Estimation of agent-based models using Bayesian deep learning approach of BayesFlow

塩野剛志 [1]

Takashi SHIONO[1]

[1] クレディ・スイス経済調査部
[1]Credit Suisse Economic Research

Abstract: This paper examines the possibility of applying the novel likelihood-free Bayesian inference called BayesFlow proposed by Radev et al. (2020) to the estimation of agent-based models (ABMs). The BayesFlow is a fully likelihood-free approach, which directly approximates a posterior rather than a likelihood function, by learning an invertible probabilistic mapping that implements a Normalizing Flow between parameters and a standard Gaussian variables conditioned by data from simulations. This deep neural network-based method can mitigate the trilemma in the existing methods that all of the following three –higher flexibility, lower computational cost, and smaller arbitrariness cannot be achieved at the same time. As a result of the experiments, BayesFlow certainly achieved the superior accuracies in the validation task of recovering the ground-truth values of parameters from the simulated datasets, in case of a minimal stock market ABM. The method did not involve any extensive search of the hyperparameters or hand-crafted pre-selections of summary statistics, and took a significantly shorter computational time than an existing non-parametric MCMC approach.

1. Introduction

Agent-based models (ABM) have widely been used in the field of artificial markets and related-studies to investigate the microstructure of financial markets. In recent years, they also have increasingly been adopted in macroeconomic analysis, as an alternative to DSGE models. According to Grazzini et al. (2017), agent-based models are characterized by the following three features: (1) there are a multitude of agents that interact with each other and environment, (2) these agents are autonomous, in the sense that there is no central coordinator such as a Walrasian auctioneer nor the concealed time-axis in which the central coordinator works, and (3) aggregation is performed numerically.

Thus, an ABM allows for extreme flexibility in setting the behavioral patterns of each agent in heterogeneous manner. There is no prerequisite for rational expectation nor (ex-ante) market equilibrium, as the adaptive processes of learning and selection by agents are explicitly modeled. This assumption in agent's behavior is a fundamental difference between DSGE and ABM. Recent studies of agent-based macroeconomic modeling like Assenza et al. (2015) and Caiani et al. (2016) succeed to reproduce the stylized macroeconomic phenomena such as the Phillips curve and the Oaken law from simulations of the ABM with adaptive agents. In other words, a rational agent was not a necessary condition for these observed phenomena of macroeconomy. Furthermore, their models precisely described the relationship between the financial sector and the real economy by explicitly modeling the balance sheet for each economic agent, which are therefore useful in discussing the impact on real economy from prudent policies on financial institutions. Because of its flexibility, ABMs are increasingly being understood for their usefulness as a complement to DSGE and will continue to be applied to various economic phenomena.

However, it has often been pointed out that an ABM has its weakness in the lack of empirical validation (Gallegati & Richiardi, 2009). The parameters in an ABM are usually calibrated manually to make the model's simulation outputs reproduce some particular characteristics of economic observables (e.g. fat-tail distribution or volatility clustering in stock return). In response to this challenge, studies to statistically estimate the parameters of ABM have recently begun.

Against these backgrounds, this paper adopts a novel likelihood-free Bayesian inference called BayesFlow proposed by Radev et al. (2020) for the statistical

estimation of ABMs. This method can mitigate the trilemma in the existing methods that all of the following three –higher flexibility, lower computational cost, and smaller arbitrariness cannot be achieved at the same time.

The BayesFlow is a fully likelihood-free approach, which directly approximates a Bayesian posterior rather than a likelihood function. This method is highly flexible, only requiring the ability to output simulation data from a mathematical forward model, and has an asymptotic theoretical guarantee for sampling from the true posterior without any specific assumption on the shape of the target posterior or prior. Hence, it does not need to presume ergodicity or stationarity of simulated time series of a model. Indeed, Radev et al. (2020) showed high accuracies of the BayesFlow even in the cases of potentially chaotic (the Ricker population model) and non-ergodic (the Levy-Flight model) mathematical models.

Secondly, in contrast to the typical likelihood-free methods such as Approximate Bayesian Computation, BayesFlow does not involve discretionary pre-selections or hand-crafted design in the critical parts of inference. It accompanies the learnable summary network which can compress variable-length potentially large dimensional inputs into fixed-length small dimensional summary statistics. Namely, a researcher does not need to pre-select specific moments of specific observables as a hand-crafted summary statistics.

Finally, it is computationally efficient, particularly in the case that repeated inferences with different observation datasets are needed. BayesFlow realizes amortized inference, where estimation is split into a computationally intensive training phase, and a much cheaper inference phase. In the training phase, BayesFlow tries to learn a model to output an approximate posterior that works properly for any possible observation sequence. Then, evaluating the trained model over a specific observation dataset is computationally very cheap, so that the upfront training efforts amortizes over multiple inferences.

