Maching Learning & Data Mining HW4

21307289 刘森元

Task. 1

在`utils.py`的`sample_gaussian`函数中实现重参数化技巧。

实现如下:

```
def sample_gaussian(m, v):
    """
    Element-wise application reparameterization trick to sample from Gaussian

    Args:
        m: tensor: (batch, ...): Mean
        v: tensor: (batch, ...): Variance

    Return:
        z: tensor: (batch, ...): Samples
    """
    z = torch.distributions.normal.Normal(m,torch.sqrt(v)).rsample()
    return z
```

这里直接使用到了`torch`模块中的用于表示正态分布的类`torch.distributions.normal.Normal()`，只需手动指定均值和标准差。

我们需要实现的是高斯分布的重参数化，因此直接将函数传入的`m`和`sqrt(v)`传入即可。最后的`.rsample()`即按照制定的均值和标准差生成一个样本`z`.

Task. 2

在`vae.py`中实现 ELBO 下界。

本题只需要修改 `vae.py` 中的`negative_elbo_bound()`，具体实现如下:

```
    def negative_elbo_bound(self, x):
        """
        Computes the Evidence Lower Bound, KL and, Reconstruction costs

        Args:
            x: tensor: (batch, dim): Observations
```

```
        Returns:
            nelbo: tensor: (): Negative evidence lower bound
            kl: tensor: (): ELBO KL divergence to prior
            rec: tensor: (): ELBO Reconstruction term
        """
        # 使用 encoder 计算潜在变量 z 的均值和方差
        # 这里使用的是类中绑定的编码器。
        latent_mean, latent_variance = self.enc.encode(x)

        # 计算先验分布的均值和方差
        prior_mean = self.z_prior[0].expand(latent_mean.shape)
        prior_variance = self.z_prior[1].expand(latent_variance.shape)

        # 这里使用的是 utils.py 中的 kl_normal() 函数
        # 传入潜变量 z 的均值和方差，以及先验分布的均值和方差
        # 即可计算出kl散度
        kl_divergences = ut.kl_normal(latent_mean, latent_variance, prior_mean,
prior_variance)
        kl_divergence = torch.mean(kl_divergences)

        # 通过重参数化技巧，从潜在变量 z 的分布中抽样
        sampled_latent = ut.sample_gaussian(latent_mean, latent_variance)

        # 使用 decoder 生成重构概率
        reconstructed_probs = self.dec.decode(sampled_latent)

        # 计算重构损失，即负的重构概率的平均值
        # 这是了使用到了 utils.py 中定义的 log_bernoulli_with_logits函数
        # 给定 Bernoulli 分布的 logits 就可以计算样本的对数概率
        reconstruction_losses = ut.log_bernoulli_with_logits(x,
reconstructed_probs)
        reconstruction_loss = -torch.mean(reconstruction_losses)

        # 计算负的证据下界
        negative_elbo = kl_divergence + reconstruction_loss


        return negative_elbo, kl_divergence, reconstruction_loss
```

关于代码中一些引用的部分，见下方：

`VAE` 类提供构造函数，用以使用制定的神经网络结构创建编码器和解码器。文件已经被放在了 `/nn/models/nns/` 目录下，我们不需要深入了解。

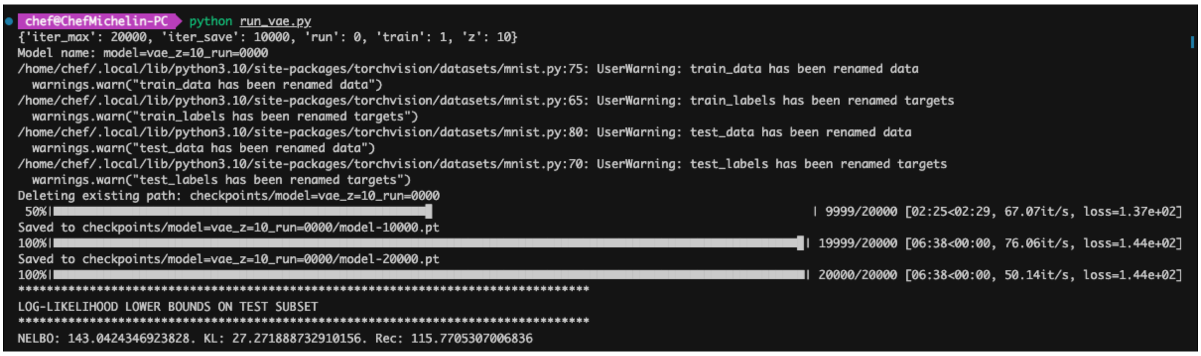
```python
        nn = getattr(nns, nn)
        self.enc = nn.Encoder(self.z_dim)
        self.dec = nn.Decoder(self.z_dim)
```

