

Machine Learning for Gravitational Waves: jax

Thibaud Wouters



December 15, 2023



**Utrecht
University**

Table of Contents

- ① Introduction
- ② Some jax take-aways
- ③ Conclusion
- ④ Appendix: GW parameter estimation with jax

Table of Contents

① Introduction

② Some jax take-aways

③ Conclusion

④ Appendix: GW parameter estimation with jax

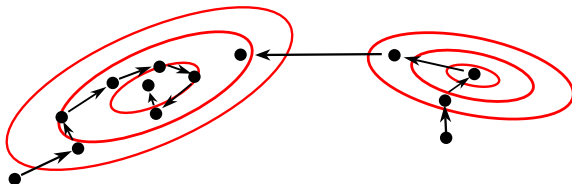
Parameter estimation

- Parameter estimation (PE): get **posterior** of EM/GW parameters θ

$$p(\theta|d) = \frac{p(d|\theta)p(\theta)}{p(d)}$$

- Sampling via Markov Chain Monte Carlo (MCMC) [1]

Goal: Improve algorithms with a jax implementation



Why jax?

What are the benefits of jax for MCMC?

- ① Automatic differentiation (AD)
- ② Just-in-time (JIT) compilation
- ③ GPU acceleration
- ④ Parallelization
- ⑤ Shown to speed up PE for GWs [2], cosmology [3],...



What is my experience with jax?

- 1 TaylorF2, NRTidalv2 GW in ripple [4]
- 2 Worked with flowMC [5, 6], a gradient-based, normalizing flow-enhanced MCMC sampler
- 3 Worked on jim [2], a PE pipeline for GWs (see appendix of this presentation for more information)
- 4 Implemented kilonova surrogate models in flax [7] for NMMA [8]

What are some take-aways for getting started with jax?

Table of Contents

① Introduction

② Some jax take-aways

③ Conclusion

④ Appendix: GW parameter estimation with jax

#1 – Don't be scared, just try it out!

- jax is easy to get started with: `numpy` → `jax.numpy` works well!
- What about large projects?
 - Make sure to read the “sharp bits”
 - Check out other repositories for inspiration ([list of repos](#))
 - Read/watch some tutorials, such as [this one](#)
 - Large project? *Start from scratch!*

Small [demo](#) about first caveats

#2 – Functions must be pure: conditionals

Example: lalsuite vs jax for NRTidalv2

```
else if ( 0. <= lambda2bar && lambda2bar < 1. ) {
    /* Extension of the fit in the range lambda2bar=[0,1.] so that
    the BH limit is enforced, lambda2bar->0 gives quadparam->1. and
    the junction with the universal relation is smooth, of class C2 */
    return 1. + lambda2bar*(0.42768866723244 + lambda2bar*(-0.324336526985068 + lambda2bar*0.1107439432180572));
}
else {
    lnx = log( lambda2bar );
}
REAL8 lny = XLALSimUniversalRelation( lnx, coeffs );
return exp(lny);
```

lalsuite

```
def get_quadparam_octparam(lambda_: float) -> tuple[float, float]:
    # Check if lambda is low or not, and choose right subroutine
    is_low_lambda = lambda_ < 1
    return jax.lax.cond(is_low_lambda, _get_quadparam_octparam_low, _get_quadparam_octparam_high, lambda_)

def _get_quadparam_octparam_low(lambda_: float) -> tuple[float, float]:
    You, now * Uncommitted changes
    # Coefficients of universal relation
    oct_coeffs = [0.003131, 2.071, -0.7152, 0.2458, -0.03309]

    # Extension of the fit in the range lambda2 = [0,1.] so that the BH limit is enforced, lambda2bar->0 gives quadpa
    quadparam = 1. + lambda_ * (0.42768866723244 + lambda_ * (-0.324336526985068 + lambda_ * 0.1107439432180572))
    log_quadparam = jnp.log(quadparam)

    # Compute octparam:
    log_octparam = universal_relation(oct_coeffs, log_quadparam)
    octparam = jnp.exp(log_octparam)

    return quadparam, octparam

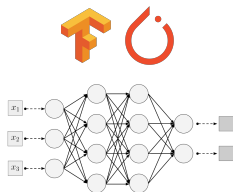
def _get_quadparam_octparam_high(lambda_: float) -> tuple[float, float]:
```

jax
(code)

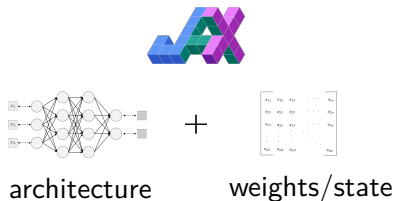
#3 Neural networks

Neural networks in flax:

- Most similar to PyTorch, more work than tensorflow
- flax has a different “mindset”
- Check out my very basic [flax code](#)



`nn.train()`



`state=apply_gradients(grads)`

Table of Contents

① Introduction

② Some jax take-aways

③ Conclusion

④ Appendix: GW parameter estimation with jax

Conclusion

- jax is a great tool to speed up code
- jax is easy, but watch out for the “sharp bits”
- flax implements neural networks with jax
- Interested? Let's connect and learn together!

Thanks for listening! Questions?

