Neural Nets for Indirect Inference

Michael Creel

Universitat Autònoma de Barcelona, BGSE and MOVE [1]

November, 2016

Abstract

For simulable models, neural networks are used to approximate the limited information posterior mean, which conditions on a vector of statistics, rather than on the full sample. Because the model is simulable, training and testing samples may be generated with sizes large enough to train well a net that is large enough, in terms of number of hidden layers and neurons, to learn the limited information posterior mean with good accuracy. Targeting the limited information posterior mean using neural nets is simpler, faster, and more successful than is targeting the full information posterior mean, which conditions on the observed sample. The output of the trained net can be used directly as an estimator of the model's parameters, or as an input to subsequent classical or Bayesian indirect inference estimation. Examples of indirect inference based on the output of the net include a small dynamic stochastic general equilibrium model, estimated using both classical indirect inference methods and approximate Bayesian computing (ABC) methods, and a continuous time jump-diffusion model for stock index returns, estimated using ABC.

Keywords: neural networks; indirect inference; approximate Bayesian computing; machine learning; DSGE; jump-diffusion

1. Introduction[2]

Neural networks are a well known tool in machine learning, and have been of interest to econometricians for many years. Early work includes Gallant and White [12] and Hornik et al. [18], who establish conditions under which certain feed-forward neural networks can learn a nonlinear mapping to any required degree of accuracy. A textbook reference is Haykin [17], and a survey from the econometric point of view is Kuan and White [20]. In more recent years, computational advances, such as graphical processing unit (GPU) computing, that allow for use of "deep learning" nets with many layers,

[1]Address: Dept. of Economics and Economic History, Edifici B, Universitat Autònoma de Barcelona, Bellaterra (Barcelona), Spain, 08193. email: michael.creel@uab.cat.

[2]Code to replicate the examples and to use the methods for other models is available at https://github.com/mcreel/NeuralNetsForIndirectInference.jl.

combined with the availability of large data sets ("big data"), have stimulated a resurgence of interest in neural networks for applications such as classification and labeling of images (LeCun et al. [21]). Economists have not missed out on the big data and machine learning trends, and recent work seeks to integrate econometricians' concern with causality with machine learning methods' ability to predict well (e.g., Varian [25]).

This paper proposes to use neural nets for the estimation of the parameters of a simulable model, in situations where indirect inference (II - Gouriéroux et al. [15]) or approximate Bayesian computing (ABC - Beaumont et al. [1]) might be used. Let θ be a draw of a parameter vector, from a prior. The model may be simulated at the draw, to give a random sample, Y_n, of size n, from which the statistic vector $Z_n(Y_n)$ may be computed. This can be repeated many times, to give a large body of independent simulated realizations (θ^s, z_n^s), $s = 1, 2, ..., S$. This paper proposes to use neural networks with simulated data to learn the limited information posterior mean, $E(\theta|Z_n)$, or a good approximation to it. The inputs to the net are the simulated statistics, z_n^s, and the net is trained to fit the parameter values, θ^s, as well as possible according to a criterion such as squared error loss. In the context of simulation-based estimation, it is feasible to provide as much data as is needed to train and test a net that is large enough, in terms of the numbers of hidden layers and neurons, to predict the outputs well. Once the net is trained, when it is provided with the input value z_n, the real sample value of the statistic vector, its output, denoted by $\hat{\theta}(z_n)$, is an estimate of the posterior mean $E(\theta|Z_n = z_n)$, and it is thus a reasonable choice as a point estimator of the unknown parameter.

Blum and François [3] also fit $E(\theta|Z_n = z_n)$ using a feed forward neural network. The nets they work with are small, with a single hidden layer that has only 4 hidden units ("neurons"). In contrast, this paper uses much larger nets, with multiple hidden layers and many more neurons. The most important conceptual difference with what is proposed here is that Blum and François [3] assume (their eqn. 8 and beginning of page 66) that a net fit using kernel weighting based on proximity to z_n will continue to be accurate for input values different from z_n. This assumption allows them to use the output of the fitted net, along with a second net which is used to model the posterior log variance, in order to directly sample from the posterior. In the present paper, no assumption of the global validity of a net fit with local weighting is made, and local weighting is not used. Instead, in order to obtain a good global fit to $E(\theta|Z_n = z)$, for any z which may result from a parameter draw from the prior or parameter space, much larger nets are used, because small nets are not able to learn the mapping well. With the advances which have been made in the software which is available for working with neural nets, the use of much larger nets is now straightforward.

Jiang et al. [19] propose to use deep (multiple hidden layer) neural networks to learn the full information posterior mean, $E(\theta|Y_n)$, where Y_n is the full sample, rather than the limited information posterior mean, $E(\theta|Z_n)$, which is the object that this paper proposes to approximate. This apparently small difference has some important consequences in terms of practicality of application and final performance of the estimator. In particular, as is seen in Section 4, below, using a net to fit $E(\theta|Z_n)$ is both much

easier to do, and performs much better. While the true full information posterior mean is a more efficient (infeasible) estimator than is the limited information posterior mean, the output of a neural net is an estimator of the respective posterior mean. This paper shows that neural nets used to fit the full information posterior mean are too inaccurate for the theoretical ranking of efficiencies to hold, and, in practice, an accurate estimate of the less efficient limited information posterior mean, which is attainable using neural nets, provides a better estimator. Jiang et al. [19] also point out that the output of a trained net can be used not only as a point estimator, but also as the input to one of the well-known simulation-based econometric methods which operate through a statistic, such as indirect inference or approximate Bayesian computing (ABC). If this is done, the well developed asymptotic theory for such methods will apply to the final estimator, so statistical inference will be possible. Because their proposal takes the full sample as the input to the net, the output can be thought of as an automatically constructed statistic for ABC (or classical indirect inference) estimation. In contrast, the proposal of the present paper requires specifying a candidate set of statistics, Z_n, so the process is not fully automatic. However, performance using an informative set of statistics can be much better than what is obtained with fully automatic statistics, as long as the candidate set is chosen reasonably well. Because the model is fully specified, prior knowledge and analysis can usually suggest what will be informative candidate statistics. Examples are given, below.

The proposed method has several advantages, which are illustrated in the examples which follow. These include:

Speed. The network may be trained ahead of time, so that when sample data becomes available, an estimate may be obtained essentially instantaneously, as it is given by a closed form expression. This may be advantageous for applications such as finance, where early access to results carries a premium. Additionally, sophisticated software packages, which use massively parallel processing through GPU computing, are available to train and test large nets quickly and easily.

Offers a solution to the curse of dimensionality. With complex models, it may not be obvious which statistics will be most informative for the parameters of the model, and sufficient statistics are unlikely to exist. For this reason, we may begin with a fairly large set of candidate statistics (that is, Z_n may be of high dimensionality). Neural networks have been shown to work well for classification of image data, such as the MNIST data (LeCun et al. [21]), where each input is a 784 dimensional vector of real numbers. Neural networks are able to obtain error rates of less than 1% for this data set (LeCun et al. [21], see also `http://yann.lecun.com/exdb/mnist/`). With large enough training and validation sets, and perhaps making use of regularization methods, neural nets can deal effectively with high dimensional input data, because they learn to down-weight uninformative or irrelevant inputs. In the present simulation-based estimation context, the training and testing data sets may be made as large as is required for effective training. The cost of a using a high dimensional input is that training times will be longer, but training can still be done quite quickly for the size of problems that econometricians are likely to work with, as is seen in the examples

below, and it may be done before the estimation data is even available, as noted above. Also, the output of the net can be viewed as a statistic which is highly informative for the parameter, and which is of the minimal dimensionality which retains identification, as it has the same dimension as the model's parameter vector. This statistic can be used for subsequent estimation by indirect inference or ABC.

