the predicted error, rather than the error prediction itself.

Parameters

x	Batched input

Returns

torch.Tensor Loss function evaluated on the predicted error compared to the true error

7.3.3.2 sample()

```
torch.Tensor diffusiontools.models.DDPM.sample (
              self,
             int n_sample,
              size,
              device )
```

Reverse diffusion sampling for the DDPM model.

Implements Algorithm 18.2 in Understanding Deep Learning, found at http://udlbook.com.

Parameters

n_sample	Number of images to generate
size	Image size tuple in the format (channel, height, width)

Returns

z_t Stack of generated image samples

7.3.4 Member Data Documentation

7.3.4.1 criterion

```
diffusiontools.models.DDPM.criterion
```

7.3.4.2 gt

`diffusiontools.models.DDPM.gt`

7.3.4.3 n_T

`diffusiontools.models.DDPM.n_T`

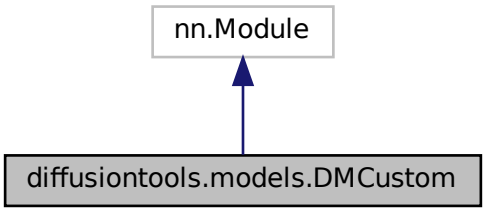
The documentation for this class was generated from the following file:

- /home/jhughes2712/projects/m2_assessment/jh2284/src/diffusiontools/models.py

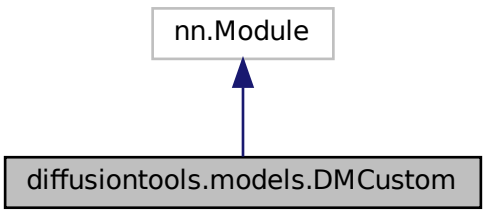
7.4 diffusiontools.models.DMCustom Class Reference

Class defining custom diffusion model.

Inheritance diagram for diffusiontools.models.DMCustom:



Collaboration diagram for diffusiontools.models.DMCustom:



Public Member Functions

- None __init__ (self, gt, Tuple[float, float] alphas, int n_T, Tuple[int, int] size, nn.Module criterion=nn.↩ MSELoss())

  Initialisation for DMCustom class.
- torch.Tensor degrade (self, torch.Tensor x, int t, device)

  Implements custom degradation operation.
- torch.Tensor forward (self, torch.Tensor x)

  Forward pass for the DMCustom model.
- torch.Tensor sample (self, int n_sample, size, device)

  Reverse diffusion sampling for the DMCustom model.

Public Attributes

- gt
- n_T
- criterion
- size

7.4.1 Detailed Description

Class defining custom diffusion model.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 __init__()

```
None diffusiontools.models.DMCustom.__init__ (
              self,
              gt,
             Tuple[float, float] alphas,
             int n_T,
             Tuple[int, int] size,
             nn.Module  criterion = nn.MSELoss() )
```

Initialisation for DMCustom class.

Parameters

gt	Trainable model used to approximate the reconstruction process
alphas	Determines the extremes of the noise levels along the diffusion process
n_T	Number of discrete diffusion time steps
size	Image size tuple in the format (channel, height, width)
criterion	Loss function to use for the denoising process

7.4.3 Member Function Documentation

7.4.3.1 degrade()

```
torch.Tensor diffusiontools.models.DMCustom.degrade (
              self,
             torch.Tensor x,
             int t,
              device )
```

Implements custom degradation operation.

Parameters

x	Batched image tensor
t	Integer discrete time-step; note that the same degradation is applied to all images in the stack
device	torch.device object which enables GPU or CPU to be specified

Returns

z_t Stack of images with degradation applied

7.4.3.2 forward()

```
torch.Tensor diffusiontools.models.DMCustom.forward (
              self,
             torch.Tensor x )
```

Forward pass for the DMCustom model.

Implements Algorithm 18.1 in Understanding Deep Learning, found at http://udlbook.com. Note that unusually, this forward method returns the loss function of the predicted error, rather than the error prediction itself.

Parameters

x	Batched input

Returns

torch.Tensor Loss function evaluated on the predicted error compared to the true error

7.4.3.3 sample()

```
torch.Tensor diffusiontools.models.DMCustom.sample (
               self,
              int n_sample,
               size,
               device )
```

Reverse diffusion sampling for the DMCustom model.

Implements Algorithm 18.2 in Understanding Deep Learning, found at http://udlbook.com.

