ABC-NN: Approximate Bayesian Computation with Neural Networks to learn likelihood functions for efficient parameter estimation

Alexander Fengler (alexander_fengler@brown.edu)
Department of Cognitive Linguistic and Psychological Sciences, Brown University, 190 Thayer Street
Providence, RI 02912 United States

Michael J. Frank (Michael_Frank@brown.edu)
Department of Cognitive Linguistic and Psychological Sciences, Brown University, 190 Thayer Street
Providence, RI 02912 United States

Abstract

In cognitive neuroscience, computational modeling offers a principled interpretation of the functional demands of cognitive systems. Bayesian parameter estimation provides information about the full posterior distribution over likely parameters. Importantly, the set of models with known likelihoods is dramatically smaller than the set of plausible generative models. Approximate Bayesian Computation (ABC) methods facilitate sampling from posterior parameters for models specified only up to a data generating process, overcoming this limitation to afford bayesian estimation of complex stochastic models (Wood, 2010; Beaumont, 2010; Akeret, Refregier, Amara, Seehars, & Hasner, 2015; Turner & P., 2014). Relying on model simulations to generate synthetic likelihoods, these methods however come with substantial computational cost at inference where simulations are typically conducted at each step in a MCMC algorithm. We propose a method that learns an approximate likelihood over the parameter space of interest, using multilayered perceptrons (MLPs). This incurs a single upfront cost, but the resulting network comprises a usable likelihood function that can be freely used in standard inference algorithms. We test this approach in the context of drift diffusion models, a class of cognitive process models commonly used in the cognitive sciences to jointly account for choice and reaction time data in a variety of experimental settings (Ratcliff, Smith, Brown, & McKoon, 2016).

Keywords: ABC, Neural Networks, DDM

General Idea

The main shortcoming of current approximate bayesian inference (ABC) approaches to parameter estimation is the computational burden incurred by performing model simulations inside of MCMC algorithms. These burdens are especially salient in scenarios in which analytically intractable models are used and where one wishes to conduct model comparison over various hierarchical model extensions (illustrated in *Figure 1*). Simulations need to be run for every combination of parameters searched, for each step in each chain, and for every model variant (e.g., in a cognitive task with multiple conditions, researchers often test models in which one or any number of parameters may or may not vary per condition). An application of ABC methods in these scenarios may limit thorough exploration of alternative variants simply for pragmatic computational reasons. Our goal here is to make this process more computationally efficient and to permit model estimation for otherwise intractable models.

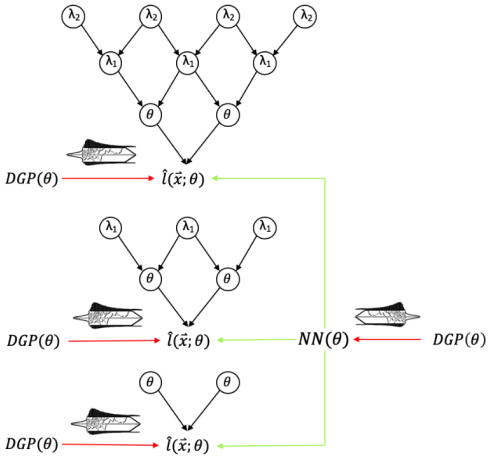

Figure 1: Purchase of ABC-NN for Hierarchical Model Extensions

The basic idea of our approach is to separate the model simulation / likelihood estimation stage from the inference stage. More specifically, we attempt to separate the MCMC-algorithms, employed locally by researchers at the inference stage, from the generation of approximate likelihoods, which can be computed up-front for a given model class over a range of parameters and stored as a function, which can then be called directly or inside software packages (e.g., HDDM: hierarchical bayesian estimation of the drift diffusion model in Python (Wiecki, Sofer, & Frank, 2013)). While a likelihood function is analytically available for many popular models (e.g., the DDM), even slight change to these models may not have analytical forms (e.g., a DDM with a collapsing bound). We propose here a machine learning approach, using model simulations across parameters to *learn the likelihood function upfront* with the help of a multi-layered-perceptron (MLP, also simply referred to as neural network in the following). Once such an approximate likelihood function is provided, simulat-

ing the $\mathcal{DGP}$ (*Data Generating Process*), is then not necessary at the stage of inference, during which MCMC-algorithms need only to evaluate the function, reducing the costs of bayesian inference back to what is typical for analytic likelihood functions. *Figure 2* illustrates the idea in terms of the change in the corresponding MCMC algorithm for inference.