Statistical efficiency. Given a large enough network, the posterior mean of the parameters, conditional on the statistics, $E(\theta|Z_n)$, can be learnt with excellent accuracy. Certain configurations of nets have been shown to possess a "universal approximation" property (Gallant and White [12], Hornik et al. [18]), though the "universal approximation" properties of deep neural nets is still an open question. In the full information context, the posterior mean has the same first order asymptotic distribution as does the maximum likelihood estimator (Bickel and Yahav [2]), and is thus fully efficient. In the present context of working with a statistic rather than the full sample, we speculate that the output of a large enough net, that is trained well enough, should be approximately efficient in a *limited information* context. The choice of the initial statistic vector Z_n will of course condition the statistical efficiency of the net's output. Because of neural nets' ability to deal with high dimensional inputs, it is feasible to provide a large number of statistics as inputs, so that the limited information carried by the statistic is a good proxy for the full information of the sample, in an effort to obtain an estimator which is approximately fully efficient. However, this paper does not go into an asymptotic analysis, which is left for future work.

Simplicity. Modern software for specification and training of neural nets is easy to use and can take advantage of powerful computational resources such as graphical processing units (GPUs), which allow for massively parallel computing. To implement the proposed method, a user simply needs simulate from the model to generate a large database of parameter values and associated statistics. The rest of the computations can be done using any of a number of powerful packages for neural networks. Example code using the `MXNet.jl` (https://github.com/dmlc/MXNet.jl) framework accompanies this paper. With this code, an interested user can fit a neural net to their own model simply and easily, by adjusting the references to the input data sets, and perhaps changing the number of layers and/or the number of neurons in given layer.

A disadvantage of the proposed method is that the output of the trained network is not an extremum estimator, as it is essentially impossible to train a large net, with many thousands of parameters, to convergence to a global minimum. Nor is it a Bayesian estimator, though it is an approximation to the posterior mean. Because of this, asymptotic theory that supports hypothesis testing, confidence intervals, and so forth, is lacking. However, as has been noted, the output of the trained net can be used as the input to one of the versions indirect inference, which enjoy a well developed asymptotic theory. Examples of classical and Bayesian indirect inference using the output of the net as a statistic are provided in what follows. The next section discusses indirect inference methods, both classical and Bayesian, that rely on statistics, to understand how the proposed methods can ameliorate some of the difficulties associated with these methods. Section 3 provides a brief review of feed-forward neural nets, and describes the

proposed method in detail. Section 4 gives results for two simulation examples, while Section 5 applies the methods to the estimation of a continuous time jump-diffusion model for daily returns of the Standard & Poor's 500 index. A final section offers conclusions and directions for further research. Code to replicate the results of the paper, which can also serve as a template for application of the methods to other data and models, is available at https://github.com/mcreel/NeuralNetsForIndirectInference.jl.

2. Indirect inference

Suppose we have a fully specified model indexed by a parameter $\theta \in \Theta \subset \mathbb{R}^k$. Let Y_n be a sample, generated at the unknown true parameter value θ_0. Consider a statistic $Z_n = Z_n(Y_n)$, chosen by the researcher, with sample realization z_n. This could be composed of simple sample moments, more complicated statistics, such as the parameter estimates of an auxiliary model, or a combination of both. A continuous-updating (CU) GMM estimator is

$$\hat{\theta}_{CU} = \arg\min_{\theta\in\Theta} \left(z_n - E_\theta\left[Z_n\right]\right)' \Omega_n^{-1}\left(\theta\right)\left(z_n - E_\theta\left[Z_n\right]\right), \tag{1}$$

where $\Omega_n\left(\theta\right) = V_\theta(Z_n) = E_\theta\left[\left(Z_n - E_\theta\left[Z_n\right]\right)\left(Z_n - E_\theta\left[Z_n\right]\right)'\right]$ and $E_\theta\left[\cdot\right]$ denotes expectations implied by the model evaluated at θ. In many cases, analytical expressions for $E_\theta\left[Z_n\right]$ and $\Omega_n\left(\theta\right)$ are not available, and instead simulated versions are used. The method of simulated moments (McFadden [23], Pakes and Pollard [24]) sets Z_n to sample moments; indirect inference (Gouriéroux et al. [15]) sets it to the parameter estimate of an auxiliary model, and the efficient method of moments (EMM - Gallant and Tauchen [13]) sets it to the score of an auxiliary model. In principle, moments can be specified to use combinations of these ideas, the general idea may be referred to as indirect inference. Some variants of approximate Bayesian computing seek to work with the posterior distribution of θ given $Z_n = z_n$, and thus, this form of ABC estimation can be thought of as Bayesian indirect inference. Such estimators allow for econometric estimation of complex models that are not amenable to more traditional methods of estimation. The previously cited papers give some examples of applications, but interesting new applications continue to appear: a recent example is given by Grazzini and Richiardi [16], who use simulated moments to estimate agent-based models. A common asymptotic theory applies to the classical and Bayesian forms of indirect inference (Creel and Kristensen [8]). Two general issues affect these methods: the curse of dimensionality, and high (and complex) computational burden.

To begin with the first, methods which rely on a statistic (though the singular form is used, it is understood that the statistic may be multi-dimensional) will not, in general, achieve full asymptotic efficiency, because sample information is filtered through the statistic, rather than used directly. If the statistic were sufficient, then there would be no information loss, but in most cases of interest, there will exist no sufficient statistic, or no sufficient statistic will be known. The score function of the maximum likelihood (ML) estimator would constitute a statistic that, asymptotically,

looses no information (this is what motivates the EMM estimator), but in most cases, one contemplates using an estimator that is based on a statistic when the ML estimator is not feasible, so its score will not be available. In order to retain as much of the sample information as is possible, one may think of using a high dimensional statistic. This idea is also motivated by the result that GMM estimators that use a full set of moment conditions are asymptotically more efficient than GMM estimators which use a subset of the moment conditions. For asymptotic efficiency, more is better. However, the cost of using too many statistics is that small sample bias and/or variance may explode. In the context of GMM estimation, these considerations are explained in detail by Donald et al. [11], who also investigate methods for selecting moments out of a candidate set. In the context of ABC, Blum et al. [4] provide a survey of methods that have been proposed for selection of statistics. The method proposed here, which is adapted from the ideas of Jiang et al. [19] to work with a net that takes an initial vector of candidate statistics as its input, is shown in the examples below to work very well: it provides minimal dimensional statistics that are very informative for the parameters. It is also quick and simple to implement, taking advantage of the "black box" nature of neural nets: they fit well, without requiring precise or expert tuning.