Parameters

n_sample	Number of images to generate
size	Image size tuple in the format (channel, height, width)

Returns

z_t Stack of generated image samples

7.4.4 Member Data Documentation

7.4.4.1 criterion

```
diffusiontools.models.DMCustom.criterion
```

7.4.4.2 gt

```
diffusiontools.models.DMCustom.gt
```

7.4.4.3 n_T

```
diffusiontools.models.DMCustom.n_T
```

7.4.4.4 size

```
diffusiontools.models.DMCustom.size
```

The documentation for this class was generated from the following file:

- /home/jhughes2712/projects/m2_assessment/jh2284/src/diffusiontools/models.py

Chapter 8

File Documentation

8.1 /home/jhughes2712/projects/m2_↩ assessment/jh2284/src/diffusiontools/__init__.py File Reference

Namespaces

- diffusiontools

8.2 /home/jhughes2712/projects/m2_↩ assessment/jh2284/src/diffusiontools/analysis.py File Reference

Module containing procedures used to analyse and compare trained diffusion models.

Namespaces

- diffusiontools.analysis

Functions

- def diffusiontools.analysis.plot_learning_curve (List[float] losses_train, List[float] losses_val, str title, str filename)

  Plot changes in loss function across epochs for model training.
- def diffusiontools.analysis.compute_image_diffusion (DDPM model, List[float] image_fractions, int n_sample, size, device)

  Save intermediate image generations at specified parts of the Gaussian diffusion process.
- torch.Tensor diffusiontools.analysis.compute_image_diffusion_custom (DMCustom model, List[int] image_↩ fractions, int n_sample, size, device, bool direct=False)

  Save intermediate image generations at specified parts of the custom diffusion process.
- def diffusiontools.analysis.plot_image_diffusion (model, List[int] image_fractions, int n_sample, size, device, str title, str filename)

  Plot intermediate image generations at specified parts of the custom diffusion process.
- def diffusiontools.analysis.plot_samples (np.ndarray samples, int n_row, int n_col, str filename)

Produce matrix of synthetic image samples.

- def diffusiontools.analysis.plot_gt_direct_diffusion (np.ndarray direct_sample, np.ndarray diffusion_sample, int n_plot, str filename)

  Compare direct and diffused reconstructions.
- def diffusiontools.analysis.compute_variance (np.ndarray sample)

  Compute the total variance of a sample of vectors.
- def diffusiontools.analysis.degradation_demo (DMCustom model, device, List[int] t_list, str title, str filename)

  Plot MNIST images of digits 1 to 4 with varying custom degradation levels.
- def diffusiontools.analysis.compute_tsne_kl_div (List[np.ndarray] samples, List[str] labels)

  Map the given samples into a 2D t-SNE embedding alongside the MNIST test data, and compute the KL divergences.
- def diffusiontools.analysis.plot_sample_tsne (np.ndarray density_gt, np.ndarray sample_1_fitted, np.ndarray sample_2_fitted, np.ndarray density_1, np.ndarray density_2, str label_1, str label_2, str filename)

  Plot the t-SNE embeddings of two samples alongside the MNIST test data fitted density.
- def diffusiontools.analysis.plot_mnist_tsne (str filename)

  Plot the t-SNE embeddings of MNIST test data alone, digit classes indicated.

8.2.1 Detailed Description

Module containing procedures used to analyse and compare trained diffusion models.

8.3 /home/jhughes2712/projects/m2_↩ assessment/jh2284/src/diffusiontools/models.py File Reference

Module containing classes and procedures used to build diffusion models.

Classes

- class diffusiontools.models.CNNBlock

  Class instantiating a convolutional block with layer normalisation and non-linear activation.
- class diffusiontools.models.CNN

  Defines CNN class used to reconstruct images during diffusion.
- class diffusiontools.models.DDPM

  Defines Gaussian diffusion model class.
- class diffusiontools.models.DMCustom

  Class defining custom diffusion model.

Namespaces

- diffusiontools.models

Functions

- Dict[str, torch.Tensor] diffusiontools.models.ddpm_schedules (float beta1, float beta2, int T)

  Returns pre-computed schedules for DDPM sampling with a linear noise schedule.

8.3.1 Detailed Description

Module containing classes and procedures used to build diffusion models.

Script used to evaluate the quality of diffusion samples at intermediate stages of training.

Script used to produce plots and statistics related to final trained diffusion models.

8.4 /home/jhughes2712/projects/m2_↩ assessment/jh2284/src/diffusiontools/train.py File Reference

Module containing procedures used to train and save diffusion models.