In the next section, I explain a basic structure of the BayesFlow comparing with other related methodologies. Then, the procedure and results of validation are presented in Section 3, before discussing the advantages and limitations with conclusion.

2. Methods

2.1.Notation

In the following, I denote the data simulated from agent-based model as $X_{1:T} \equiv (X_1, \dots, X_T) \in \mathbb{R}^K \times, \dots, \times \mathbb{R}^K$, where individual X_t is a vector of observable variables in a model with its dimension being denoted as K. The number of observation points in a dataset is denoted as T to make it clear that simulation outputs from an ABM are usually multivariate time-series. Simulated data is also expressed as $X_{1:T}(\theta)$ when it needs to be emphasized that the data is generated from the ABM with parameters θ. Actual observed data or test data will be expressed with a superscript as $X^o_{1:T}$. Parameters of a forward model (i.e. ABM) are represented as a vector $\theta \in \mathbb{R}^d$.

2.2.Agent-Based Model

The state of the whole system of an agent-based model at time t is described by the collection of all micro-states of individual agent i in time period t as $S_t \equiv \{s_{i,t}\}_{i=1}^N$. The evolution or the law of transition of each agent's state is expressed as:

$$s_{i,t} = \mathrm{f}_i(S_{t-1}, \xi_t, \theta), \tag{1}$$

where $\mathrm{f_i}$ is an agent-wise state transition function, taking values in $\mathbb{R}^L$. $\xi_t \equiv \{\zeta_{i,t}\}_{i=1}^N$ is a vector to bundle all stochastic elements. The functions f reflect the detailed modelling of agent's learning, selection and interaction behavior in an ABM, which are typically heterogeneous and accompanying discontinuities such as *if-else* statements.

Aggregate observable variables X_t are then be defined over S_t:

$$X_t = \mathrm{m}(S_t), \tag{2}$$

where a function m aggregates and transforms the collection of micro-states into observable variables X_t.

Combining the state transition $\mathrm{f_i}$ in Eq.(1) and observation m in Eq.(2), a simulation data generation function G of a ABM can be defined as follows:

$$X_{1:T} = G(\theta, \xi_{1:T}) \text{ with } \xi_{1:T} \sim p(\xi). \tag{3}$$

This function G effectively corresponds to one shot of ABM simulation run with the parameters θ and the stochastic elements $\xi_{1:T}$ sampled from the known distribution[1].

2.3.Bayesian Inference on Agent-Based Model

In Bayesian inference, the posterior defined below contains all information about θ extractable from a series of observations $X_{1:T}$:

$$p(\theta|X_{1:T}) = \frac{p(X_{1:T}|\theta)p(\theta)}{\int p(X_{1:T}|\theta)p(\theta)\,d\theta}.$$

Even when a closed-form expression of the posterior is unobtainable, various sampling schemes from the posterior such as MCMC or Sequential Monte Carlo (SMC) are applicable, as long as the likelihood of a forward model $p(X_{1:T}|\theta)$ can easily be evaluated by the actual observations $X^o_{1:T}$ together with any prior $p(\theta)$, such that:

$$p(\theta|X^o_{1:T}) \propto \mathcal{L}(\theta; X^o_{1:T})\, p(\theta).$$

In case of most practical-scale ABMs, however, the likelihood function $\mathcal{L}(\theta; X^o_{1:T}) \equiv p(X^o_{1:T}|\theta)$ is generally intractable, in other words, not available in closed-form. This is the central challenge in Bayesian inference on an ABM. The existing study by Grazzini et al. (2017) tried to approximate the likelihood function, and apply standard posterior sampling schemes such as MCMC with the approximated likelihood. They limited their scope to ergodic ABMs, to ensure simulation time series generated from a model remain stationary around the deterministic steady state level $g^*(\theta)$ as:

$$X_{1:T} = \{g^*(\theta) + \epsilon_t\}_{t=1}^T.$$

In that case, a likelihood of the observation series $X_{1:T}$ is just products of a time-invariant density function: $p(X_{1:T}|\theta) = \prod_t f(X_t|\theta)$.