Indirect inference methods can also be computationally demanding. The versions which are extremum estimators may present an objective function which has many local minima, so simply computing the estimator may be difficult (Section 5, below, provides an example). This point is made by Chernozhukov and Hong [6], who propose to use Markov chain Monte Carlo (MCMC) to compute a quasi-Bayesian alternative to an extremum estimator that is difficult to compute. ABC estimators also rely on methods such as MCMC or sequential Monte Carlo. MCMC is not a universal, automatic solution to the problem, as it requires careful choice of the proposal density in order to perform well, and it is inherently computationally demanding, even when tuned well. The computational demand of both GMM-type and ABC methods is compounded when the chosen statistic upon which they are based is of high dimension, or contains weakly informative or irrelevant statistics. High dimension of the statistic increases the chances that multiple local minima of the GMM criterion function will be present, because the weight matrix, $\Omega_n^{-1}(\theta)$, is likely to become poorly conditioned when the number of moments is large. For methods that use MCMC, it will be difficult to find simulated statistics which are close to the observed sample realization, when the dimension is high, so it will be difficult to obtain MCMC draws which are accepted, which means that the chain will have to be very long to obtain accurate evaluation of posterior quantities.

This is not to argue that indirect inference methods are not useful or reliable - without doubt, they are, and in fact they are used in what follows. The point I hope to make is that care is needed when applying the methods. Exercising this care requires expertise and time, on the part of the practitioner. Furthermore, a considerable amount of time may also be required simply to perform the computations, once the details of the tuning process have been worked out. For an application which may be time-sensitive, such as the estimation of a model to be used for financial analysis, the time demands of

the methods may be a drawback. The neural net methods proposed in the next section are based on the same idea which underlies indirect inference - using the information in a statistic to learn about a parameter - but they have the virtue of requiring little expertise or care, as they are very simple and can be tuned using objective methods (cross validation), in a very straightforward way. Also, all of the tuning steps may be done before the sample data is obtained. Computing the estimate, given a net which has been trained ahead of time, is essentially instantaneous. Finally, the methods deal very well with high dimensional statistics, some elements of which may be irrelevant. A network can be trained to ignore irrelevant inputs, which effectively sidesteps the curse of dimensionality. Once these steps are done, the output can be taken as an estimator, or, optionally, it can be used as a minimal dimension input to subsequent indirect inference. The fact that the dimension is reduced to the minimal value which maintains identification of the parameters helps to address the difficulties associated with indirect inference which have been pointed out in this section.

3. Neural networks

Neural networks are a well known tool in many fields, and there are many presentations, both academic and more informal, of various structures that can be used. For this reason, the presentation here is brief. For more details and references, see Kuan and White [20], Haykin [17], LeCun et al. [22] and Jiang et al. [19].

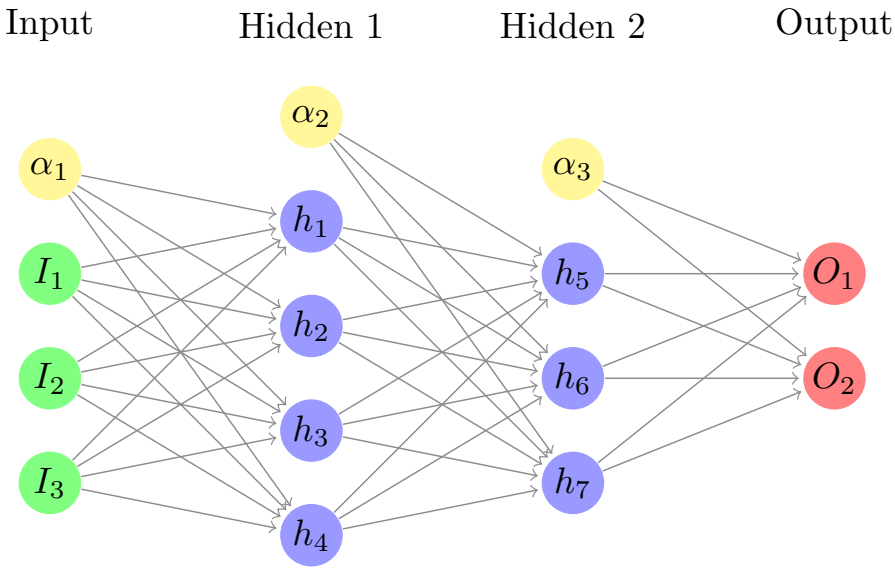
Here, we consider a simple feed forward neural net for regression of an output in R^K upon an input in R^G. A typical feed forward net is depicted in Figure 1, which maps 3 inputs to 2 outputs. The inputs to the net, I_1, I_2, and I_3, are scalar real numbers, as are the outputs O_1 and O_2. The net has two hidden layers, formed by 4 and 3 hidden nodes or neurons, h_1,h_2,...,h_7, and an output layer, which gives the values of the two outputs O_1 and O_2. The values α_1, α_2 and α_3 are vectors of "bias" parameters, which are discussed below.

In general, a net is a series of transformations of the inputs. Each of the transformations is referred to as a layer. The inputs themselves constitute the zeroth layer, and the final result of the transformations is the output layer. A layer, H_j, is a vector of real numbers, which is the result of the j^{th} in the series of transformations. Let H_0 be the G dimensional vector of inputs. Suppose that there are P layers. Let n_j be the number of neurons in the j^{th} layer, $j = 1, 2, ..., P$. The value taken by a neuron in the j^{th} layer is the result of the layer's "activation function", $f_j(\cdot)$, applied on an element-by-element basis to an affine function of the inputs to the layer. The relationship between the layers is given by

$$H_j = f_j(\alpha_j + \beta_j H_{j-1}),\ j = 1, 2, ..., P, \tag{2}$$

where α_j is a n_j dimensional vector of parameters (these are known as bias parameters in the neural net literature), and β_j is a $n_j \times n_{j-1}$ matrix of parameters. The layers $1, 2, ..., P-1$ are referred to as hidden layers, while layer P is the output layer. The

Figure 1: A simple neural net



input to the first hidden layer, known as the input layer, is simply the input data, $H_0 \in \mathbb{R}^G$. The output of the net is the final layer, $H_P \in \mathbb{R}^K$. When using a net for regression, the last activation function, $f_P(\cdot)$, is simply an identity function, so that $H_P = \alpha_P + \beta_P H_{P-1}$. The reason that an activation function is used with the hidden layers is that this is what allows the net to approximate a nonlinear mapping. If all activation functions were identity functions, the entire net would reduce to an over-parameterized linear regression model. In this paper, the activation function that is used for the hidden layers is the "rectified linear unit" (ReLU) function, $f(x) = \max(0, x)$, a very widely used choice in modern deep learning applications.

The inputs to a neural net may be the result of preprocessing of raw inputs. For example, standardization and normalization are often useful transformations. In the case of trying to learn $E(\theta|Z_n)$, it may be useful to use as input to the net the residuals of a linear regression of θ upon Z_n, so that the net only needs to learn the nonlinear part of the mapping. This may help to reduce the size of the net that is needed to learn the mapping well. As noted by LeCun et al. [22], attempting to train a net when the inputs are correlated is a much more difficult problem than is training a net with uncorrelated inputs. With time series data, the sample Y_n will consist of a vector of dependent elements. Jiang et al. [19] use the full sample Y_n as the input to the net. The arguments of LeCun et al. [22] may be relevant for explaining the relatively poor results obtained by Jiang et al. [19] for the moving average model discussed in Section 4.2, below, compared to an approach which uses a statistic as the input to the net. One would like to use a preprocessor that allows use of a relatively simple net, without serious loss of information. A well chosen statistic may serve as such a preprocessor.