Table of Contents

① Introduction

② Some jax take-aways

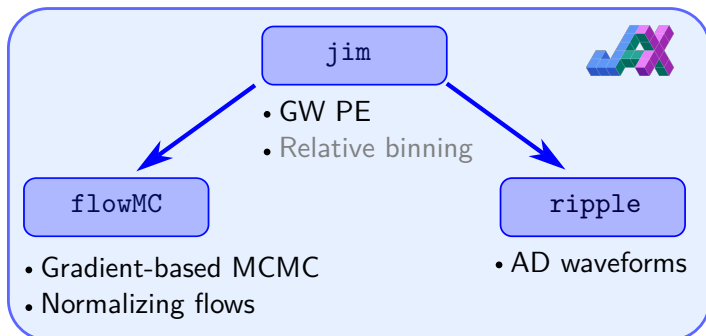
③ Conclusion

④ Appendix: GW parameter estimation with jax

Overview

We extend `jim` [2], based on `jax` [9], with building blocks:

- 1 Normalizing flow-enhanced, gradient-based MCMC (`flowMC` [5, 6])
- 2 Automatically-differentiable (AD) GW (`ripple` [4])
- 3 Relative binning likelihood [10]

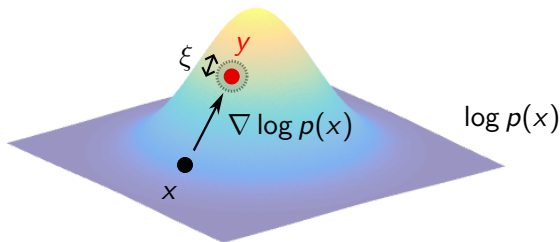


① **Local sampling:** MALA (Metropolis-adjusted Langevin algorithm)

- Proposal y : Langevin diffusion

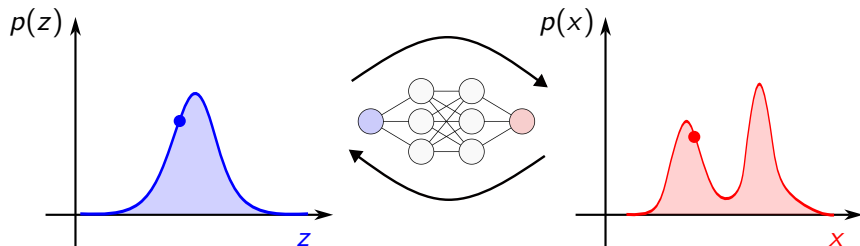
$$y = x + \frac{\epsilon^2}{2} \nabla \log p(x) + \epsilon \xi$$

- Metropolis-Hastings acceptance step
- Motivates the need for automatic differentiation



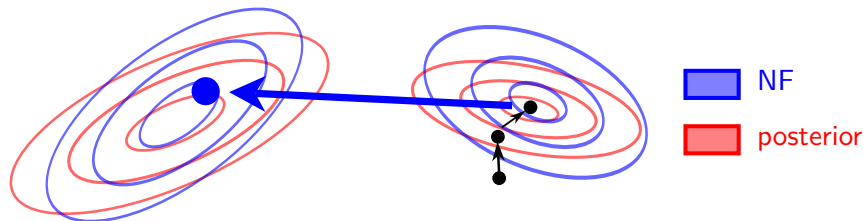
Normalizing flows (NF):

- **Latent space**: easy to sample (e.g. Gaussian)
- **Data space**: distribution learned from samples
- Enable approximate sampling from complicated distributions



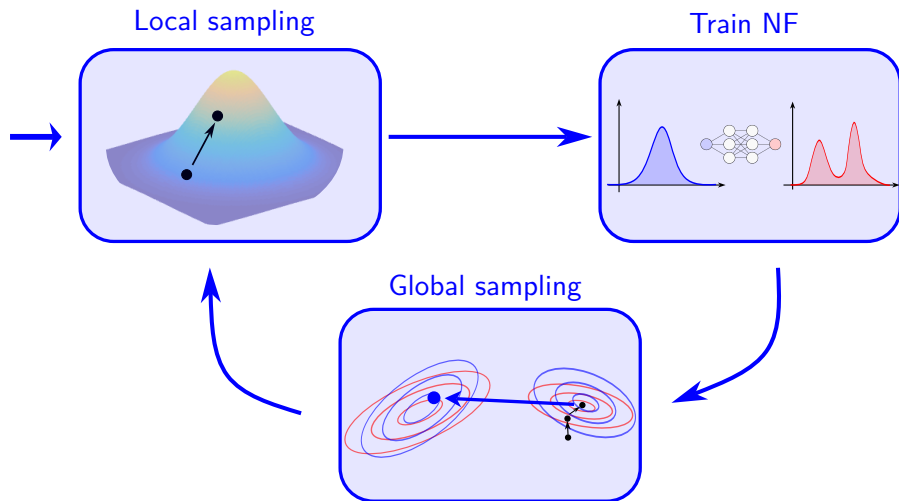
② Global sampling

- Global proposal by sampling from NF
- Metropolis-Hastings acceptance step



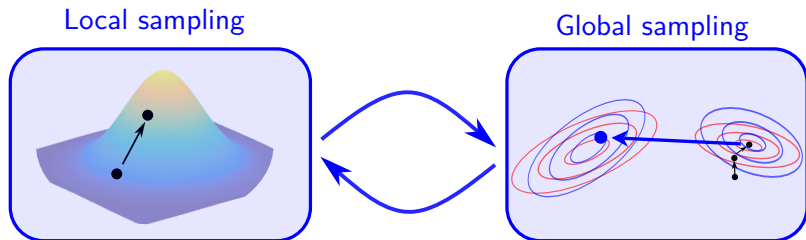
flowMC – complete algorithm

Training loop & Production loop



flowMC – complete algorithm

Training loop & **Production** loop



Results

- TaylorF2 in ripple
- IMRPhenomD_NRTidalv2 in ripple (ongoing)
- Reproduced PE for GW170817 & GW190425 with TaylorF2
- ~ 30 mins training, ~ 1 min sampling

