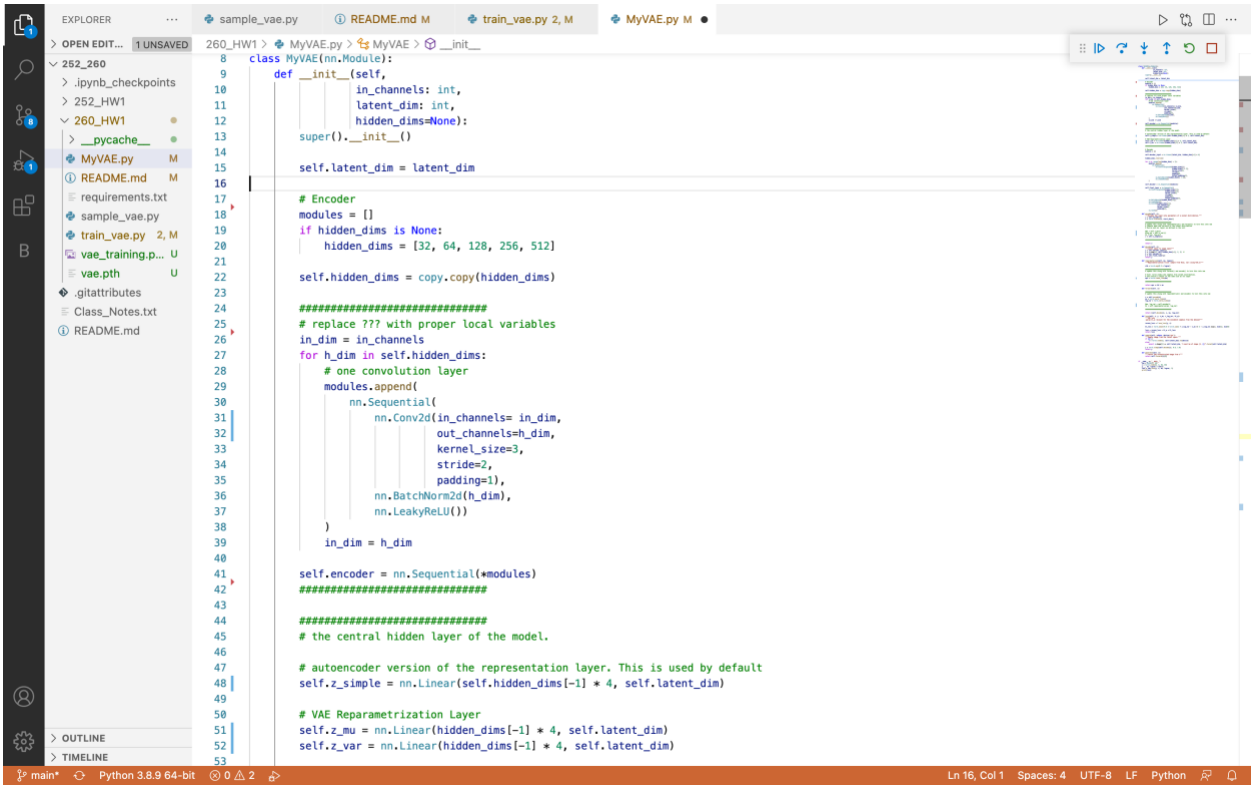
CMPE 260 – Reinforcement Learning (Spring 2023)
Homework1

Submitted by: Kavan Vipinbhai Soni (016085966)

Updated code at : https://github.com/kavan-soni/252_260/tree/main/260_HW1

Activity 1 :

Finish the `__init__()` in `MyVAE.py` model.

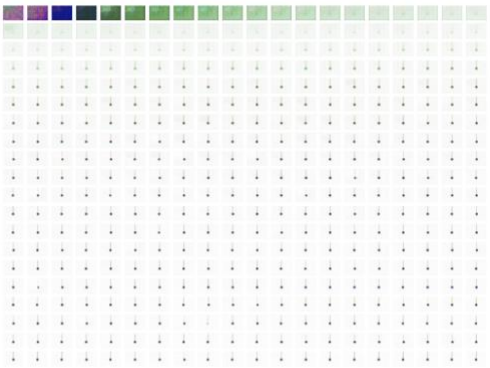


Run `MyVAE.py` to quickly test if your model is working

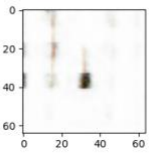
```
(a1) kavansoni@Kavans-MacBook-Air 260_HW1 % python3 MyVAE.py
[W NNPACK.cpp:51] Could not initialize NNPACK! Reason: Unsupported hardware.
tensor(1.1228, grad_fn=<AddBackward0>)
(a1) kavansoni@Kavans-MacBook-Air 260_HW1 %
```