A neural net may contain many, many hidden parameters. For example, for the DSGE model presented below, the number of inputs, G, is 40, and the number of

outputs, K, is 9. Suppose the net has two hidden layers, of size 300 and 40, respectively. Then there are 300×40 parameters in the β_1 matrix of the first layer, and there are 40 elements in the α_1 vector. Similarly, in the second hidden layer, there are $40 \times 300 + 40$ parameters, and there are $9 \times 40 + 9$ parameters corresponding to the output layer. Thus, the total number of parameters is 24449. One can see that such a feed forward net is an extremely highly parameterized nonlinear regression model. Furthermore, the model is not identified, as a reordering of the neurons in a given layer leads to an observationally equivalent model. The question arises: how are the parameters to be "estimated" and the model to be fit?

First, the loss function must be chosen. A common choice, for real valued outputs, and which is used here, is the least squares loss function. This can be justified on Bayesian grounds, and it facilitates computing the gradient by back propagation, as is discussed below. With least squares loss, training the neural net is conceptually the same as performing nonlinear least squares, with the difference that the model under consideration is extremely highly parameterized, in comparison with ordinary econometric models. A second difference is that the available data is typically split into a training set and a validation set. Having done this, the method that is most widely used for optimization of the loss function is simple steepest descent, or a variation known as stochastic gradient descent. In the neural net literature (see, for example, LeCun et al. [22]), this is known as back propagation, as the gradient may be computed using the chain rule, working from the prediction error in the last layer backwards to the inputs. Stochastic gradient descent computes the gradient for a subset ("batch") of the training set observations, rather than for the entire collection of inputs. This has two advantages: it is faster than computing the gradient using the full set of observations, and, more importantly, a stochastic gradient, computed using a relatively small and randomly chosen set of observations, can allow the solver to jump out of the region around a local minimum, so that the chances that a good solution is found increase, for a given starting value. Typically, many, many iterations are done, using a small and often decreasing step size (the "learning rate" in neural net parlance) at each iteration. The performance of the network, as it learns, is periodically checked using the validation, or test set of observations. For a reader with a background in econometrics, this the familiar idea of cross validation using a hold out sample. The net may be trained, using back propagation and the training set, to the point where there is no improvement in the ability to fit the cross validation sample. For use of neural nets with real data sets, which may be of limited size, regularization methods are often used. These methods can help to stabilize the net's hidden parameters, or to encourage that some are set to zero (see Girosi et al. [14]). In the present case, the training and testing data sets may be made as large as is desired, so regularization is of limited importance. There are many wrinkles available in tuning the learning method. Fortunately, efficient software is available, which incorporates these possibilities. For this reason, we do not go into further details. A complete example of how to train a net is given, below.

If a net is trained to the point where there is no improvement in its ability to fit a

cross validation sample, then, typically, training is stopped, and the final fitted net is used for forecasting the outputs. Let $\hat{\theta}(z_n)$ be the output of the net when training is completed, evaluated at the sample value of the statistic, z_n. While $\hat{\theta}(z_n)$ should be a very good approximation to $E(\theta|Z_n = z_n)$, if the net is large enough and trained well enough, it is not an extremum estimator - the gradient at the end of training will not be zero. However, with such a highly parameterized model that is not identified without additional restrictions, it is first doubtful that it would actually be possible to train the net to the point where the gradient were zero, and secondly, *it would not be desirable to do so*, as the over-parameterized model would be fitting the errors of the training set, rather than the regression function. Even though the direct output of the net, $\hat{\theta}(z_n)$, is not a conventional extremum estimator, it is an estimator which we may use, though it does not enjoy the asymptotic theory that applies to extremum estimators. We may note that the use of stochastic gradient descent, plus termination based on a randomly generated test set implies that the final output of the net constitutes the result of a stochastic optimization process. The effect of stochastic optimization algorithms on econometric estimators was studied by Winker and Maringer [26]. It may be possible to extend these results to formalize the properties of $\hat{\theta}(z_n)$, but this possibility is left for future work.

Instead of using the output of the net as an estimator, it may also be used as a statistic to define an indirect inference or ABC estimator. Such an II estimator is

$$\hat{\theta}_{II} = \arg\min_{\theta\in\Theta} \left(\hat{\theta}(z_n) - \frac{1}{S}\sum_{s=1}^{S}\hat{\theta}(z_n^s(\theta))\right)' \left(\hat{\theta}(z_n) - \frac{1}{S}\sum_{s=1}^{S}\hat{\theta}(z_n^s(\theta))\right), \tag{3}$$

where each $z_n^s(\theta)$ is a simulated statistic generated at the parameter value θ. Because the dimension of $\hat{\theta}(z)$ is the same as that of the model's parameter vector, the econometric model is just identified, and there is no need for specifying a weighting matrix. This also implies that the objective function value at the II estimate should be zero (supposing that the initial statistics Z_n are differentiable functions of the model's parameters). Finally, the simple direct estimator $\hat{\theta}(z_n)$ provides an excellent starting value for the minimization algorithm. All of these factors may facilitate computation of the indirect inference estimator. This idea is pursued in Section 4.1, below, for estimation of a small DSGE model. In the same vein, Jiang et al. [19] propose to use the output of the trained net, $\hat{\theta}(z_n)$, as a statistic for ABC estimation. This idea is illustrated by examples in Jiang et al. [19], and below, in Section 5, where a jump-diffusion model is estimated via ABC. The benefit with using II or ABC is that theoretical results on inference, which do not apply to the raw output of the net, become available.

One final issue which has not been discussed is how to specify the net: how many hidden layers should be used, and how many neurons per layer. This can easily be done by comparing the abilities of different specifications to fit the out of sample testing data. Even with large training and testing data sets, modern deep learning packages which allow for massively parallel computing using graphical processing units (GPUs) are efficient enough so that a specification search to configure the net is not a very time

consuming process.

4. Simulation examples

This section presents two simulation examples which illustrate the proposed methods. This first is an example of an economic model that is similar in complexity to problems of real research interest, which shows the feasibility and good performance of the methods. The second replicates a simple example considered by Jiang et al. [19], which allows us to explore the trade-offs between targeting the limited information posterior mean, $E(\theta|Z_n)$, versus the full information posterior mean, $E(\theta|Y_n)$.