Namespaces

- diffusiontools.train

Functions

- def diffusiontools.train.train_model (nn.Module ddpm, torch.optim.Optimizer optim, DataLoader dataloader↩ _train, DataLoader dataloader_val, Accelerator accelerator, int n_epoch, int save_interval, str sample_path, str checkpoint_path, str config_id)

  Handles the training process for a diffusion model.

8.4.1 Detailed Description

Module containing procedures used to train and save diffusion models.

8.5 /home/jhughes2712/projects/m2_assessment/jh2284/src/final_↩ analysis.py File Reference

Namespaces

- final_analysis

Variables

- final_analysis.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
- final_analysis.gt_1
- final_analysis.model_1 = DDPM(gt=gt_1, betas=(0.0001, 0.02), n_T=1000).to(device)
- final_analysis.ckpt_0001_0490
- final_analysis.gt_2
- final_analysis.model_2 = DDPM(gt=gt_2, betas=(0.001, 0.2), n_T=100).to(device)
- final_analysis.ckpt_0002_0490
- final_analysis.gt_9
- final_analysis.model_9
- final_analysis.ckpt_0009_0080
- final_analysis.gt_10
- final_analysis.model_10
- final_analysis.ckpt_0010_0180
- final_analysis.ckpt_0001_0120
- final_analysis.ckpt_0002_0080
- final_analysis.ckpt_0009_0020
- final_analysis.ckpt_0010_0140
- int final_analysis.SAMPLE_SIZE = 200
- final_analysis._
- final_analysis.sample_1
- final_analysis.sample_2
- final_analysis.sample_9
- final_analysis.sample_9_direct
- final_analysis.direct
- final_analysis.sample_10
- final_analysis.sample_10_direct
- final_analysis.total_var_1 = compute_variance(sample_1[0])
- final_analysis.total_var_2 = compute_variance(sample_2[0])
- final_analysis.total_var_9 = compute_variance(sample_9[0])
- final_analysis.total_var_10 = compute_variance(sample_10[0])
- final_analysis.density_gt
- final_analysis.samples_fitted
- final_analysis.samples_densities

8.6 /home/jhughes2712/projects/m2_↩ assessment/jh2284/src/intermediate_analysis.py File Reference

Namespaces

- intermediate_analysis

Variables

- intermediate_analysis.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
- intermediate_analysis.ckpt_0001
- intermediate_analysis.gt_1
- intermediate_analysis.model_1 = DDPM(gt=gt_1, betas=(0.0001, 0.02), n_T=1000).to(device)
- intermediate_analysis.ckpt_0002
- intermediate_analysis.gt_2
- intermediate_analysis.model_2 = DDPM(gt=gt_2, betas=(0.0001, 0.02), n_T=1000).to(device)
- intermediate_analysis.ckpt_0009
- intermediate_analysis.gt_9
- intermediate_analysis.model_9
- intermediate_analysis.ckpt_0010
- intermediate_analysis.gt_10
- intermediate_analysis.model_10

8.7 /home/jhughes2712/projects/m2_assessment/jh2284/src/run.py File Reference

Script used to create and train diffusion model, based on parameters provided via the config.ini file.

Namespaces

- run

Variables

- run.config = cfg.ConfigParser()
- run.input_file = sys.argv[1]
- run.noise_min = config.getfloat("model", "noise_min", fallback=1e-4)
- run.noise_max = config.getfloat("model", "noise_max", fallback=0.02)
- run.n_T = config.getint("model", "n_T", fallback=1000)
- run.degradation = config.get("model", "degradation", fallback="gaussian")
- run.n_hidden = config.get("model", "n_hidden", fallback="16 32 32 16")
- run.loss_fn = config.get("model", "loss_fn", fallback="L2")
- run.n_epoch = config.getint("training", "n_epoch", fallback=100)
- run.batch_size = config.getint("training", "batch_size", fallback=128)
- run.lr_initial = config.getfloat("training", "lr_initial", fallback=2e-4)
- run.checkpoint_path
- run.sample_path
- run.save_interval = config.getint("output", "save_interval", fallback=10)
- run.config_id = config.getint("output", "config_id", fallback=1234)
- run.tf
- run.dataset = MNIST("./data", train=True, download=True, transform=tf)
- run.dataset_train
- run.dataset_val
- run.dataloader_train
- run.dataloader_val
- run.gt
- run.criterion = nn.MSELoss()
- run.model
- run.optim = torch.optim.Adam(model.parameters(), lr=lr_initial)
- run.accelerator = Accelerator()

8.7.1 Detailed Description

Script used to create and train diffusion model, based on parameters provided via the config.ini file.

Author

Created by J. Hughes on 18/03/2024.

Index