MCMC Algorithms:

initialization; (θ_1, λ_1) , $n = 2$

while $n \leq N$ **do**
  propose $[\theta^*, \lambda^*]$;
  $\hat{l}_{KDE}(x;\theta) \leftarrow \mathcal{DGP}(\theta)$ [simulations]
  $\hat{p}(\theta^*|x) \propto \prod_{i=1}^{K} \hat{l}_{KDE}(x_i;\theta^*)p(\theta^*, \lambda^*)$
  if *accept*$((\theta^*, \lambda^*))$ **then**
    $(\theta_n, \lambda_n) \leftarrow (\theta^*, \lambda^*)$;
  else
    $(\theta_n, \lambda_n) \leftarrow (\theta_{n-1}, \lambda_{n-1})$;
  end
end

while $n \leq N$ **do**
  propose $[\theta^*, \lambda^*]$;
  $\hat{p}(\theta^*|x) \propto \prod_{i=1}^{K} \hat{l}_{NN}(x_i;\theta^*)p(\theta^*, \lambda^*)$
  if *accept*$((\theta^*, \lambda^*))$ **then**
    $(\theta_n, \lambda_n) \leftarrow (\theta^*, \lambda^*)$;
  else
    $(\theta_n, \lambda_n) \leftarrow (\theta_{n-1}, \lambda_{n-1})$;
  end
end

Figure 2: *Left* is the standard ABC approach where the DGP sits inside the markov chain. *Right* the MCMC algorithm where the likelihood is provided by a neural network $\hat{l}_{NN}$

Using this approach we can learn the likelihood manifold for a given model into a neural network first, and then provide researchers with the learned network (set of weights, requiring only matrix algebra) to be used for inference. We hope to exploit model simulations to directly inform an interpolation algorithm that can leverage regularities in the global structure of the likelihood manifold and then apply them during inference without need for further simulation.

Tested Model

As an intial test we focus on the viability of our approach to the DDM, a popular model of decision making in which reaction times and decisions are described as the crossing of one of two boundaries of a single diffusion process, and for which the analytical likelihood is available (and hence can be used as a benchmark) but non trivial. *Figure 3* provides a an illustration of the DDM model.

For a starting point w, drift μ, and (symmetric) functional shape of the boundaries $g(t)$, we have two *defective* probability distributions f_+ and f_-, which jointly provide the first passage distribution of (upper and lower) boundary crossings (Ratcliff, 1978) of some associated stochastic process $X(t;w,\mu)$. For a given set of parameters (w,μ), the time integral of the sum of the two *defective* probability distributions $f = f_- + f_+$ is a valid probability density. For a given data point, where the choice $c \in \{-,+\}$, and reaction time $t \in (0,+\infty)$, we can describe it's likelihood given parameters of the model as, $f_c(t;w,\mu)$. As described above, our problem is that f_+ and f_- are given in analytic form only for a small class of models (e.g., for the DDM in which the drift rate and

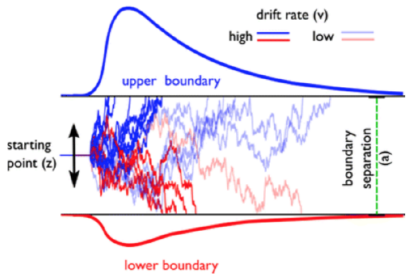

Figure 3: Drift Diffusion Model (DDM)

decision boundary are fixed across time), even though there are many situations in which these assumptions are untenable, motivating ABC approaches in the first place. To get an approximation for the likelihood of (c,t), given (w,μ) and $g(t)$, Turner et. al. (2014), suggest the following approach. We simulate the process to get an artificial data set (for example using *Euler Maruyama*), $D = \{(c_1,t_1),...,(c_n,t_n)\}$, which we split according to whether $c = +$, or $c = -$, and re-index, to create two separate data-sets $D_+ = \{(+,t_{11}),...,(+,t_{1n_1})\}$, $D_- = \{(-,t_{21},...,(t_{2n_2})\}$. Computing the proportion of upper and lower boundary crossings respectively as, $p_+ = \frac{|D_+|}{|D_-|+|D_+|}$ and $p_- = \frac{|D_-|}{|D_-|+|D_+|}$, and choosing some kernel density estimator, (KDE) we end up with approximations to f_+ and f_- as,

$$\hat{f}_+(t;w,\mu) = p_+ * KDE(t;D_+)$$
$$\hat{f}_-(t;w,\mu) = p_- * KDE(t;D_-)$$

which, can then be used inside any MCMC algorithm.

The approach proposed here is to use $\hat{f}_+$ and $\hat{f}_-$, so computed, not inside the Markov Chain, but instead to form a training set to serve the training of Neural Network, which learns an approximate likelihood function $\hat{l}_{NN}(t,c;w,\mu)$ for the relevant parameter space a priori.