One approach of them is to approximate this density function by (1) non-parametric way of Kernel Density Estimation: $f(X_t|\theta) \approx \hat{f}(X_t|\theta) = KDE\big(X_{1:T}(\theta)\big)$. This is simply a histogram smoothing of the simulation data generated from an ABM with a set of parameters. Another approach proposed by the study is to use (2) a Gaussian density $f(X_t|\theta) \approx N(g^*(\theta), \sigma_\epsilon^2 \mathbb{I}_K)$ rather than KDE. The use of a parametric distribution can clearly reduce computational costs, which could be prohibitive in case of multivariate KDE, at the expense of flexibility or expressive power. In both cases, a likelihood of the parameters is calculated by evaluating the constructed density at observed data points: $\mathcal{L}(\theta; X^o_{1:T}) = \prod_t \hat{f}(X^o_t|\theta)$. They also applied (3) Approximate Bayesian Computation that directory approximates a likelihood function by a 1-0 indicator function: $p(X^o_{1:T}|\theta) \approx \mathbb{I}\big(d\big[\mu\big(X_{1:T}(\theta)\big), \mu(X^o_{1:T})\big] < h\big)$, where $\mu(\cdot)$ is a summary statistics, $d[\cdot,\cdot]$ is a distance measure, and h is a threshold for distance. This approach could potentially perform well in the wider class of ABMs with a relatively cheaper computational burden, only if pre-selection or hand-crafted design of an indicator function is appropriate to approximate the target likelihood. However, such a good pre-selection is difficult in practice.

2.4.Likelihood-free Approach: BayesFlow

In this paper, I opt to adopt a novel likelihood-free Bayesian inference of *BayesFlow* proposed by Radev et al. (2020). This can mitigate the trilemma in the existing methods that all of the following three – higher flexibility, lower computational cost, and smaller arbitrariness cannot be achieved at the same time.

BayesFlow is fully likelihood-free approach, which directly approximates a posterior $p(\theta|X_{1:T})$, rather than a likelihood function $p(X_{1:T}|\theta)$. It does not need to presume ergodicity of ABM, or stationarity of simulated time series. This likelihood-free approach only requires the ability to output simulation data from a forward model, which generates samples of observable variables by a deterministic function G of parameters θ and independent noises ξ with known distribution as:

$$X_t \sim p(X|\theta) \Leftrightarrow X_t = g(\theta, \xi_t) \text{ with } \xi_t \sim p(\xi),$$
or T samples simultaneously as:
$$X_{1:T} \sim p(X_{1:T}|\theta) \Leftrightarrow X_{1:T} = G(\theta, \xi_{1:T}) \text{ with } \xi_{1:T} \sim p(\xi). \quad (4)$$

[1] Initial micro-states $S_0 = \{s_{i,0}\}_{i=1}^N$ could be included in parameters or sampled from the given distribution as stochastic elements.

Here, the likelihood $p(X_{1:T}|\theta)$ is only implicitly defined and need not be available in closed-form. Therefore, BayesFlow can seamlessly be applied to an agent-based model since a simulation data generation function G of an ABM in Eq. (3) meets this sampling requirement.

The goal of BayesFlow is to approximate the target posterior by the parameterized approximate posterior as accurately as possible:

$$p(\theta|X_{1:T}) \approx p_\phi(\theta|X_{1:T}).$$

BayesFlow utilizes a conditional invertible neural network (cINN) to this objective. In other word, the cINN constitutes an invertible function $f_\phi: \mathbb{R}^d \to \mathbb{R}^d$ parameterized by a vector of parameters ϕ, for which the inverse $f_\phi^{-1}: \mathbb{R}^d \to \mathbb{R}^d$ exists. The approximate posterior p_ϕ is then reparameterized in terms of a cINN f_ϕ that implements a Normalizing Flow (Rezende & Mohamed, 2015) between θ and a standard Gaussian latent variable z:

$$\theta \sim p_\phi(\theta|X_{1:T}) \Leftrightarrow \theta = f_\phi^{-1}(z; X_{1:T}) \text{ with } z \sim N(z|0, \mathbb{I}_d).$$

Namely, the cINN is to be trained so that outputs of its inverse f_ϕ^{-1} follow the target posterior $p(\theta|X_{1:T})$.

2.5.Learning the Posterior

The training objective for the cINN is thus to minimize the Kullback-Leibler (KL) divergence between the target and the model-induced approximate posterior for all possible series of observable variables $X_{1:T}$ as follows:

$$\begin{aligned}\hat{\phi} &= \underset{\phi}{\operatorname{argmin}}\, \mathbb{E}_{X_{1:T}\sim p(X;T)}\left[\mathbb{KL}\left[p(\theta|X_{1:T})||p_\phi(\theta|X_{1:T})\right]\right] \\ &= \underset{\phi}{\operatorname{argmin}}\, \mathbb{E}_{X_{1:T}\sim p(X;T)}\left[\mathbb{E}_{\theta\sim p(\theta|X_{1:T})}\left[\log p(\theta|X_{1:T}) - \log p_\phi(\theta|X_{1:T})\right]\right] \\ &= \underset{\phi}{\operatorname{argmax}}\, \mathbb{E}_{X_{1:T}\sim p(X;T)}\left[\mathbb{E}_{\theta\sim p(\theta|X_{1:T})}\left[\log p_\phi(\theta|X_{1:T})\right]\right] \\ &= \underset{\phi}{\operatorname{argmax}} \iint p(X,\theta;T) \log p_\phi(\theta|X_{1:T})\, dX_{1:T} d\theta\end{aligned}$$

