tf.contrib.bayesflow.hmc.leapfrog_integrator

```
leapfrog_integrator(
    step_size,
    n_steps,
    initial_position,
    initial_momentum,
    potential_and_grad,
    initial_grad,
    name=None
)
```

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

Applies `n_steps` steps of the leapfrog integrator.

This just wraps `leapfrog_step()` in a `tf.while_loop()`, reusing gradient computations where possible.

Args:

- `step_size`: Scalar step size or array of step sizes for the leapfrog integrator. Broadcasts to the shape of `initial_position`. Larger step sizes lead to faster progress, but too-large step sizes lead to larger discretization error and worse energy conservation.
- `n_steps`: Number of steps to run the leapfrog integrator.
- `initial_position`: Tensor containing the value(s) of the position variable(s) to update.
- `initial_momentum`: Tensor containing the value(s) of the momentum variable(s) to update.
- `potential_and_grad`: Python callable that takes a position tensor like `initial_position` and returns the potential energy and its gradient at that position.
- `initial_grad`: Tensor with the value of the gradient of the potential energy at `initial_position`.
- `name`: Python `str` name prefixed to Ops created by this function.

Returns:

- `updated_position`: Updated value of the position.
- `updated_momentum`: Updated value of the momentum.
- `new_potential`: Potential energy of the new position. Has shape matching `potential_and_grad(initial_position)`.
- `new_grad`: Gradient from potential_and_grad() evaluated at the new position. Has shape matching `initial_position`.

Example: Simple quadratic potential.

```
def potential_and_grad(position):
  return tf.reduce_sum(0.5 * tf.square(position)), position
position = tf.placeholder(np.float32)
momentum = tf.placeholder(np.float32)
potential, grad = potential_and_grad(position)
new_position, new_momentum, new_potential, new_grad = hmc.leapfrog_integrator(
  0.1, 3, position, momentum, potential_and_grad, grad)

sess = tf.Session()
position_val = np.random.randn(10)
momentum_val = np.random.randn(10)
potential_val, grad_val = sess.run([potential, grad],
                                   {position: position_val})
positions = np.zeros([100, 10])
for i in xrange(100):
  position_val, momentum_val, potential_val, grad_val = sess.run(
    [new_position, new_momentum, new_potential, new_grad],
    {position: position_val, momentum: momentum_val})
  positions[i] = position_val
# Should trace out sinusoidal dynamics.
plt.plot(positions[:, 0])
```


Last updated November 2, 2017.