HackPPL: a universal probabilistic programming language

J. Ai et al.

Presented by Oliver Hope

Background

- PPLs are becoming more important
- Reduce development time for Bayesian Modelling
- PPLs trade off efficiency and expressivity
- Eg: DSLs: Stan[1], BUGS[2]; Embedded: Edward[3], Pyro[4]

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

What is HackPPL

- An extension to Hack
- A Universal Probabilistic Programming Language
- Features:
  - Modelling
  - Inference
  - Assessment
  - Mix with arbitrary Hack[5] code

Language Features: Coroutines

- Inference often uses Monte Carlo Algorithms
- Want to avoid unnecessary re-execution for selectively exploring sub-computations.
- “Models are implemented as coroutines that are reified as multi-shot continuations in inference code”
- fundamental characteristics:
  1. Values persist between calls
  2. Execution continues where left off are returning from suspension
- Uses state machines, CPS and Trampolining

Language Features: Coroutines

Listing 4. Example coroutine function myCoroutine()

```
class MyClass {
  ?MultishotContinuation<string> $suspended_coroutine;
  public coroutine function myCoroutine(): string {
    print "Started the coroutine";
    $resumed_value = suspend suspendMultiple(
      coroutine ($my_suspended_coroutine) ==> {
        $this->suspended_coroutine = $my_suspended_coroutine;
    });
    print "Continued the coroutine";
    return $resumed_value;
  }
}
```

Listing 5. Coroutine invocation with multiple resumptions

```
class CoroutineCallback implements Continuation<string> {
  public function resume(string $coroutine_return): void {
    print $coroutine_return;
  }
}
StartCoroutine::start(
  coroutine () ==> suspend $my_class->myCoroutine(),
  new CoroutineCallback());
// Later on...
$my_class->suspended_coroutine->resumeAsync("First");
$my_class->suspended_coroutine->resumeAsync("Second");
```

Language Features: Data models

- Continuous Values:
  - Tensors for distributions, samples, and observations
  - Imported to Hack from PyTorch[6]
  - Natively support reverse-mode automatic differentiation
- Discrete Values:
  - Introduce DTensor
  - Can convert to one-hot encoding
  - When used, we run simulations for all values

Listing 7. DTensor construction with a vocabulary mapping. The one-hot-encoded tensor will be of the form [[1,0,0], [0,0,1], [0,1,0]].

```
$labels = vec[1, 3, 2];
$vocab = vec['a', 'c', 'b'];
$dtensor = new DTensor($labels, $vocab);
$one_hot_tensor = $dtensor->toOneHotEncodedTensor();
```

Language Features: Distributions

- Many built in
- Must implement:
  - `sample(n)`: retrieve n i.i.d samples from distribution
  - `score(x)`: compute the log probability at x
- Allow for batch sampling and scoring too.

Inference Engine

- Completely separate to modelling (for flexibility)
- Aim: “Obtain a posterior estimate for model parameters”
- Takes a trace-based approach
- `PPLInfer` class:
  - Centralised way to specify configuration
  - Centralised way to construct pipelines
- Built-ins such as Hamiltonian Monte Carlo

Listing 9. Custom inference pipeline. This returns the expected value for a particular random variable.

```
$results = PPLInfer::hmc($model)
  ->map($samples ==> $samples['w'])->reduce(($d, $val) ==> {
    $weight = $d['weight'] + 1.;
    $mean = ($d['mean'] * $d['weight'] + $val) / $weight;

    return dict['mean' => $mean, 'weight' => $weight];
  })->run($iterations);
```

Inference Engine

- Auto-tunes hyperparameters using No-U-Turn[7] (for HMC)
- Supports automatic marginalisation[1] for discrete parameter sampling
- This requires multi-shot coroutines
- Can resume inference from history
- Supports Black Box Variational Inference[8] (a form of scalable inference)

Listing 10. Finite mixture model in HackPPL

```
$mu = sample(new Normal($mu_p, $sigma_p), 'mu');
$c = sample(new Categorical($p), 'c');
foreach ($data as $i => $y) { // Observe on data bound to y$i
  sample(new Normal($mu->getTensorAt($c[$i]), $sigma), "y$i");
}
```

