**tfp.mcmc: Modern Markov Chain Monte Carlo Tools Built for Modern Hardware**

Junpeng Lao, Christopher Suter, Ian Langmore, Cyril Chimisov, Ashish Saxena, Pavel Sountsov, Dave Moore, Rif A. Saurous, Matthew D. Hoffman, and Joshua V. Dillon

---

**Key takeaways**

**What:** tfp.mcmc is a highly flexible and modular framework for MCMC research and Bayesian inference, focused on performance, and built on top of TensorFlow and Jax.

**How:**
- Pervasive Data Parallelism (using “batch semantics” that leverage “single instruction, multiple data” (SIMD) instruction sets (“data parallelism”)
- Requires only a simple Python callable that maps inputs -> log_prob, where inputs is a nested Python structure
- TransitionKernels and drivers that can be nested together to create new MCMC routines

**Where:** tfp.mcmc and tfp.experimental.mcmc

---

**TransitionKernel**

class TransitionKernel:

```python
@abc.abstractmethod
def one_step(self, current_state, previous_kernel_results, seed=None):
  """Takes one step of the TransitionKernel."""
  ...

def bootstrap_results(self, init_state):
  """Returns an object with the same type as returned by `one_step(...)`."""
  ...

@abc.abstractproperty
def is_calibrated(self):
  """Returns True if Markov chain converges to specified distribution."""
  ...
```

**TransitionKernels’ are composable**

```python
randomwalk_hmc = tfp.mcmc.Metropolised Hastings(
  inner_kernel=tfp.mcmc.UnCalibratedRandomWalk(  
    target_log_prob_fn=target_log_prob_fn,  
    new_state_fn=new_state_fn),
)

hmc = tfp.mcmc.Metropolised Hastings(  
  inner_kernel=tfp.mcmc.UnCalibratedHamiltonMonteCarlo(  
    target_log_prob_fn=target_log_prob_fn,  
    step_size=step_size),
)

hmc_unbounded_with_tuning = tfp.mcmc.DualAveragingStepSizeAdaptation(  
  tfp.mcmc.TransformedTransitionKernel(inner_kernel=hmc, bijector=bijector),  
  target_accept_prob=0.8, num_adaptation_steps=500)
```

---

**drivers**

def driver(kernel, initial_state):
  [] = results
  side_results = kernel.bootstrap_results(initial_state)
  for _ in range(num_samples):
    x, side_results = kernel.one_step(results[-1], side_results)
  results = [x]
  return results
  results = driver(SomeKernel(target_log_prob_callable), x0)

---

**driver examples**

def trace_fn(state, adaptive_pkr):
  """Adaptive_pkr is the previous kernel result."""
  transformed_pkr = adaptive_pkr.inner_results
  metastropolis_pkr = transformed_pkr.inner_results
  return metastropolis_pkr.is_accepted

# Draw 500 samples and trace the HW acceptance outcomes.
# samples, is_accepted = tfpmcmc.sample_chain(
#  current_state=init_state,
#  kernel=hmc_unbounded_with_tuning,
#  num_burnin_steps=500, num_results=500,
#  trace_fn=trace_fn)

cov_reducer = tfp.experimental.mcmc.CovarianceReducer()
covariance_estimate, _ = tfp.experimental.mcmc.sample_fold(
  current_state=init_state,
  kernel=hmc_unbounded_with_tuning,
  num_burnin_steps=500, num_results=500,
  trace_fn=trace_fn,
  reducers=[cov_reducer],
)

smc_result = sample_sequential_monte_carlo(
  prior_log_prob_fn,  
  likelihood_log_prob_fn,  
  current_state,  
  make_kernel_fn=make_rwmh_kernel)
```

---

**Discussion**

**Advantages and challenges of pervasive data parallelism**

Q: Why is the pervasive data parallelism advantageous? Can we just use vectorizing function like tf.vectorized_map or jax.vmap and wrap the TK into a SIMD function?

A: Pervasive data parallelism opens new opportunities to directly manipulate across "batches", even during one MCMC step. For example, we can flexibly implement population-wise MCMC methods, or coupling MCMC methods.

There are also significant challenges, for example, in the implementation of the NUTS sampler.

---

**Challenges of being modular**

Onion-like nesting TransitionKernels are powerful, but also create challenges when we try to access some properties in one of the layer of the kernel_results. We have made some progress to make this process easier with tfp.experimental.unnest

---

**Contact**

[https://www.tensorflow.org/probability/](https://www.tensorflow.org/probability/)
Reach out to us on our Google group if you have any questions: tfprobability@tensorflow.org