Then, since the forward transmission of cINN outputs by definition a standard Gaussian latent variable $f_\phi(\theta; X_{1:T}) = z$, the density transformation law of random variable enables the reparameterization of the approximate posterior p_ϕ in terms of cINN f_ϕ as follows:

$$p_\phi(\theta|X_{1:T}) = p\left(z = f_\phi(\theta; X_{1:T})\right)\left|\det\left(\frac{\partial f_\phi(\theta; X_{1:T})}{\partial\theta}\right)\right|.$$

This is a fundamental operation of a Normalizing Flow. Incorporating this fact, the training objective can be re-written as:

$$\hat{\phi} = \underset{\phi}{\operatorname{argmax}} \iint p(X,\theta;T)\left\{\log p\left(f_\phi(\theta; X_{1:T})\right) + \log\left|\det J_{f_\phi}\right|\right\} dX_{1:T} d\theta, \tag{5}$$

where J_{f_ϕ} stands for $\partial f_\phi(\theta; X_{1:T})/\partial\theta$ (the Jacobian of f_ϕ evaluated at θ and $X_{1:T}$).

Even in the likelihood-free setting, it is easy to generate samples from $\left(\theta^{(j)}, X_{1:T}^{(j)}\right) \sim p(X,\theta;T)$ with a forward model G and the prior $p(\theta)$ as shown in Eq.(4). Utilizing the M sets of data-generating parameters and corresponding simulated data $\left\{\left(\theta^{(j)}, X_{1:T}^{(j)}\right)\right\}_{j=1}^{M}$, the expectation in Eq.(5) is approximated by the Monte-Carlo estimate as follows:

$$\hat{\phi} = \underset{\phi}{\operatorname{argmax}} \frac{1}{M}\sum_{j=1}^{M} \log p\left(f_\phi\left(\theta^{(j)}; X_{1:T}^{(j)}\right)\right) + \log\left|\det J_{f_\phi}^{(j)}\right|. \tag{6}$$

By taking negative of Eq.(6) and using the fact that $\log N(z|0,\mathbb{I}_d) \propto -\frac{1}{2}\|z\|_2^2$, the training objective for cINN now becomes:

$$\hat{\phi} = \underset{\phi}{\operatorname{argmin}}\, \mathcal{L}(\phi)$$

with

$$\mathcal{L}(\phi) = \frac{1}{M}\sum_{j=1}^{M}\left(\frac{\left\|f_\phi\left(\theta^{(j)}; X_{1:T}^{(j)}\right)\right\|_2^2}{2} - \log\left|\det J_{f_\phi}^{(j)}\right|\right).$$

(7)

The $\mathcal{L}(\phi)$ is a loss function for this posterior approximation task, which can be minimized with any stochastic gradient descent algorithm.

2.6.Summary Network

When training cINN with simulated datasets, Radev et al. (2020) recommended to use a summary network f_ψ to construct an estimate of sufficient statistics that captures all information about θ contained in $X_{1:T}$ in a fixed-size representation $\tilde{X} = f_\psi(X_{1:T})$. Since the number of observations or time points usually varies in a practical conduct of Bayesian inference, the method needs to be generalized to data of variable size T. Furthermore, the training of cINN could be more efficient with the sort of dimensionality reduction as datasets might exhibit some redundancies without any pre-selection of observable variables.

BayesFlow is designed to use a bidirectional LSTM (Graves & Schmidhuber, 2005) as a summary network for time series-data. LSTM network is well known to be able to effectively deal with the long-memory (i.e. non-ergodic and non-stationary) serial data such as natural language sentences. Indeed, Radev et al. (2020) showed high accuracies[2] of BayesFlow even in the tasks to estimate the parameters of potentially chaotic (the Ricker population model) and non-ergodic (the Levy-Flight model) mathematical models.

In the context of ABM estimation, this feature of BayesFlow can unleash full flexibility of agent-based models, as it does no longer require ergodicity or stationarity of its output time series.

