Reparameterization of common distributions

We will work with Torch throughout this notebook.

```
In [ ]: import torch
        from torch.distributions import Exponential, Categorical, Gumbel #, ...  import the distributions you need here
        from torch.nn import functional as F
```

A helper function to visualize the generated samples:

```
In [ ]: import matplotlib.pyplot as plt
        def compare_samples (samples_1, samples_2, bins=100, range=None):
          fig = plt.figure()
          if range is not None:
            plt.hist(samples_1, bins=bins, range=range, alpha=0.5)
            plt.hist(samples_2, bins=bins, range=range, alpha=0.5)
          else:
            plt.hist(samples_1, bins=bins, alpha=0.5)
            plt.hist(samples_2, bins=bins, alpha=0.5)
          plt.xlabel('value')
          plt.ylabel('number of samples')
          plt.legend(['direct','via reparameterization'])
          plt.show()
```

Q1. Exponential Distribution

Below write a function that generates N samples from $Exp(\lambda)$.
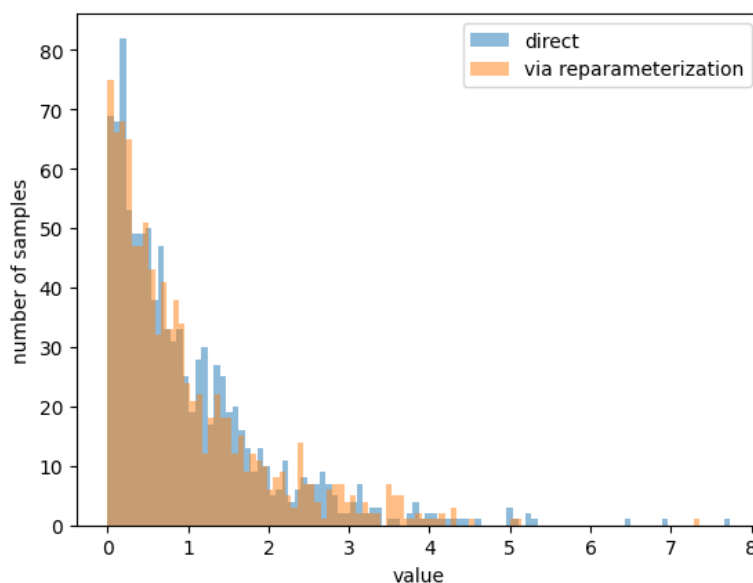
```
In [ ]: def exp_sampler(l, N):
          # insert your code
          samples = Exponential(l).sample((N,))
          return samples # should be N-by-1
```

Now, implement the reparameterization trick:

```
In [ ]: def exp_reparametrize(l,N):
          # this function should return N samples via reparametrization,
          # insert your code
          samples = -1/l *torch.log(1-torch.rand(N,))
          return samples
```

Generate samples for $\lambda = 1$ and compare:

```
In [ ]: l = 1      #lambda
        N = 1000
        direct_samples = exp_sampler(l, N)
        reparametrized_samples = exp_reparametrize(l, N)
        direct_samples = direct_samples.detach().numpy()
        reparametrized_samples = reparametrized_samples.detach().numpy()
        compare_samples(direct_samples, reparametrized_samples)
```


Q2. Categorical Distribution

Below write a function that generates N samples from Categorical (**a**), where $\mathbf{a} = [a_0, a_1, a_2, a_3]$.

```
In [ ]: def categorical_sampler(a, N):
            # insert your code
            samples = Categorical(a).sample((N,))
            return samples  # should be N-by-1
```

Now write a function that generates samples from Categorical (**a**) via reparameterization:

```
In [ ]: # Hint: approximate the Categorical distribution with the Gumbel-Softmax distribution
        def categorical_reparametrize(a, N, temp=0.1, eps=1e-20):  # temp and eps are hyperparameters for Gumbel-Softmax
            # insert your code
            samples = Gumbel(0,1).sample((N,a.shape[0]))
            samples = F.softmax((samples + torch.log(a+eps))/temp, dim=1)

            samples = torch.argmax(samples, dim=1)

            return samples # make sure that your implementation allows the gradient to backpropagate
```

Generate samples when $a = [0.1, 0.2, 0.5, 0.2]$ and visualize them:

```
In [ ]: a = torch.tensor([0.1,0.2,0.5,0.2])
        N = 1000
        direct_samples = categorical_sampler(a, N)
        reparametrized_samples = categorical_reparametrize(a, N, temp=0.1, eps=1e-20)
        direct_samples = direct_samples.detach().numpy()
        reparametrized_samples = reparametrized_samples.detach().numpy()
        compare_samples(direct_samples, reparametrized_samples)
```