tf.contrib.bayesflow.csiszar_divergence.monte_carlo_csiszar_f_divergence

```
monte_carlo_csiszar_f_divergence(
    f,
    p_log_prob,
    q,
    num_draws,
    use_reparametrization=None,
    seed=None,
    name=None
)
```

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

Monte-Carlo approximation of the Csiszar f-Divergence.

A Csiszar-function is a member of,

```
F = { f:R_+ to R : f convex }.
```

The Csiszar f-Divergence for Csiszar-function f is given by:

```
D_f[p(X), q(X)] := E_{q(X)}[ f( p(X) / q(X) ) ]
                ~= m**-1 sum_j^m f( p(x_j) / q(x_j) ),
                           where x_j ~iid q(X)
```

Tricks: Reparameterization and Score-Gradient

When q 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 ~iid q(X)`.

However, if q is not reparameterized, TensorFlow's gradient will be incorrect since the chain-rule stops at samples of unreparameterized distributions. In this circumstance using the Score-Gradient trick results in an unbiased gradient, i.e.,

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

Unless `q.reparameterization_type != distribution.FULLY_REPARAMETERIZED` it is usually preferable to set `use_reparametrization = True`.

Example Application:

The Csiszar f-Divergence is a useful framework for variational inference. I.e., observe that,

```
f(p(x)) =  f( E_{q(Z | x)}[ p(x, Z) / q(Z | x) ] )
        <= E_{q(Z | x)}[ f( p(x, Z) / q(Z | x) ) ]
        := D_f[p(x, Z), q(Z | x)]
```

The inequality follows from the fact that the "perspective" of `f`, i.e., `(s, t) |-> t f(s / t))`, is convex in `(s, t)` when `s/t in domain(f)` and `t` is a real. Since the above framework includes the popular Evidence Lower BOund (ELBO) as a special case, i.e., `f(u) = -log(u)`, we call this framework "Evidence Divergence Bound Optimization" (EDBO).

Args:

- `f`: Python `callable` representing a Csiszar-function in log-space, i.e., takes `p_log_prob(q_samples) - q.log_prob(q_samples)`.
- `p_log_prob`: Python `callable` taking (a batch of) samples from `q` and returning the natural-log of the probability under distribution `p`. (In variational inference `p` is the joint distribution.)
- `q`: `tf.Distribution`-like instance; must implement: `reparameterization_type`, `sample(n, seed)`, and `log_prob(x)`. (In variational inference `q` is the approximate posterior distribution.)
- `num_draws`: Integer scalar number of draws used to approximate the f-Divergence expectation.
- `use_reparametrization`: Python `bool`. When `None` (the default), automatically set to: `q.reparameterization_type == distribution.FULLY_REPARAMETERIZED`. When `True` uses the standard Monte-Carlo average. When `False` uses the score-gradient trick. (See above for details.) When `False`, consider using `csiszar_vimco`.
- `seed`: Python `int` seed for `q.sample`.
- `name`: Python `str` name prefixed to Ops created by this function.

Returns:

- `monte_carlo_csiszar_f_divergence`: `float`-like `Tensor` Monte Carlo approximation of the Csiszar f-Divergence.

Raises:

- `ValueError`: if `q` is not a reparameterized distribution and `use_reparametrization = True`. A distribution `q` is said to be "reparameterized" when its samples are generated by transforming the samples of another distribution which does not depend on the parameterization of `q`. This property ensures the gradient (with respect to parameters) is valid.
- `TypeError`: if `p_log_prob` is not a Python `callable`.

Last updated November 2, 2017.