4.1. DSGE model

Creel et al. [7] use ABC methods to estimate a small dynamic stochastic general equilibrium (DSGE) model, and this same model is used here to illustrate direct estimation using a neural net, and also estimation by indirect inference, using the output of the neural net as the statistic to match. The model is as follows: A single good can be consumed or used for investment, and a single competitive firm maximizes profits. The variables are: y output; c consumption; k capital; i investment; n labor; w real wages; and r return to capital. The representative consumer maximizes expected discounted utility $E_t \sum_{s=0}^{\infty} \beta^s \left(\frac{c_{t+s}^{1-\gamma}}{1-\gamma} + (1 - n_{t+s})\eta_t \psi \right)$ subject to the budget constraint $c_t + i_t = r_t k_t + w_t n_t$ and the accumulation of capital $k_{t+1} = i_t + (1 - \delta)k_t$. There is a preference shock, η_t, that affects the desirability of leisure, which evolves according to $\ln \eta_t = \rho_\eta \ln \eta_{t-1} + \sigma_\eta \epsilon_t$. The competitive firm produces the good y_t using the technology $y_t = k_t^\alpha n_t^{1-\alpha} z_t$. Technology shocks z_t evolve according to $\ln z_t = \rho_z \ln z_{t-1} + \sigma_z u_t$. The innovations to the preference and technology shocks, ϵ_t and u_t, are independent standard normal random variables. The good can be allocated by the consumer to consumption or investment: $y_t = c_t + i_t$. The consumer provides capital and labor to the firm, and is paid at the rates r_t and w_t, respectively. The model has nine parameters: $\alpha, \beta, \delta, \gamma, \psi, \rho_z, \sigma_z, \rho_\eta, \sigma_\eta$. Rather than estimate ψ, instead, the nonstochastic steady state value of hours (n) is estimated, and ψ is recovered from this and the other parameter estimates. Given parameter values, the models is solved and simulated to give a sample of 160 time series observations on 5 observable variables: output, consumption, hours, the wage rate, and the interest rate. Using the data, 40 candidate statistics are computed. These include means, standard deviations, estimated vector autoregression of order 1 (VAR(1)) coefficients, estimated VAR(1) shock covariances, and several other statistics from auxiliary regressions which are suggested by the structure of the model. The chosen candidate statistics are illustrative of the fact that knowledge of the structure of the model allows selection of candidate statistics that should be informative for the parameters of the model. VAR models are widely used to model macroeconomic data, and the use of a low order VAR model to capture dynamics and correlations across variables is an obvious choice for data simulated from a DSGE model. Means and standard deviations are natural choices to capture levels and variability. For example, the sample mean of n, labor hours, will clearly be informative about the nonstochastic

steady state value of labor, which is one of the 9 parameters that are estimated. Full details of the candidate statistics are in the file `aux_stat.m`, which is in the DSGE directory of the code archive which accompanies this paper. The choice of candidate statistics is not automatic, it is informed by the structure of the model that is being estimated, as interpreted by the econometrician. The econometrician's knowledge and intuition are an important input at this stage of the process. Careful study of the model being estimated should be undertaken in order to successfully form the candidate set of statistics.

From the candidate set, II or ABC methods often use a selection procedure to chose the final statistics used for estimation, in order to reduce dimensionality. Here, we take the entire vector of 40 candidate statistics as the input to a neural net, and the parameters that generated the statistics as the output. Several nets with one, two and three hidden layers, with different numbers of neurons per layer, were trained and tested using 500000 training observations of the statistics, and 50000 validation observations. One of the reasons why neural nets may be exceptionally well suited for simulation based estimation is that it is possible to generate very large training and validation sets. This allows a given neural net to be trained very well, and it is a simple matter to train a number of nets, to choose the one which fits best, based upon the validation data. The training was done using the `MXNet.jl` package (https://github.com/dmlc/MXNet.jl) for the Julia language . Both MXNet and Julia are freely available under the MIT license, and run on all of the popular operating systems. MXNet allows for computations to be done using GPU (graphical processing unit) computing, if a GPU that supports NVIDIA CUDA (http://www.nvidia.com/object/cuda_home_new.html) is available. The selected net, based upon best performance for the validation set, has two hidden layers of 300 and 40 neurons, respectively. As noted above, this net contains a total of 24449 parameters. The output of the training process, $\hat{\theta}(z_n)$, the fitted posterior mean, is a closed form expression, and evaluation of this expression is essentially instantaneous. An entire Monte Carlo study of 10000 replications of the estimator can be done in less than two hours, including the time to generate the training and testing sets, and the time to train the net.

The Julia script `300_40.jl` which was used to train the net is presented in Listing 6, in the Appendix. To run this, one simple enters "`julia 300_40.jl`" at the system command prompt. This will train the net, as described above, and will save the final trained parameters of the net for subsequent use. While an interested reader will need to consult the MXNet documentation to fully understand this file, it is presented here to document that it is a simple matter to define and train a net using a package such as MXnet.

Once the net is trained, it may be used to make predictions of the outputs, given the inputs. The first estimator we consider is $\hat{\theta}(z_n)$, the output of the net at the real data value of the statistic. This direct estimator was implemented using the same initial statistics and true parameter values as were used in the Monte Carlo results reported in Creel et al. [7], so the results are comparable with the results of that paper. Results for 10000 Monte Carlo replications of this estimator are given in Table 1, in the columns

labeled "NN". For this problem, the indirect inference objective function of eqn. 3 is smooth and convex, and gradient based minimization converges readily to a function value of zero, so it is feasible to study the properties of this estimator by Monte Carlo. The results for 100 Monte Carlo replications are reported in Table 1, in the columns labeled "II". We also can consider an ABC estimator which approximates the posterior mean $E(\theta|\hat{\theta}(Z_n) = \hat{\theta}(z_n))$, using the methods of Creel et al. [7]. The results, again for 100 Monte Carlo replications, are reported in Table 1, in the columns labeled "ABC". The II and ABC estimators are much more costly to compute than is $\hat{\theta}(z_n)$, which is the reason why fewer replications were done.

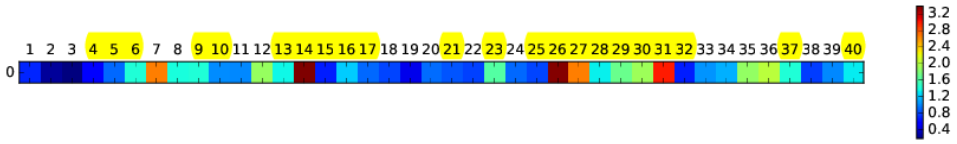
In the table, bias and root mean squared error (RMSE) are presented in percentage terms, relative to the true parameter values, to ease interpretation and comparison. We see that, overall, the estimators perform quite well. The direct output of the net estimator (NN) is somewhat more biased (in absolute terms) than are the II and ABC estimators, in most cases, but the maximum bias is still less than 4% of the true parameter value. The II and ABC estimators obtain biases which are usually less than 1%, and are always less than 4%, with the II estimator doing particularly well. Examining RMSE, the three estimators are quite similar, overall, with no clear best performer. While the NN estimator has moderately more bias, and RMSE similar to the alternative, it has the advantage that it is extremely rapid to compute, once the net has been trained. The entire 10000 Monte Carlo replications can be computed in seconds. The other two estimators, ABC and II, are much more time consuming to compute, requiring hours to do the computations for 100 replications. For the II and ABC estimators, the percentage of times that the true parameter value lies in a 90% confidence interval is reported in the last two columns of the table. We see that confidence interval coverage is quite good, for both estimators. Recall that only 100 Monte Carlo replications have been used, so some random deviation from 90% coverage is to be expected (the 0.05 and 0.95 quantiles of a binomial $n = 100$, $p = 0.9$ random variable are 85 and 95, respectively). In summary, II and ABC results are somewhat better than the direct NN results, in terms of bias, but they are similar in terms of RMSE. They are very much more time consuming to compute. Between II and ABC, the ABC method requires tuning of bandwidths, while the II estimator involves minimizing a criterion. Both operations are time consuming. For the DSGE model under consideration here, minimization of the criterion given in eqn. 3 can be done easily and reliably (though slowly) using gradient based methods, in combination with the excellent start value provided by the NN estimator, so, in this case, the II estimator may be a better choice than the ABC estimator. Inference based on asymptotic results for the II and ABC estimators is quite reliable, as seen in the results for the 90% confidence intervals.