The parameters of the summary network are jointly optimized with those of the cINN. Hence, the training objective is now finalized as:

$$\hat{\phi}, \hat{\psi} = \underset{\phi,\psi}{\operatorname{argmax}}\, \mathbb{E}_{X_{1:T}\sim p(X;T)}\left[\mathbb{E}_{\theta\sim p(\theta|X_{1:T})}\left[\log p_\phi\left(\theta | f_\psi\left(X_{1:T}\right)\right)\right]\right]$$
$$= \underset{\phi,\psi}{\operatorname{argmin}}\, \mathcal{L}(\phi,\psi)$$

with

$$\mathcal{L}(\phi,\psi) = \frac{1}{M}\sum_{j=1}^{M}\left(\frac{\left\|f_\phi\left(\theta^{(j)}; f_\psi\left(X_{1:T}^{(j)}\right)\right)\right\|_2^2}{2} - \log\left|\det J_{f_\phi}^{(j)}\right|\right).$$

(8)

2.7.Structure of Invertible Networks

The cINN is constructed as a chain of multiple conditional affine coupling blocks (cACBs). The idea of an ACB was originally introduced by Dinh et al. (2017), which implements an invertible non-linear transformation: $f_{acb}: \mathbb{R}^d \to \mathbb{R}^d$ and $f_{acb}^{-1}: \mathbb{R}^d \to \mathbb{R}^d$. Each ACB consists of four separate fully connected neural networks denoted as $s_1(\cdot), s_2(\cdot), t_1(\cdot), t_2(\cdot)$. These internal networks need not be invertible since they are only evaluated in a forward direction during both the forward and the inverse operations of an ACB. By denoting input vector of f_{acb} as U and output vector as V, the forward and inverse transformations of the vectors are expressed as $f_{acb}(U) = V$ and $f_{acb}^{-1}(V) = U$. This invertibility is realized by splitting the input vector into two parts $U = (U_1, U_2)$ with $U_1 = u_{1:d/2}$ and $U_2 = u_{d/2+1:d}$ and performing the following operations on the split input:

$$V_1 = U_1 \odot \exp\left(s_1(U_2)\right) + t_1(U_2)$$
$$V_2 = U_2 \odot \exp\left(s_2(U_1)\right) + t_2(U_1)$$
$$V = (V_1, V_2)$$

where $\odot$ represents element-wise multiplication. Then, the inverse operation is given by:

$$U_1 = \left(V_2 - t_2(U_1)\right) \odot \exp\left(s_2(U_1)\right)$$
$$U_2 = \left(V_1 - t_1(U_2)\right) \odot \exp\left(s_1(U_2)\right)$$

This formulation ensures that the Jacobian of cINN is a strictly upper or a lower triangular matrix and therefore its

[2] They tested accuracy of recovering the ground-truth parameter values from out-of-sample simulation data.

determinant ($\det J_{f_\phi}$) is quite cheap to compute, which is an important feature when using it in a Normalizing Flow.

Then, ACB is augmented to take the summary statistics $\tilde{X}$ as a conditioning input, so as to switch the pattern of bidirectional transformations along with the values of observations $X_{1:T}$ as follows:

$$V_1 = U_1 \odot \exp\left(s_1\left(U_2, \tilde{X}\right)\right) + t_1\left(U_2, \tilde{X}\right)$$

$$V_2 = U_2 \odot \exp\left(s_2\left(U_1, \tilde{X}\right)\right) + t_2\left(U_1, \tilde{X}\right).$$

This structure is a conditional affine coupling block (cACB). BayesFlow stacks the multiple cACBs to make the whole neural network architecture (i.e. cINN) expressive enough to implement a potentially complex mapping between the d-dimantinal vector of parameters θ and a same dimensional vector of unit Gaussian variables z. Eventually, the entire conditional invertible neural network (cINN) is expressed as a function $z = f_\phi\left(\theta; \hat{X}\right)$, together with the inverse operation $\theta = f_\phi^{-1}\left(z; \hat{X}\right)$.

2.8.Amortized Inference

For most Bayesian inference algorithms, the entire estimation process must be repeated from scratch when dealing with the different observation sequences (e.g. $X_{1:T}^{(i)}$ and $X_{1:T}^{(j)}$ with $i \neq j$). In contrast, Bayes flow realizes *amortized inference*, where estimation is split into a computationally expensive training phase, and a much cheaper inference phase. In the training phase, BayesFlow tries to learn a model to output an approximate posterior $\widehat{p_\phi}(\theta|X_{1:T})$ that works well for any possible observation sequence $X_{1:T}$. Namely, cINN is trained up front so that its inverse operation outputs samples from an approximate posterior given observations: $f_\phi^{-1}(z|X_{1:T}^o) = \hat{\theta} \sim p_\phi(\theta|X_{1:T}^o)$ with $z \sim N(0, \mathbb{I}_d)$. Hence, evaluating the trained model over a specific observation dataset $X_{1:T}^o$ is computationally very chap, so that the upfront training efforts amortizes over multiple inferences.

