Environment

pytorch	1.5.1
torchvision	0.6.1
numpy	1.18.5
matplotlib	3.3.0

Main usage of class

Take in samples in dictionary form: {'joint': (x, z), 'marginal': x}, where (x, z) and x are all 'torch.float32' type. And return the mutual information neural estimation of the inputs. For example:

```python
import torch
import numpy as np
from Mine import MINE

# define data (x, z) in multi_normal distribution with correlation 0.4
data = np.random.multivariate_normal(mean=[0, 0],
                                     cov=[[1, 0.4], [0.4, 1]],
                                     size=10000)


def _totorch(x):
    """
    other type to torch.float32
    """
    return torch.tensor(x, dtype=torch.float32) \
        if type(x) is not torch.float32 else x


# sample minibatch data from dataset
def sample_batch(data, bs):
    data_batch = {}
    rn_joint = np.random.choice(range(data.shape[0]), size=bs, replace=False)
    data_batch['joint'] = _totorch(data[rn_joint, :])
    rn_marginal = np.random.choice(range(data.shape[0]), size=bs, replace=False)
    data_batch['marginal'] = _totorch(data[rn_marginal, 1])
```

```
    return data_batch


mine = MINE(num_input=2,
            mode='fGAN',
            measure='JSD',
            lr=1e-3,
            device='cpu')
data_batch = sample_batch(data, bs=64)

loss, lb = mine.get_loss(data_batch)
print(loss)
```

with output:

```
tensor(0.0001, grad_fn=<AddBackward0>)
```

see more function parameters details in MINE.py

support material under different mode

DV

根据KL散度的Donsker-Varadhan表示

$$D_{KL}(\mathbb{P}||\mathbb{Q}) = \sup_{T:\Omega\to\mathbb{R}} \mathbb{E}_{\mathbb{P}}[T] - \log \mathbb{E}_{\mathbb{Q}}[e^T]$$

不断推进这个函数的上界就可以得到KL-散度的估计值，算法的伪代码如下所示

Algorithm 1 MINE

$\theta \leftarrow$ initialize network parameters
repeat
Draw b minibatch samples from the joint distribution:
$(\boldsymbol{x}^{(1)}, \boldsymbol{z}^{(1)}), \ldots, (\boldsymbol{x}^{(b)}, \boldsymbol{z}^{(b)}) \sim \mathbb{P}_{XZ}$
Draw n samples from the Z marginal distribution:
$\bar{\boldsymbol{z}}^{(1)}, \ldots, \bar{\boldsymbol{z}}^{(b)} \sim \mathbb{P}_Z$
Evaluate the lower-bound:
$\mathcal{V}(\theta) \leftarrow \frac{1}{b}\sum_{i=1}^{b} T_\theta(\boldsymbol{x}^{(i)}, \boldsymbol{z}^{(i)}) - \log(\frac{1}{b}\sum_{i=1}^{b} e^{T_\theta(\boldsymbol{x}^{(i)}, \bar{\boldsymbol{z}}^{(i)})})$
Evaluate bias corrected gradients (e.g., moving average):
$\widehat{G}(\theta) \leftarrow \widetilde{\nabla}_\theta \mathcal{V}(\theta)$
Update the statistics network parameters:
$\theta \leftarrow \theta + \widehat{G}(\theta)$
until convergence

其中(x, z)为代码中的sample['joint']，z为代码中的sample['marginal']。为了稳定训练，可以使用移动平均(measure='ma')来对梯度进行修正：

$$e^T_{ma} = (1-\gamma)e^T_{ma} + \gamma e^T$$
$$\mathcal{L} = -\frac{1}{b}\sum T(x, z) + \frac{\sum e^T}{\sum e^T_{ma}}$$

其中γ是移动平均的控制量，对应代码中的'ma_rate'。更多细节请看论文：

https://arxiv.org/pdf/1801.04062.pdf

fGAN

除了DV下界，也可以使用f-散度对互信息进行估计：

$$\mathcal{L}(\theta, \omega) = \mathbb{E}_{x\sim P}[g_f(V_\omega(x))] + \mathbb{E}_{x\sim Q_\theta}[-f^*(g_f(V_\omega(x)))]$$

对于不同的f-散度，选择不同的激活函数g_f和共轭函数f^*即可计算不同的fGAN。

Name	Output activation g_f	dom_{f^*}	Conjugate $f^*(t)$	$f'(1)$
Kullback-Leibler (KL)	v	$\mathbb{R}$	$\exp(t-1)$	1
Reverse KL	$-\exp(-v)$	$\mathbb{R}_-$	$-1-\log(-t)$	-1
Pearson χ^2	v	$\mathbb{R}$	$\frac{1}{4}t^2+t$	0
Squared Hellinger	$1-\exp(-v)$	$t<1$	$\frac{t}{1-t}$	0
Jensen-Shannon	$\log(2)-\log(1+\exp(-v))$	$t<\log(2)$	$-\log(2-\exp(t))$	0
GAN	$-\log(1+\exp(-v))$	$\mathbb{R}_-$	$-\log(1-\exp(t))$	$-\log(2)$

更多细节请看论文：https://arxiv.org/pdf/1606.00709.pdf

infoNCE

也可以用NLP中常用的NCE loss，将此预测问题转化为二分类问题：

$$\hat{I}_{\omega,\psi}^{(\text{info}NCE)} := \mathbb{E}_{\mathbb{P}}[T_{\omega,\psi}(x, E_\psi(x)) - \mathbb{E}_{\tilde{\mathbb{P}}}[\log \sum_{x'} e^{T_{\omega,\psi}(x', E_\psi(x))}]]$$

(Note: 此部分代码测试不完全，可能存在问题，谨慎使用。)

更多细节请看论文：https://arxiv.org/pdf/1808.06670.pdf

todo

- [ ] 更多关于infoNCE的测试
- [ ] 添加互信息至GAN的测试demo

更多有关互信息的内容也可访问我的csdn：https://blog.csdn.net/qq_39337332/article/details/108583901