Preliminary Results

DDM: Training on known likelihood

We attempt to fit the first passage distribution of a standard DDM (constant boundaries at level 0 and a). The functional form for the probability density of lower barrier crossings at time t (similar for upper barrier crossings) given parameters (μ,a,w) is (Feller, 1968),

$$f(t|\mu,a,w) = \frac{\pi}{a^2}\exp\left(-\mu a w - \frac{\mu^2 t}{2}\right)\sum_{k=1}^{\infty} k \exp\left(-\frac{k^2\pi^2 t}{2a^2}\right) sin(k\pi w)$$

The parameter space under consideration was restricted to, $(\mu,w,a) \in [-2.5,2.5] \times [0.15,0.85] \times [0.5,3]$. A data

set of $54.000.000$ examples was generated on which the NN's were trained. A training data point has the form $\{(\mu_i, w_i, a_i, c_i, rt_i), (\ell_i)\}$ where $(\mu_i, w_i, a_i, c_i, rt_i)$ is the set of *features or inputs* and ℓ_i is the corresponding *label or output*. The training examples were generated from the following mixture distribution. First, a set of parameters (μ_i, a_i, w_i) was sampled uniformly from the parameter space of interest. Then (c_i, rt_i) were sampled from a mixture distribution of three components.

The first component, with $p_3 = 0.9$, produced joint choice and reaction time samples directly from the drift diffusion process given the previously sampled parameters (μ_i, a_i, w_i). The second component, with $p_1 = 0.05$, samples a choice $c_i \sim \mathcal{B}(0.5)$ and a reaction time $rt_i \sim \mathcal{N}(0, 1)$. The purpose of this component is to ensure that the network encounters training examples close to and below 0 on the reaction time axis, to ensure a collapse towards 0 of the learned likelihood close to 0. The third component, with $p_2 = 0.05$ samples choice $c_i \sim \mathcal{B}(0.5)$ and reaction time $rt_i \sim \mathcal{U}([5, 20])$ to ensure that the training set contains *reaction-time-outliers* for a any given parameter set. While these proportions were not chosen based on some strict theoretical justification (more principled approaches are in progress, e.g. active sampling of parameters for more uncertain regions), empirically they were enough to constrain behavior of the learned manifolds. Finally, we split the resulting data-set into a training and test set using $p_{train} = 0.8$.

Next we performed a *hyperparameter search* over NN architectures. A random search over architectures was performed, ranging from 3-5 *hidden-layers* and between 20-60 *nodes* by layer. The best performing model was close to ceiling on both hyper-parameters, with 5 *hidden layers* and $(50, 60, 50, 50, 60)$ *nodes* by layer. From here we increased the architecture size to 6 hidden-layers, and tested a setup with 80 *nodes* and 100 *nodes* each and sigmoid-activation functions throughout, yielding the currently best performing network with $(100, 100, 100, 100, 100)$ nodes and *test MSE* of 0.0009. Increasing the network size to 120 *nodes* each did not further improve performance. We plan to test deeper networks going forward.

The following approach was used for a *parameter recovery study* to test the performance of our network for inference. We sample 100 parameter sets (μ_i, a_i, w_i) uniformly from $[-2.5, 2.5] \times [0.15, 0.85] \times [0.5, 3]$. For each parameter set, we simulate the corresponding data generating process, 5000 (2500, 500, 100) times and record the respective choice outcomes. For each set of parameters, we attempt parameter recovery via maximum likelihood methods. For this purpose, a simple genetic algorithm was used (population size: 40, steps: 25, mutation probability: 0.1), but other approaches such as simplex or MCMC for full bayesian inference will work similarly. *Figures 4 and 5* illustrate our preliminary results.

DDM: Training on empirical likelihood

While the previous section demonstrated the viability of the approach when a known analytic likelihood is used to train the network, here we assess the potential when this likelihood is not known, but instead provided via empirical likelihoods using KDE. Specifically, rather than providing as $f(t|\mu, a, w)$ in training the actual likelihood function,

$$f(t|\mu, a, w) = \frac{\pi}{a^2} \exp\left(-\mu a w - \frac{\mu^2 t}{2}\right) \sum_{k=1}^{\infty} k \exp\left(-\frac{k^2 \pi^2 t}{2a^2}\right) sin(k\pi w)$$

we instead provide at each training step the approximate (empirical) likelihood (generated using Gaussian Kernel Density Estimates on top of realizations of model simulations). While this procedure is more computationally intensive (more samples needed for each training step), it is again a one-off cost. Notably, we were able to perform MLE-based model recovery with similar precision, thus serving as validation of our general method, and a stepping stone towards application to models where we have no access to the ground truth (likelihood function) to begin with. The results are shown in *Figure 6*.