Putting it all together, a whole procedure of Bayesian inference with the BayesFlow method is summarized as Algorithm 1.

Training (with online simulation data generations)
repeat
  Sample sequence length of observations: $T \sim U(T_{min}, T_{max})$
  Sample a batch of parameters from prior: $\{\theta^{(j)}\}_{j=1}^M \sim p(\theta)$
  Simulate M data sets size T via the data generation function Eq.(4): $\left\{X_{1:T}^{(j)} = G\left(\theta^{(j)}, \xi_{1:T}\right)\right\}_{j=1}^M$
  Pass $\left\{X_{1:T}^{(j)}\right\}_{j=1}^M$ into summary network to obtain summary statistics: $\left\{\tilde{X}^{(j)} = f_\psi\left(X_{1:T}^{(j)}\right)\right\}_{j=1}^M$
  Pass $\left\{\left(\theta^{(j)}, X_{1:T}^{(j)}\right)\right\}_{j=1}^M$ into cINN to obtain $\left\{z^{(j)} = f_\phi\left(\theta^{(j)}; \tilde{X}^{(j)}\right)\right\}_{j=1}^M$
  Compute loss according to Eq.(8)
  Update neural network parameters ϕ, ψ via backpropagation
until convergence to $\hat{\phi}, \hat{\psi}$

Inference (given observed or test data X^o)
Compute summary of the data $\tilde{X}^o = f_{\hat{\psi}}(X_{1:T}^o)$
for $l = 1, \ldots, L$ **do**
  Sample $z^{(l)} \sim N(0, \mathbb{I}_d)$
  Compute inverse $\theta^{(l)} = f_{\hat{\phi}}^{-1}\left(z^{(l)}; \tilde{X}^o\right)$
end
Return $\{\theta^{(l)}\}_{l=1}^L$ as a sample from $p(\theta|X^o)$

Algorithm 1: Bayesian inference with the BayesFlow method

3. Experiments

3.1.Training

All networks were implemented in Python using the *pytorch* library and trained on a single-GPU machine quipped with NVIDIA(R) GTX1050Ti graphics card. Stochastic gradient decent is implemented by Adam optimizer with default setting of *pytorch* package (learning rate of 0.001). Following the original paper of BayesFlow (Radev et al., 2020), *online learning* approach is taken, where data are simulated from an ABM on demand. As the network never experiences the same input data twice, training can continue as long as the loss keeps decreasing without any concern on overfitting in the classical sense. I performed total 40 000 online update steps in the training with each step using a new pair of parameters and simulated timeseries from an ABM. Incidentally, in the both of two examples explained below, just around 20 000 online steps were enough for the neural networks to reach convergence. Meanwhile, if one simulation takes a high computational cost, a researcher

can opt to perform *off-line learning* approach, in which the fixed number of samples according with the computational budget are generated ex-ante from an ABM, and then the widely parallelized batch learning should be performed on GPU. In any case, the converged networks can repeatedly be used to perform amortized inference on a different observation dataset. As for the hyperparameters, I opt to use a default BayesFlow with 5 ABCs, and a summary vector of size 32 obtained through 3-layer bidirectional LSTM without extensive tune-up.

3.2.Performance Validation

To evaluate the performance of applying BayesFlow to ABM estimation, I opt to use the following two simple metrics defined between the ground-truth parameters $\{\theta^{(l)}\}_{l=1}^{L}$ which generate the test simulation datasets $\{X_{1:T}^{o}(\theta^{(l)})\}_{l=1}^{L}$ and the estimated parameters $\{\hat{\theta}^{(l)}\}_{l=1}^{L}$ reproduced from the test datasets. The number of test estimations is one hundred: $L = 100$.

(1) Normalized Rooted Mean Squared Error: $NRMSE = \sqrt{\sum_{l=1}^{L} \frac{(\theta^{(l)} - \hat{\theta}^{(l)})^2}{\theta_{max} - \theta_{min}}}$

(2) Coefficient of determination: $R^2 = 1 - \sum_{l=1}^{L} \frac{(\theta^{(l)} - \hat{\theta}^{(l)})^2}{(\theta^{(l)} - \bar{\theta}^{(l)})^2}$

The competing benchmark for the validation is the non-parametric KDE with MCMC (a random walk Metropolis-Hastings algorithm) proposed by Grazzini et al. (2017), in which 4 000 iterations are conducted for each estimation procedure.

In the actual experiments, I opt to perform Bayesian inference on the ABM: the minimal stock market ABM with 1 parameter which was originally proposed by Cliff & Bruten (1997), taken up for the estimation experiments by Grazzini & Richiardi (2015) and Grazzini et al. (2017).