Table 1: DSGE model. Bias and RMSE as a percentage of the true parameter value, and 90% confidence interval inclusion of true value. NN=direct estimator using output of the net. II=indirect inference estimator using the output of the net as the auxiliary statistic. ABC=approximate Bayesian computation estimated posterior mean, using the output of the net as the auxiliary statistic.

		% Bias			% RMSE			90% CI	
Param.	True	NN	II	ABC	NN	II	ABC	II	ABC
α	0.33	0.05	0.01	0.11	0.27	0.30	0.27	89	94
β	0.99	0.04	0.01	0.02	0.06	0.05	0.05	92	86
δ	0.025	1.35	0.27	0.76	3.52	3.09	3.09	87	79
γ	2.00	3.85	-1.29	0.65	5.90	4.47	4.21	92	88
ρ_z	0.90	-0.29	0.08	0.26	1.71	1.54	1.26	97	94
σ_z	0.01	-3.18	-0.13	3.09	8.92	8.70	8.27	82	91
ρ_η	0.70	-1.03	-1.71	-1.32	7.90	8.30	8.15	92	92
σ_η	0.01	0.89	1.41	3.54	11.13	11.05	9.39	84	86
$\bar{n}$	1/3	0.01	0.07	0.14	0.83	0.80	0.98	93	85

It may be of interest to compare the results given here with those of Creel et al. [7], which gives results for an ABC estimator which uses a more traditional selection procedure to reduce the dimension of the 40 candidate statistics down to 22 selected statistics. If the results of Creel et al. [7], Table 3, columns 3 and 6 are put in percentage terms, as is done in Table 1 of the present paper, one can see that the results given here are, overall, somewhat better than those of the estimator which relies on the selection procedure. Let us consider how the neural net is able to extract information as well or better than the selection procedure. Let β_{i1} be the ith row of β_1, the matrix of coefficients of the first hidden layer (see eqn. 2). This is a $G-$vector, where G is the dimension of the input statistic (40 in the present case). Let $\beta_{i1}(g)$ be the g^{th} element of this vector, for the i^{th} neuron of the first layer. For each statistic, there are 300 such elements, one for each neuron. Consider the maximum of the absolute value of this quantity, over the 300 first layer neurons, for a given $g = 1, 2, ..., G$. If the g^{th} input statistic is important, this quantity must be significantly different from zero, compared to the similar quantities for other statistics; otherwise, the input statistic does not affect any neuron in an important way, compared to the other inputs. It is possible that an input could be unimportant even if this quantity were different from zero, if the affected neurons in the first layer had little or no weight in subsequent layers, so difference of this quantity from zero in the first hidden layer is a necessary but not sufficient condition for an input to be important. This distinction is not explored here. The values of this vector are presented in Figure 2. It is immediately apparent that some statistics are more important than others. Some statistics, for example statistics 1-4, 15, 19, 32, and 38 are apparently of little help for fitting θ. Others, such as statistics 7, 14, 26, 27, 31 and 36 are clearly important. The selected statistics used in Creel et al. [7] are the ones with the numbers highlighted in Figure 2. We can see that the important statistics as identified here agree quite well with those selected in the previous work, but it appears that the previous work left out some statistics that could

Figure 2: DSGE model, identification of important statistics. For each input statistic $Z_n^{(g)}$, $g = 1, ..., 40$, the maximum over $i = 1, ..., 300$ of $\left|\beta_{i1}^{(g)}\right|$.



be relevant: for example, statistics 7, 12, 35 and 36. One of the important advantages of the proposed method is simplicity: statistics do not have to be selected - their weights are determined endogenously by the network during training. This is similar in spirit to what a continuously updating GMM estimator attempts to do (see eqn. 1), where the weighting matrix $\Omega_n^{-1}(\theta)$ adjusts the weights on each statistic in an optimal way. Unfortunately, for GMM estimators, this process often fails: numerous local minima and the numerical imprecision of estimates of $\Omega_n(\theta)$, due to a high dimensionality of the statistic can lead to a singular or near singular weight matrix. The neural network approach seems to be able to accomplish the weighting task successfully, without the need to drop statistics entirely, as occurs when a selection procedure is used. Thus, all information available in the entire vector of statistics is retained, and may be used to improve the fit to $E(\theta|Z_n)$.

4.2. Moving average model

One of the examples presented by Jiang et al. [19] is a moving average model of order 2 (MA(2)). Data is generated by the model

$$\begin{aligned} y_t &= u_t + \theta_1 u_{t-1} + \theta_2 u_{t-2} \\ \theta_1 &\in [-2, 2] \\ \theta_2 &\in [-1, 1] \\ \theta_1 \pm \theta_2 &\geq -1 \end{aligned} \tag{4}$$

where the u_t are independent and identically distributed standard normal shocks. The parameter restrictions are those that define the identifiable region. The prior they use is a uniform distribution over this region, and the sample size is $n = 100$ observations. They train a neural net which takes the sample $Y_n = \{y_1, y_2, ..., y_n\}$ as the inputs, and which has 3 hidden layers of 500, 200 and 100 neurons. The output targets are the two parameter values, θ_1 and θ_2, so the network is attempting to learn the full information posterior mean, $E(\theta|Y_n)$. The number of parameters of this network is $500 + 100 \times 500 + 200 + 500 \times 200 + 100 + 200 \times 100 + 2 + 100 \times 2 = 171002$.

Here, data is generated by exactly the same process. However, the proposal is to use a statistic Z_n to capture the information in the sample, and then use the statistic as the input to a neural net, which will be trained to approximate $E(\theta|Z_n)$. The statistic used here is the vector of estimated parameters of an autoregressive model of order 10,

fit to the sample data. Using an AR(p) model to define auxiliary statistics in order to estimate an MA(q) model has a long tradition in the indirect inference literature, beginning with Gouriéroux et al. [15]. The AR(10) model is

$$y_t = \rho_0 + \sum_{s=1}^{10} \rho_s y_{t-s} + v_t. \tag{5}$$

This auxiliary model is fit by ordinary least squares, and the statistic $Z_n = [\hat{\rho}_0, ..., \hat{\rho}_{10}]$. The training set is 9×10^5 draws of parameter-statistic pairs, and the test set is 10^5 similar draws. Thus, the test set is the same size as was used by Jiang et al. [19], but the training set is 10% smaller.

The network used here has two hidden layers of size 100 and 20. This configuration was selected as the best performer of several configurations that were explored, including nets with two and three hidden layers (again, nets reasonably similar to the one selected give qualitatively very similar results, and the provided code allows exploration). There are 11 inputs and two outputs, so the total number of parameters is $100 + 11 \times 100 + 20 + 100 \times 20 + 2 + 20 \times 2 = 3262$. The time needed to train this net using `Mocha.jl` and a GPU-enabled server is about 15 minutes.

