tf.contrib.bayesflow.monte_carlo.expectation

```
expectation(
    f,
    samples,
    log_prob=None,
    use_reparametrization=True,
    axis=0,
    keep_dims=False,
    name=None
)
```

Defined in `tensorflow/contrib/bayesflow/python/ops/monte_carlo_impl.py`.

See the guide: BayesFlow Monte Carlo (contrib) > Ops

Computes the Monte-Carlo approximation of `E_p[f(X)]`.

This function computes the Monte-Carlo approximation of an expectation, i.e.,

```
E_p[f(X)] approx= m**-1 sum_i^m f(x_j),  x_j ~iid p(X)
```

where:

- `x_j = samples[j, ...]`,
- `log(p(samples)) = log_prob(samples)` and
- `m = prod(shape(samples)[axis])`.

Tricks: Reparameterization and Score-Gradient

When p is "reparameterized", i.e., a diffeomorphic transformation of a parameterless distribution (e.g., `Normal(Y; m, s) <=> Y = sX + m, X ~ Normal(0,1)`), we can swap gradient and expectation, i.e., `grad[ Avg{ s_i : i=1...n } ] = Avg{ grad[s_i] : i=1...n }` where `S_n = Avg{s_i}` and `s_i = f(x_i), x_i ~ p`.

However, if p is not reparameterized, TensorFlow's gradient will be incorrect since the chain-rule stops at samples of non-reparameterized distributions. (The non-differentiated result, `approx_expectation`, is the same regardless of `use_reparametrization`.) In this circumstance using the Score-Gradient trick results in an unbiased gradient, i.e.,

```
grad[ E_p[f(X)] ]
= grad[ int dx p(x) f(x) ]
= int dx grad[ p(x) f(x) ]
= int dx [ p'(x) f(x) + p(x) f'(x) ]
= int dx p(x) [p'(x) / p(x) f(x) + f'(x) ]
= int dx p(x) grad[ f(x) p(x) / stop_grad[p(x)] ]
= E_p[ grad[ f(x) p(x) / stop_grad[p(x)] ] ]
```

Unless p is not reparametrized, it is usually preferable to `use_reparametrization = True`.

Warning: users are responsible for verifying p is a "reparameterized" distribution.

Example Use:

```
bf = tf.contrib.bayesflow
ds = tf.contrib.distributions

# Monte-Carlo approximation of a reparameterized distribution, e.g., Normal.

num_draws = int(1e5)
p = ds.Normal(loc=0., scale=1.)
q = ds.Normal(loc=1., scale=2.)
exact_kl_normal_normal = ds.kl_divergence(p, q)
# ==> 0.44314718
approx_kl_normal_normal = bf.expectation(
    f=lambda x: p.log_prob(x) - q.log_prob(x),
    samples=p.sample(num_draws, seed=42),
    log_prob=p.log_prob,
    use_reparametrization=(p.reparameterization_type
                           == distribution.FULLY_REPARAMETERIZED))
# ==> 0.44632751
# Relative Error: <1%

# Monte-Carlo approximation of non-reparameterized distribution, e.g., Gamma.

num_draws = int(1e5)
p = ds.Gamma(concentration=1., rate=1.)
q = ds.Gamma(concentration=2., rate=3.)
exact_kl_gamma_gamma = ds.kl_divergence(p, q)
# ==> 0.37999129
approx_kl_gamma_gamma = bf.expectation(
    f=lambda x: p.log_prob(x) - q.log_prob(x),
    samples=p.sample(num_draws, seed=42),
    log_prob=p.log_prob,
    use_reparametrization=(p.reparameterization_type
                           == distribution.FULLY_REPARAMETERIZED))
# ==> 0.37696719
# Relative Error: <1%

# For comparing the gradients, see `monte_carlo_test.py`.
```

Note: The above example is for illustration only. To compute approximate KL-divergence, the following is preferred:

```
approx_kl_p_q = bf.monte_carlo_csiszar_f_divergence(
    f=bf.kl_reverse,
    p_log_prob=q.log_prob,
    q=p,
    num_draws=num_draws)
```

Args:

- `f`: Python callable which can return `f(samples)`.
- `samples`: `Tensor` of samples used to form the Monte-Carlo approximation of `E_p[f(X)]`. A batch of samples should be indexed by `axis` dimensions.
- `log_prob`: Python callable which can return `log_prob(samples)`. Must correspond to the natural-logarithm of the pdf/pmf of each sample. Only required/used if `use_reparametrization=False`. Default value: `None`.
- `use_reparametrization`: Python `bool` indicating that the approximation should use the fact that the gradient of samples is unbiased. Whether `True` or `False`, this arg only affects the gradient of the resulting `approx_expectation`. Default value: `True`.
- `axis`: The dimensions to average. If `None`, averages all dimensions. Default value: `0` (the left-most dimension).
- `keep_dims`: If True, retains averaged dimensions using size `1`. Default value: `False`.

- `name`: A `name_scope` for operations created by this function. Default value: `None` (which implies "expectation").

Returns:

- `approx_expectation`: `Tensor` corresponding to the Monte-Carlo approximation of `E_p[f(X)]`.

Raises:

- `ValueError`: if `f` is not a Python `callable`.
- `ValueError`: if `use_reparametrization=False` and `log_prob` is not a Python `callable`.