3.3.A Minimal Stock Market ABM with 1 parameter

This minimal stock market ABM is populated by sellers and buyers who adjust their profit margin $\mu_{i,t}$, and set price for a bid or ask limit orders by watching an order book of market. The limit price of trader i in period t is updated as the following rule:

$$p_{i,t+1} = v_i(1 + \mu_{i,t+1}),$$
$$\mu_{i,t+1} = (p_{i,t} + \Delta_{i,t})/v_i - 1,$$
$$\Delta_{i,t} = \beta(\tau_{i,t} - p_{i,t}),$$

where v_i is a trader specific volume of order, $\tau_{i,t}$ is an implicit target price of a trader that is hiked if the last trade occurred at a higher price, and lowered otherwise. The behavioral parameter β, common to all traders, is the sensitivity of how traders react to the existing gap between the target price $\tau_{i,t}$ and the current price $p_{i,t}$. The higher this β value, the more sensitive traders are to the prices of the others, resulting in more elastic market.

In the validation procedure, for each separate trial of $l = 1, \ldots, 100$, I tried to recover a ground-truth value of the parameter $\beta^{(l)}$ from the simulated time series of the market price from this ABM (i.e. $X_{1:T}^{o}(\beta^{(l)})$) by using BayesFlow and KDE-MCMC, respectively. The values of ground-truth parameter for the test trials are sampled from uniform distribution: $\beta^{(l)} \sim U(0,1)$. The prior distribution for BayesFlow and KDE-MCMC is the same with this: $U(0,1)$.

The results are depicted in Table **1** and

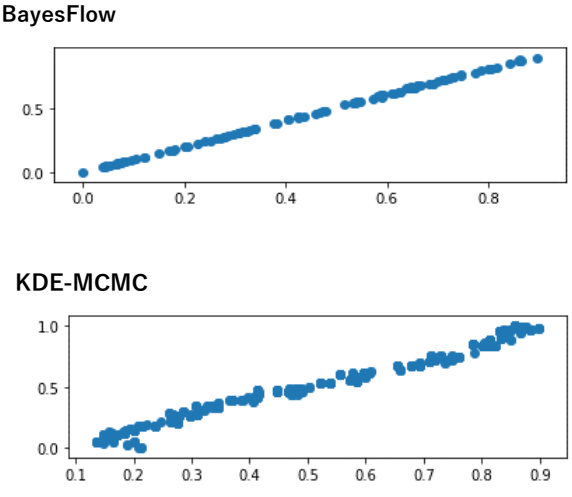

Figure **1**. While the both methods fairly succeeded to

recover the ground-truth parameter from the simulated dataset in most trials, the overall precision measured by NRMSE and R2 is clearly higher in BayesFlow.

	BayesFlow	KDE-MCMC
NRMSE	0.007	0.058
R2	1.000	0.993

Table 1: Performance results on a minimal stock market ABM.

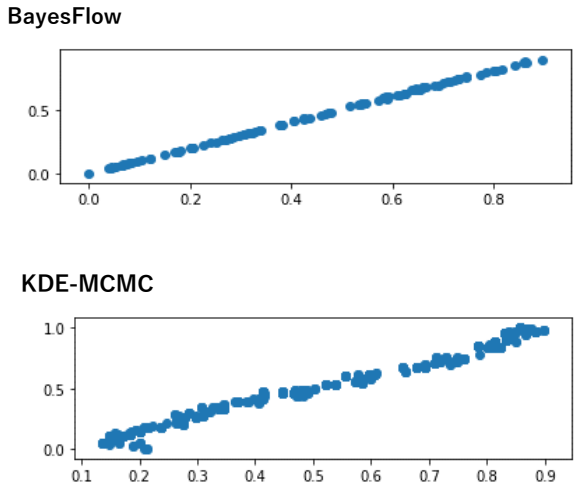

Figure 1: Parameter recovery plots where x-axis takes value of the estimated parameter and y-axis is the ground-truth parameter (L=100).

