Module `train`

▸ EXPAND SOURCE CODE

Functions

```
def generate_new_images(ddpm, n_samples=16, device=None, frames_per_gif=100,
gif_name='sampling.gif', c=1, h=256, w=256, gif=True)
```

Generates new images using the given DDPM model.

Args

`ddpm` : `MyDDPM`
: The DDPM model.

`n_samples` : `int`, optional
: The number of samples to generate. Defaults to 16.

`device` : `str`, optional
: The device for computation. Defaults to None.

`frames_per_gif` : `int`, optional
: The number of frames per GIF. Defaults to 100.

`gif_name` : `str`, optional
: The name of the GIF file. Defaults to "sampling.gif".

`c` : `int`, optional
: The number of channels in the images. Defaults to channel.

`h` : `int`, optional
: The height of the images. Defaults to img_size.

`w` : `int`, optional
: The width of the images. Defaults to img_size.

`gif` : `bool`, optional
: Whether to generate a GIF. Defaults to True.

Returns

`torch.Tensor`
: The generated images.

Description

This function generates new images using the given DDPM model. It starts by initializing the images with random noise. Then, it iterates over the time steps in reverse order. For each time step, it estimates the noise to be removed and performs partial denoising of the images. If the time step is greater than 0, it adds some more noise to the images. After denoising, it adds frames to the GIF if the gif flag is set to True and the current index is in the frame indices or the time step is 0. The frames are stored in a list. Finally, if the gif flag is set to True, the GIF is stored as a file.

▸ EXPAND SOURCE CODE

```
def get_config()
```

Reads the configuration from 'config.ini' and returns it as a dictionary.

:return: A dictionary containing the configurations. :rtype: dict

▸ EXPAND SOURCE CODE

```
def make_path(path)
```

A function that creates a directory at the specified path if it does not already exist.

Args: - path (str): The path where the directory will be created.

Returns: - None

▸ EXPAND SOURCE CODE

```
def save_images(images, path='output', prefix='output', n_images=1)
```

Saves a specified number of images to a given path with a specified prefix.

Parameters

images (torch.Tensor): The tensor containing the images to be saved. path (str, optional): The path to save the images. Defaults to "output". prefix (str, optional): The prefix to be added to the saved image filenames. Defaults to "output". n_images (int, optional): The number of images to be saved. Defaults to 1.

Returns

None

▸ EXPAND SOURCE CODE

```
def show_first_batch(loader)
```

Shows the first batch of images.

Parameters: - loader: The data loader containing the batches of images.

Returns: None

▸ EXPAND SOURCE CODE

```
def show_forward(ddpm, loader, device)
```

Shows the forward process.

Parameters

ddpm: The ddpm model. loader: The data loader. device: The device for computation.

Returns

None

▸ EXPAND SOURCE CODE

```
def show_images(images, title,
path='debug_img')
```

Show images in a grid layout with a given title and save the resulting figure as a PNG file.

Parameters: - images (torch.Tensor or numpy.ndarray): The images to be displayed. If a torch.Tensor is provided, it will be converted to a numpy array. - title (str): The title of the figure. - path (str, optional): The path to save the figure. Defaults to img_path.

Returns: None

▸ EXPAND SOURCE CODE

```
def training_loop(ddpm, loader, n_epochs, optim, device=device(type='cpu'), debug=False,
store_path='model.pt')
```

Trains a DDPM model on a given dataset using the specified optimizer and number of epochs.

Args

`ddpm` : `DDPM`
: The DDPM model to be trained.

`loader` : `DataLoader`
: The data loader containing the training dataset.

`n_epochs` : `int`
: The number of epochs to train the model.

`optim` : `Optimizer`
: The optimizer to use for training.

`device` : `str` , optional
: The device to use for training. Defaults to the global device variable.

`debug` : `bool` , optional
: Whether to enable debug mode. Defaults to the global debug variable.

`store_path` : `str` , optional
: The path to store the trained model. Defaults to the global model_path variable.

Returns

None

Raises

None Side Effects: - Trains the DDPM model on the specified dataset. - Stores the trained model at the specified store_path if it is the best model encountered so far. - Saves the training loss plot at img_path/plot/plot_loss.png if debug mode is enabled. - Saves the generated images at img_path/gif/sampling.gif and img_path/epoch/{epoch + 1} if debug mode is enabled. - Prints the loss at each epoch and whether the model is the best encountered so far.

▸ EXPAND SOURCE CODE

Classes

```
class MyDDPM (network, n_steps=500, min_beta=0.0001, max_beta=0.02, device=None, image_chw=(1, 256,
256))
```

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes::

```
 import torch.nn as nn
import torch.nn.functional as F
```

```
class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call :meth:`to`, etc.

Note

As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

:ivar training: Boolean represents whether this module is in training or evaluation mode. :vartype training: bool

Initializes a new instance of the MyDDPM class.

Args

`network` : `torch.nn.Module`
: The neural network model used for denoising.

`n_steps` : `int` , optional
: The number of steps in the beta schedule. Defaults to n_steps.

`min_beta` : `float` , optional
: The minimum value of the beta schedule. Defaults to min_beta.

`max_beta` : `float` , optional
: The maximum value of the beta schedule. Defaults to max_beta.

`device` : `str` , optional
: The device to run the computations on. Defaults to None.

`image_chw` : `tuple` , optional
: The shape of the input images. Defaults to (channel, img_size, img_size).

Returns

None

▸ EXPAND SOURCE CODE

Ancestors

torch.nn.modules.module.Module

Methods

`def backward(self, x, t)`

Run each image through the network for each timestep t in the vector t. The network returns its estimation of the noise that was added.

Parameters

x (torch.Tensor): The input image tensor of shape (n, c, h, w). t (torch.Tensor): The time step tensor of shape (n,).

Returns

`torch.Tensor`
: The output tensor of shape (n, c, h, w).

▶ EXPAND SOURCE CODE

```
def forward(self, x0, t, eta=None)
-> Callable[..., Any]
```

Applies the forward pass of the DDPM model to generate noisy images.

Args

`x0` : `torch.Tensor`
: The input image tensor of shape (n, c, h, w).

`t` : `int`
: The time step of the DDPM model.

`eta` : `torch.Tensor` , optional
: The noise tensor of shape (n, c, h, w). If not provided, a random noise tensor is generated.

Returns

`torch.Tensor`
: The noisy image tensor of shape (n, c, h, w).

▶ EXPAND SOURCE CODE

```
class SquarePad
```

▶ EXPAND SOURCE CODE

Index

Functions