Table 2 presents the results for mean squared error (MSE) in the test set. We can see that using a net to target the limited information posterior mean $E(\theta|Z_n = z_n)$ leads to MSEs which are less than half of those obtained by targeting the full information posterior mean $E(\theta|Y_n = y_n)$, for both parameters. This is in spite of the fact that the network that is used has less than 2% as many parameters (3262 versus 171002), and that $E(\theta|Y_n)$ has full asymptotic efficiency (Bickel and Yahav [2]), while $E(\theta|Z_n)$ does not, as it looses information. Several factors may explain this difference. First, the net that targets $E(\theta|Y_n)$ makes no use of the knowledge we have about the model, beyond the fact that it generates a sample of size 100 and has two parameters. For example, it is not informed by the fact that the data is a time series. The fact that observations are serially correlated must be learned by the net, a difficult task (LeCun et al. [22]). The net which targets $E(\theta|Z_n)$ does not have to learn this fact, as the supplied inputs already incorporate this knowledge, albeit in perhaps not the most complete form. In the context of simulation based inference, we know everything about the model except the parameter value that generated the observed sample, and use of neural nets for parameter estimation may benefit from use of this knowledge, if inputs are provided which convey this information. A second factor that is probably important is that when the inputs are the full sample, every input incorporates a raw shock of the model (the u_t in eqn. 4), which is untempered by a law of large numbers. In contrast, the statistics $\hat{\rho}_s,\ s = 0, 1, ..., 10$ which comprise Z_n are functions of averages and weighted averages of the u_t, so their variances are smaller than the variances of the y_t, and are decreasing as the sample size grows. In general, Z_n can be chosen so that it follows a law of large numbers, so that, asymptotically, the net that approximates $E(\theta|Z_n)$ is attempting to learn a nonstochastic mapping. Finally, when the input to the net is the entire sample, the size of the net must grow with the size of the sample. Results such as those of Hornik

et al. [18], regarding the ability of nets to approximate continuous mappings, require the domain of the mapping to be of fixed dimension. Thus, they apply to nets used to approximate $E(\theta|Z_n)$, but not to nets used to approximate $E(\theta|Y_n)$. In summary, while the proposed method using a statistic may not be optimal, and while further research may lead to improvements, it seems clear that this approach is more promising than the approach which uses the entire sample as the input.

Table 2: MA(2) model. Results for RMSE in test sets. Results for the $E(\theta|Y_n = y_n)$ target are from Jiang et al. [19], pg. 18.

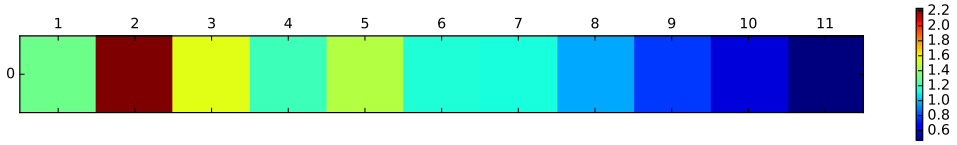
	Target of net	
Parameter	$E(\theta\|Y_n = y_n)$	$E(\theta\|Z_n = z_n)$
θ_1	0.021	0.010
θ_2	0.024	0.011

Figure 3 shows the maxima of the absolute values of the β_{i1} vectors over the 100 neurons in the first layer, similar to what was done above for the DSGE model, to identify which inputs seem to be most important. We see that the importance of $\hat{\rho}_j$ declines as j increases, almost uniformly, with $\hat{\rho}_5$ being a minor exception to this rule. This result is telling us that the lower order AR coefficients are most informative about the parameters of the MA(2) model, which makes perfect sense. The anomaly observed for $\hat{\rho}_5$ is probably due to the fact that the regressors in the AR(10) model are highly collinear. The low importance of the higher order coefficients suggests that a better statistic might result from fitting a lower order AR model. As we have already seen, the benefit of using a statistic instead of the entire sample for training a neural net is clearly demonstrated using the coefficients of the AR(10) model, so we do not further explore other possible statistics to use as the input. As has been mentioned above, the problem of determining what statistics to use as the inputs to the neural net should be informed by knowledge of the model that is being simulated. The candidate statistics used in the DSGE and MA(2) examples are illustrative of this process: knowledge of the model can lead a thoughtful researcher to propose informative candidate statistics. This is not an automatic process, and it has elements of an art. Nevertheless, the possibility of performing Monte Carlo, given that the model must be simulable, always allows one to check that a given set of candidate statistics actually leads to good identification of the model's parameters. Exactly this process has allowed us to conclude that the AR(10) model may be somewhat over-parameterized for the obtaining of best results.

5. A jump-diffusion model for S&P 500 returns

This section further illustrates the proposed methods by estimating a jump diffusion model for the Standard & Poor's 500 returns series, using data from 02 Jan. 2015 to 20 May, 2016, which gives a sample of 349 observations. Daily percentage returns for this period are plotted in Figure 4. The initial part of the sample is a sustained period of low volatility, but in August of 2015, volatility increases dramatically (at the time, there

Figure 3: MA2 model, identification of important statistics. For each input statistic $Z_n^{(g)}$, $g = 1, ..., 11$, the maximum over $i = 1, ..., 100$ of $\left|\beta_{i1}^{(g)}\right|$.



was concern about growth in China and uncertainty about when the Federal Reserve might begin to raise interest rates). Volatility remains relatively high until the final months of the sample, when it declines to low levels similar to those of the beginning of the sample.

The model, described in Creel and Kristensen [9], is a continuous time stochastic volatility model that has non-constant drift, leverage, and jumps with dynamic jump intensity. Log price, p_t, solves

$$dp_t = (\mu_0 + \mu_1(h_t - \alpha)/\sigma)\, dt + \sqrt{\exp(h_t)} dW_{1,t} + J_t dN_t. \tag{6}$$

where h_t is log volatility, J_t is jump size, and N_t is a Poisson process with time-varying jump intensity λ_t. Log-volatility is specified to follow

$$dh_t = h_t + \kappa(\alpha - h_t)dt + \sigma\left(\rho dW_{1,t} + \sqrt{1-\rho^2} dW_{2,t}\right), \tag{7}$$

where $W_{1,t}$ and $W_{2,t}$ are two independent standard Brownian motions. Jump sizes, conditional on the occurrence of a jump, are independent and conditionally normally distributed: $J_t \sim N(\mu_J, \sigma_J^2)$. Finally, the jump intensity process λ_t is modeled as

$$\lambda_t = 1(\lambda_t^* > 0) \quad \text{where } \lambda_t^* = \lambda_0 + \lambda_0\lambda_1(h_t - \alpha)/\sigma. \tag{8}$$

We collect the 10 parameters of the model in the vector $\theta = (\mu_0, \mu_1, \alpha, \kappa, \sigma, \rho, \lambda_0, \lambda_1, \mu_J, \sigma_J)$.