As I performed 100 separate estimations to calculate the validation metrics, the difference in computational costs became significant between the amortized inference of BayesFlow and the case-by-case inference of KDE-MCMC, even in the minimal 1 parameter ABM. One simulation run of the ABM with the 500 observation periods (where the first 500 periods were discarded as burn-in) takes 1.5 secs. In KDE-MCMC, the likelihood approximation and the Metropolis-Hastings update cost additional 2.5 secs. As I conducted 4 000 iterations in the M-H algorithm for each estimation, the total computational time for the 100 separate estimations was massive 444 hours (1 600 000 =100×4 000×(1.5+2.5) secs). I inevitably parallelized the CPU core processes over the separable 100 estimations. On the other hand, BayesFlow took much shorter 18.9 hours (68 152 secs) in total. While the training phase with 40 000 online training steps costs 68 000 secs as one step takes 1.7 (=1.5 of data generation + 0.2 of SGD) secs, the amortized inference with the 100 separate test datasets only takes 152 (1.5× 100 of test data generation + 2 of inference) secs. Furthermore, as already mentioned above, the training of BayesFlow actually reached the convergence with much fewer steps of 20 000, meaning that computational time could be shortened by half. Meanwhile, it looks unrealistic to cut MCMC iterations to less than 4 000 in order to obtain the accepted samples of more than 1 000.

4. Concluding Remarks

This paper examines the possibility of applying the novel likelihood-free Bayesian inference called BayesFlow proposed by Radev et al. (2020) to the estimation of agent-based models (ABMs). The BayesFlow is a fully likelihood-free approach, which directly approximates a posterior rather than a likelihood function, by learning an invertible probabilistic mapping that implements a Normalizing Flow between parameters and a standard Gaussian variables conditioned by data from simulations. This deep neural network-based method can mitigate the trilemma in the existing methods that all of the following three –higher flexibility, lower computational cost, and smaller arbitrariness cannot be achieved at the same time. As a result of the experiments, BayesFlow certainly achieved the superior accuracies in the validation task of recovering the ground-truth values of parameters from the simulated datasets, in a minimal stock market ABM. The method did not involve any extensive search of the hyperparameters or hand-crafted pre-selections of summary statistics, and took a significantly shorter computational time than an existing non-parametric MCMC approach.

References

[1] Assenza, T., Delli Gatti, D., & Grazzini, J. (2015). Emergent dynamics of a macroeconomic agent based model with capital and credit. Journal of Economic Dynamics and Control, 50, 5–28. https://doi.org/10.1016/j.jedc.2014.07.001

[2] Caiani, A., Godin, A., Caverzasi, E., Gallegati, M., Kinsella, S., & Stiglitz, J. E. (2016). Agent based-stock flow consistent macroeconomics: Towards a benchmark model. Journal of Economic Dynamics and Control, 69, 375–408. https://doi.org/10.1016/j.jedc.2016.06.001

[3] Cliff, D., & Bruten, J. (1997). Zero is not enough: on the lower limit of agent intelligence for continuous double auction markets. In HP Laboratories Technical Report (Issues 97–141).

[4] Dinh, L., Sohl-Dickstein, J., & Bengio, S. (2017, May 27). Density estimation using real NVP. 5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings. http://arxiv.org/abs/1605.08803

[5] Gallegati, M., & Richiardi, M. G. (2009). Agent Based Models in Economics and Complexity. In Encyclopedia of Complexity and Systems Science (pp. 200–224). Springer New York. https://doi.org/10.1007/978-0-387-30440-3_14

[6] Ghonghadze, J., & Lux, T. (2016). Bringing an elementary agent-based model to the data: Estimation via GMM and an application to forecasting of asset price volatility. Journal of Empirical Finance, 37, 1–19. https://doi.org/10.1016/j.jempfin.2016.02.002

[7] Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Networks, 18(5–6), 602–610. https://doi.org/10.1016/j.neunet.2005.06.042

[8] Grazzini, J., & Richiardi, M. (2015). Estimation of ergodic agent-based models by simulated minimum distance. Journal of Economic Dynamics and Control, 51, 148–165. https://doi.org/10.1016/j.jedc.2014.10.006

[9] Grazzini, J., Richiardi, M. G., & Tsionas, M. (2017). Bayesian estimation of agent-based models. Journal of Economic Dynamics and Control, 77, 26–47. https://doi.org/10.1016/j.jedc.2017.01.014

[10] Lamperti, F., Roventini, A., & Sani, A. (2018). Agent-based model calibration using machine learning surrogates. Journal of Economic Dynamics and Control, 90, 366–389. https://doi.org/10.1016/j.jedc.2018.03.011

[11] Lux, T. (2018). Estimation of agent-based models using sequential Monte Carlo methods. Journal of Economic Dynamics and Control, 91, 391–408. https://doi.org/10.1016/j.jedc.2018.01.021

[12] Radev, S. T., Mertens, U. K., Voss, A., Ardizzone, L., & Köthe, U. (2020). BayesFlow: Learning complex stochastic models with invertible neural networks. http://arxiv.org/abs/2003.06281

[13] Rezende, D. J., & Mohamed, S. (2015). Variational Inference with Normalizing Flows. 32nd International Conference on Machine Learning, ICML 2015, 2, 1530–1538. http://arxiv.org/abs/1505.05770