然后是构造函数中已经将先验设定为固定参数。

```python
        self.z_prior_m = torch.nn.Parameter(torch.zeros(1), requires_grad=False)
        self.z_prior_v = torch.nn.Parameter(torch.ones(1), requires_grad=False)
        self.z_prior = (self.z_prior_m, self.z_prior_v)
```

Task. 3

使用`run_vae.py`进行测试。



```
chef@ChefMichelin-PC  python run_vae.py
{'iter_max': 20000, 'iter_save': 10000, 'run': 0, 'train': 1, 'z': 10}
Model name: model=vae_z=10_run=0000
/home/chef/.local/lib/python3.10/site-packages/torchvision/datasets/mnist.py:75: UserWarning: train_data has been renamed data
  warnings.warn("train_data has been renamed data")
/home/chef/.local/lib/python3.10/site-packages/torchvision/datasets/mnist.py:65: UserWarning: train_labels has been renamed targets
  warnings.warn("train_labels has been renamed targets")
/home/chef/.local/lib/python3.10/site-packages/torchvision/datasets/mnist.py:80: UserWarning: test_data has been renamed data
  warnings.warn("test_data has been renamed data")
/home/chef/.local/lib/python3.10/site-packages/torchvision/datasets/mnist.py:70: UserWarning: test_labels has been renamed targets
  warnings.warn("test_labels has been renamed targets")
Deleting existing path: checkpoints/model=vae_z=10_run=0000
 50%|                                        | 9999/20000 [02:25<02:29, 67.07it/s, loss=1.37e+02]
Saved to checkpoints/model=vae_z=10_run=0000/model-10000.pt
100%|                                        | 19999/20000 [06:38<00:00, 76.06it/s, loss=1.44e+02]
Saved to checkpoints/model=vae_z=10_run=0000/model-20000.pt
100%|                                        | 20000/20000 [06:38<00:00, 50.14it/s, loss=1.44e+02]
********************************************************************************
LOG-LIKELIHOOD LOWER BOUNDS ON TEST SUBSET
********************************************************************************
NELBO: 143.0424346923828. KL: 27.271888732910156. Rec: 115.7705307006836
```

	NELBO	KL	REC	TRAINING TIME
#1	100.9282	19.4576	81.4706	05:54
#2	100.1084	19.4555	80.6529	11:50
#3	99.8885	19.2134	80.6751	12:05
#4	100.7291	19.4834	81.2459	10:05
#5	100.5239	19.1896	81.3343	11:27
#6	100.3499	19.3172	81.0326	7:27
#7	99.8592	19.3352	80.5240	12:06
#8	100.0234	19.0390	80.9844	8:24
#9	99.8293	19.3085	80.5208	11:27
#10	99.3737	18.9144	80.4593	11:28
Average	100.16136	19.27138	80.88999	x

可以看到，负ELBO都在100左右，对应实验任务中的提示，可以知道本次程序的实现基本成功。另外，100的负ELBO对应的KL，以及REC也比较稳定。

另外，每次训练的平均时间可以看到在 10min 左右。

Task. 4

为了实现可视化，我们需要对`run_vae.py`进行一定的修改。

首先，我们加入实现可视化功能的函数。

```
def visualize_samples(model, num_samples=200, grid_size=(10, 20), save_path=None):
    """
    Generate and visualize samples from the VAE.

    Args:
        model: VAE: Trained VAE model
        num_samples: int: Number of samples to generate
        grid_size: tuple: Grid size for visualization (rows, columns)
        save_path: str or None: Path to save the visualization image (if None,
show the plot)
    """
    model.eval()
```

```python
    with torch.no_grad():
        z_samples = torch.randn(num_samples, model.z_dim).to(device)
        generated_samples = model.sample_x_given(z_samples).cpu().view(-1, 28, 28)

    fig, axes = plt.subplots(*grid_size, figsize=(15, 7.5))
    for i in range(num_samples):
        row, col = divmod(i, grid_size[1])
        axes[row, col].imshow(generated_samples[i], cmap='gray')
        axes[row, col].axis('off')

    if save_path:
        plt.savefig(save_path)
        print(f"Visualization saved at {save_path}")
        plt.show()
    else:
        plt.show()
```

然后，我们在VAE训练结束之后，以我们训练好的模型作为参数调用`visualize_samples()`，即可实现结果的可视化。

```python
# 使用已训练的模型进行可视化
visualize_samples(vae, num_samples=200, grid_size=(10, 20),
save_path='./visualizing_result.png')
```

最终可视化结果如下：