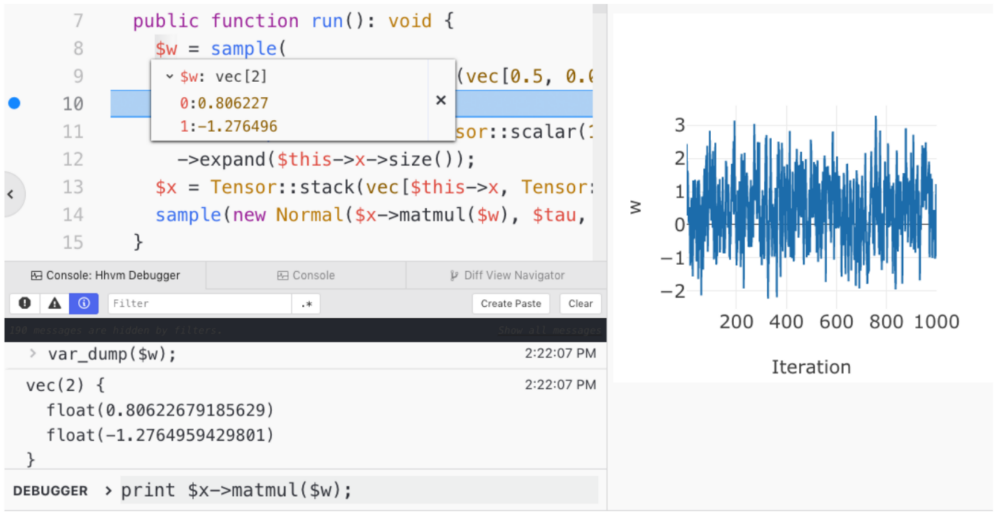
$$P(y \mid p, \mu, \sigma) = \sum_{c=1}^{C} p_c \mathsf{Normal}(y \mid \mu_c, \sigma)$$

Assessment

- Simple to obtain the posterior predictive distribution. (effectively simulation mode)

$$P(y_{new} \mid y) = \int P(y_{new} \mid \theta) P(\theta \mid y) dy$$

- There is a playground built into Nuclide IDE



- A realtime visualisation library (Viz)
- A model criticism library for posterior predictive checks[9]

Criticisms

- No comparison to existing PPLs
- No evaluation of performance
- No evaluation of UX
- Many statements lack justification
- Code is incomplete for brevity — this is not stated though.
- Nuclide (and in fact HackPPL) is not available outside Facebook.

Questions?

References I

[1] B. Carpenter, A. Gelman, M. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell, "Stan: A probabilistic programming language," Journal of Statistical Software, Articles, vol. 76, no. 1, pp. 1–32, 2017. [Online]. Available: https://www.jstatsoft.org/v076/i01

[2] W. R. Gilks, A. Thomas, and D. J. Spiegelhalter, "A language and program for complex bayesian modelling," 1994.

[3] D. Tran, A. Kucukelbir, A. B. Dieng, M. R. Rudolph, D. Liang, and D. M. Blei, "Edward: A library for probabilistic modeling, inference, and criticism," ArXiv, vol. abs/1610.09787, 2016.

[4] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. A. Szerlip, P. Horsfall, and N. D. Goodman, "Pyro: Deep universal probabilistic programming," J. Mach. Learn. Res., vol. 20, pp. 28:1–28:6, 2018.

[5] Hack · programming productivity without breaking things. [Online]. Available: https://hacklang.org/

[6] Pytorch. [Online]. Available: https://pytorch.org/

References II

[7] M. D. Hoffman and A. Gelman, "The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo," J. Mach. Learn. Res., vol. 15, pp. 1593–1623, 2011.

[8] R. Ranganath, S. Gerrish, and D. M. Blei, "Black box variational inference," in AISTATS, 2013.

[9] A. Gelman, X.-L. Meng, and H. S. Stern, "Posterior predictive assessment of model fitness via realized discrepancies," 1996.