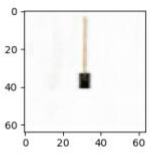
Run `train_vae.py` to train.

```
(a1) kavansoni@Kavans-MacBook-Air 260_HW1 % python3 train_vae.py
[W NNPACK.cpp:51] Could not initialize NNPACK! Reason: Unsupported hardware.
tensor(1.2485, grad_fn=<AddBackward0>)
tensor(0.9843, grad_fn=<AddBackward0>)
tensor(0.8815, grad_fn=<AddBackward0>)
tensor(0.7347, grad_fn=<AddBackward0>)
tensor(0.5662, grad_fn=<AddBackward0>)
tensor(0.4341, grad_fn=<AddBackward0>)
tensor(0.3287, grad_fn=<AddBackward0>)
```



Run `sample_vae.py` to generate a few images with your model. Save 2 generated images.





What model components are used in the forward pass and in sampling?

- Following components are used in forward pass and sampling:
  - Encoder
  - Latent space
  - Decoder
  - Image sampling

Activity 2:

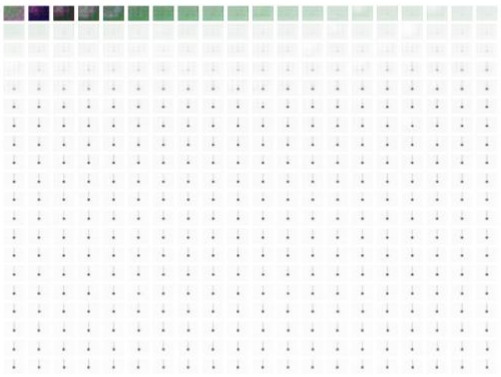
Upgrade to VAE by modifying `forward()`, `encode()`, and `reparameterize()`.

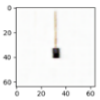
```python
def reparameterize(self, mu, logvar):
    """Reparameterization trick: sample from N(mu, var) using N(0,1)"""

    std = torch.exp(0.5 * logvar)

    ##############################
    # update this along with forward() and encode() to turn this into vae

    # hint: torch.randn_like samples from normal distribution,
    # and returns a tensor of the same size as its input
    eps = torch.randn_like(mu)

    ##############################

    return eps * std + mu
```

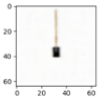
```python
def forward(self, x):

    ##############################
    # update this along with reparametrize() and encode() to turn this into vae

    #z = self.encode(x)
    #mu = torch.zeros_like(z)
    #log_var = torch.zeros_like(z)

    mu, log_var = self.encode(x)
    z = self.reparameterize(mu, log_var)

    ##############################

    return [self.decode(z), x, mu, log_var]
```

```python
def encode(self, x):
    """Encodes the input into parameters of a normal distribution."""
    z = self.encoder(x)
    z = torch.flatten(z, start_dim=1)

    ##############################
    # update this along with reparameterize() and forward() to turn this into vae
    # Compute mean and variance of the latent distribution
    # Use mu and var layers we defined in the init

    mu = self.z_mu(z)
    log_var = self.z_var(z)
    z = [mu, log_var]
    #z = self.z_simple(z)

    ##############################

    return z
```

Train and save two generated images.







Describe the difference between the AE and VAE models.

- AE and VAE are both neural networks used for unsupervised learning and dimensionality reduction. They differ in following ways:
    - VAE is generative model while AE is not.
    - Latent space of VAE is continuous and structured, while it is unstructured for AE.
    - AE do not use regularization, while VAE uses Kullback-Leibler divergence for regularization of latent space.
    - AE only use reconstruction loss, while VAE uses reconstruction loss and KL divergence to assess how well the model can reconstruct the input data.

What is the reparameterization trick?

- The reparameterization trick is a technique used in Variational Autoencoder (VAE) models to sample from a probability distribution in a differentiable way. The reparameterization trick enables the gradient-based optimization of VAE models, which makes it possible to train these models using backpropagation and stochastic gradient descent.
- In a VAE, the goal is to learn a latent space that follows a specific probability distribution, typically a normal distribution. During training, the VAE learns the parameters of this distribution, namely the mean and variance, from the input data. To generate new samples from this distribution, the VAE must sample from it.
- However, sampling from a probability distribution is a non-differentiable operation, which makes it difficult to use gradient-based optimization techniques for training the VAE. The reparameterization trick solves this problem by separating the sampling operation from the parameters of the distribution.
- Specifically, the trick involves introducing a noise variable epsilon that is drawn from a fixed standard normal distribution. The mean and variance parameters learned by the VAE are then used to transform this noise variable into a sample from the desired distribution, as follows:

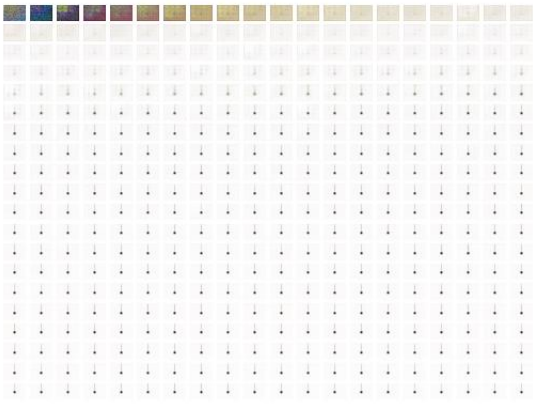
  z = mu + sigma * epsilon
- This transformation is differentiable with respect to the mean and standard deviation, which allows for the gradient-based optimization of the VAE during training.
- In summary, the reparameterization trick is a technique used in VAE models to enable the differentiable sampling from a probability distribution, which is required for gradient-based optimization during training.

Activity 3:

Update the `train_vae.py` to reset the environment after the first 20 observations from each episode.

```python
# episode frame counter
frame_idx = 0

for i in range(training_size):
    frame_idx += 1

    if frame_idx == 20 : env.reset()

    # get a random action in this environment
    action = env.action_space.sample()

    # obs is observation data from the env.
    # Look at the gym code to find which one is a pole angle.
    # https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py
    obs, reward, terminated, truncated, info = env.step(action)
```

Train and save two generated images.

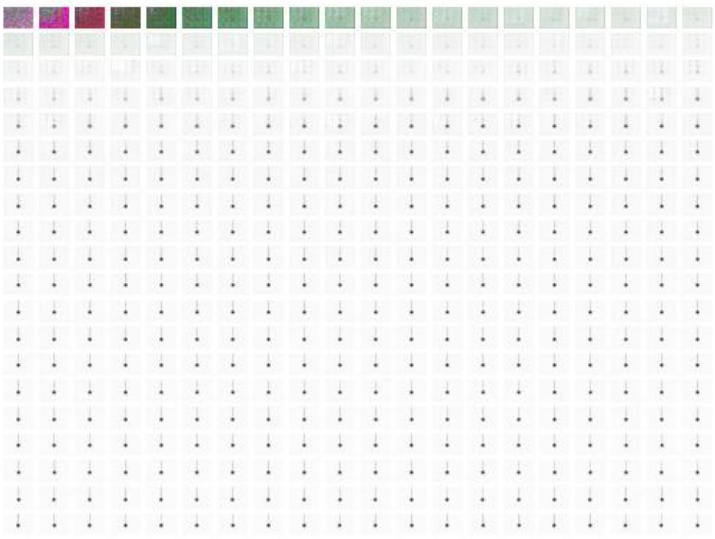


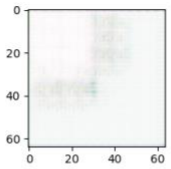
Activity 4:

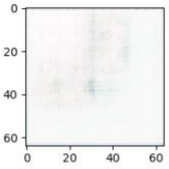
update the `train_vae.py` train vae on observations with a custom angle range.

```
# add some conditional logic to save the images you need
# collect data
# collect data only if pole angle in range [-0.3 rad, 0.3 rad]
if -0.3 <= obs[2] <= 0.3 :
    imgs[i] = img
```

Train and save two generated images.







Generate

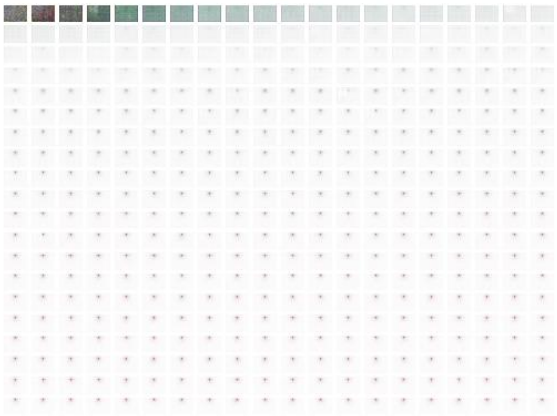
Generate

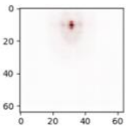
Activity 5:

pick [some other gym environment] and train vae on it.

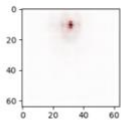
```
def train_vae():

    # initialize the gym environment
    ########################

    # try different environments
    env = gym.make("Pendulum-v1", render_mode='rgb_array')
```

Train and save two generated images.





Generate



Generate