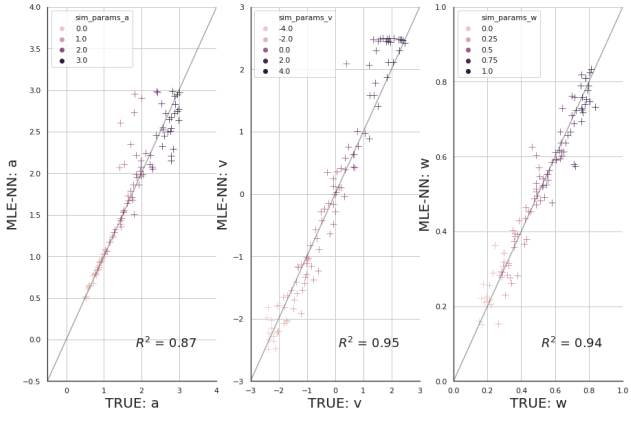

Figure 4: DDM: Parameter Recovery

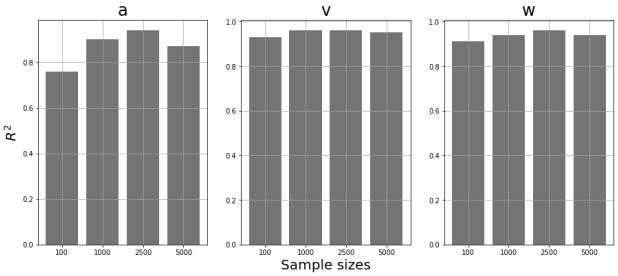

Figure 5: DDM: Parameter Recovery across data set sizes

Current Development / Future Directions

The current results provide only the first steps necessary for scaling up the approach for use more widely. First, there is still room for *improvement of training procedures* of the neural networks. Access to the data generating process, predestines

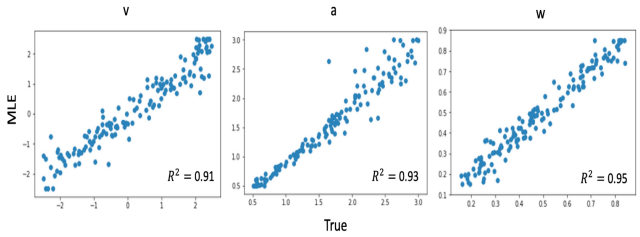

Figure 6: DDM: Parameter Recovery using empirical likelihoods in training

our method for implementation of online learning. Moreover, online learning in turn allows the opportunity to apply ideas from active learning (Bachman, Sordoni, & Trischler, 2017), essentially an interaction between the state of learning of the model and and the specifics of the training data generation process. Second, we are currently validating *more complicated models*, such as DDM with dynamically varying parameterized boundaries (e.g., *collapsing bounds*) and other models such as the leaky competing accumulator (LCA). Third, we aim to provide researchers with *two interfaces* to the method proposed in this report. An interface to perform Bayesian inference with pre-learned approximate likelihoods of standard models in the literature, and an interface that provides researchers with a training pipeline for user supplied process models.

Acknowledgments

References

Akeret, J., Refregier, A., Amara, A., Seehars, S., & Hasner, C. (2015). Approximate bayesian computation for forward modeling in cosmology. *Journal of Cosmology and Astroparticle Physics*.

Bachman, P., Sordoni, A., & Trischler, A. (2017, August). Learning Algorithms for Active Learning. *Proceedings of the th International Conference on Machine Learning*, 1–10.

Beaumont, M. A. (2010, December). Approximate Bayesian Computation in Evolution and Ecology. *Annual Review of Ecology, Evolution, and Systematics*, *41*(1), 379–406.

Feller, W. (1968). *An introduction to probability theory and its applications vol 1*. Wiley.

Ratcliff, R. (1978). A theory of memory retrieval. *Psycholoical Review*, *85*(2), 59–108.

Ratcliff, R., Smith, P., Brown, S., & McKoon, G. (2016). Diffusion decision model: Current issue and history. *Trends in Cognitive Sciences*, *20*(4), 260–281.

Turner, B., & P., S. (2014). A generalized, likelihood-free method for posterior estimation. *Psychological Bulletin*, *21*(2), 227–250.

Wiecki, T. V., Sofer, & Frank, M. (2013, jul). Hddm: Hierarchical bayesian estimation of the drift-diffusion model in python. *Frontiers in neuroinformatics*, *7*, 1–10.

Wood, S. (2010). Statistical inference for noisy nonlinear ecological dynamic systems. *Nature*, *466*(26).