The data series used to compute statistics are daily returns, r, realized volatility measured using 5 minute intervals (RV), and realized bipower variation (BV), which is a measure of volatility that is robust to jumps. The difference between RV and BV can allow for the identification of jumps. The auxiliary statistics are functions of these three variables, including sample means, standard deviations, skew and kurtosis, correlations, and statistics from auxiliary regressions involving the three variables and their lags, as is detailed in Creel and Kristensen [10] and in the file `aux_stat.m` in the CTSV/Code/Common/ directory of the code archive. The total number of statistics is 56.

Data was simulated from the model using the same uniform prior with broad support as described in Creel and Kristensen [9], to create training and testing sets of size 450,000 and 100,000 observations, respectively. Because the numbers of parameters

Figure 4: S&P 500 daily returns, 02 Jan. 2015 - 20 May, 2016

and statistics are similar to the case of the DSGE model studied above, the same two hidden layer net, with 300 and 40 nodes in the first and second hidden layers, respectively, was used, without a specification search. After the net is trained, the direct NN estimates may be computed.

Table 3 reports estimation results. The direct neural net estimates $\hat{\theta}(z_n)$ are reported in the column labeled "NN". The drift parameters μ_0 and μ_1 are included to allow for trends in the price index, and not expected to be stable out of sample, and thus are not of much interest. The parameter α is the mean of log volatility. Comparing to the results of Creel and Kristensen [9], Table 3, this estimate is considerably lower than the estimate of 0.785, for the period 2008-2011, which reflects the fact that the Jan. 2015 - May 2016 period was one of comparative stability. The estimate of the mean reversion parameter, κ, is considerably higher than the estimate of 0.036 for the 2008-2011 period, which indicates that shocks to log volatility have been considerably less persistent than in the past. The estimate of σ, the standard deviation of log volatility, is considerably higher than the estimate of 0.204, for the 2008-2011 period. To understand this result, one must note that the estimated Poisson jump rate, λ_0, is slightly negative, which means that jumps never occur (see eqn. 8). If jumps never occur, which is quite plausible, considering Figure 4, then the only shocks to volatility involve the parameter σ. In contrast, the results reported in Creel and Kristensen [9] imply that jumps occurred roughly 5 times a year during the 2008-2011 period, so jumps constituted an important source of shocks to volatility. Without this source, it is not surprising that non-jump shocks to volatility are estimated to have a more important role. The estimated leverage parameter, ρ, is somewhat less extreme than in the case of the 2008-2011 results. The parameter λ_0 has already been discussed. The remaining jump parameters are not identified when when λ_0 is less than or equal to zero, because the model generates the same data regardless of the values of these parameters, and for this reason, their estimated values are meaningless, and are not reported. To summarize the direct NN estimation results, jumps have played little or no role during the sample period, volatility is lower and less persistent than during the 2008-2011 period, and the leverage effect has lessened in magnitude.

Attempting to perform minimization-based indirect inference estimation using the NN estimate as the new statistic is unsuccessful. Gradient-based methods either do not converge, or converge to a local minimum, as the point of convergence changes when different start values are used. Even a global, derivative free optimization method, simulated annealing, is unable to find a parameter value which yields a criterion value of zero, which, according to theory, must result at the true minimizing value. The computational difficulties discussed in Section 2 manifest themselves when attempting to compute the indirect inference estimator. Recall that this was not the case when estimating the DSGE model, in the previous section. The jump-diffusion model generates data with discontinuities, when jumps occur, and the auxiliary statistics used also involve indicator functions. These features result in a non-convex objective function. As such, we turn to quasi-Bayesian methods, in the form of the ABC estimator. The methods used for ABC estimation involve adaptive importance sampling, local linear

nonparametric regression, and local constant nonparametric quantile regression, in order to compute confidence intervals. Because these methods are somewhat complex, the interested reader is referred to Creel et al. [7], which describes the methods, and to the relevant portion of code archive which accompanies this paper, at `CTSV/AIS`, for full details.

Table 3, column 3 presents the ABC point estimates, which are the estimates of the posterior mean $E(\theta|\hat{\theta}(Z_n) = \hat{\theta}(z_n))$, where $\hat{\theta}(z)$ is the output of the trained net when given the input z, and z_n is the sample realization of the 56 element vector of statistics. We can see that the ABC point estimates differ little from the direct NN estimates, which perhaps supports the idea that the direct NN estimates are reliable as a point estimate. That is to say, the ABC estimates do not seem to be correcting any serious bias of the direct NN estimator. The advantage of using ABC methods is that we can compute theoretically justified confidence intervals, which have been found in previous work (see Creel et al. [7]) to have quite accurate coverage. The bounds of 90% confidence intervals are given in the fourth column of the table. We note that the two drift parameters, μ_0 and μ_1, are not significantly different from zero at the 10% level. The mean of log volatility, α, has a somewhat broad confidence interval, but the prior for this parameter is the $U(-3, 3)$ distribution, so the sample and the estimation method have together brought about a considerable reduction of uncertainty, relative to the prior. The mean reversion parameter, κ, the volatility of volatility parameter, σ, and the leverage parameter, ρ, are all estimated with quite narrow confidence interval bounds, considering that the sample size is only $n = 349$ observations.

Table 3: Estimation results and confidence interval bounds, Jump-diffusion model. S&P 500, Jan. 2015 - May 2016.

	NN	ABC	
Param.		Point est.	90% CI
μ_0	-0.055	-0.084	(-0.100, 0.012)
μ_1	0.041	0.038	(-0.002, 0.079)
α	0.268	0.285	(0.047 0.489)
κ	0.291	0.294	(0.247, 0.349)
σ	0.779	0.768	(0.686, 0.816)
ρ	-0.577	-0.593	(-0.680, -0.514)
λ_0	-0.014	-	-

6. Conclusions

This paper has modified the proposal of Jiang et al. [19] in a small but important way: instead of using the full sample as an input to a neural net, a vector of statistics is used instead. This allows specification of a much smaller net, at least in terms of the number of neurons in the first hidden layer. When a statistic is used, the size of the net is fixed, as the sample grows, a feature which does not hold when the full sample is used as the input. We have seen that neural nets which target the limited

information posterior mean can give very accurate predictions. For the DSGE example, the results for the direct estimator $\hat{\theta}(z_n)$ obtained here are comparable with those of Creel et al. [7], and can be obtained in much less time, and with much less attention to details such as selecting statistics and tuning bandwidths, a fact which is likely to improve the robustness of research conclusions. One can then go on to apply classical indirect inference methods based on $\hat{\theta}(z_n)$, as in eqn. 3, to obtain an estimator which has known asymptotic properties, and which may be used for hypothesis tests. For the MA(2) model, predictions using a net that operates on a statistic are much more accurate than those obtained by Jiang et al. [19], when the full sample is the input. The continuous time jump-diffusion example illustrates application of the methods to real data. For this sort of model, the fact that the direct neural net estimator $\hat{\theta}(z_n)$ may be computed in extremely little time, given a pre-trained net, may be of special interest, as analysis of financial data may put a premium on timeliness of results. This example presents a case where minimization-based indirect inference fails, yet approximate Bayesian computing methods perform satisfactorily. In either case, the output of the net provides an informative statistic upon which classical or Bayesian indirect inference may be based.

The size of the nets required to achieve the results are fairly modest, in comparison to nets used in deep learning applications, and the nets can be trained in little time using freely available software that accesses state of the art computational resources. The script in Listing 1 and the rest of the accompanying code archive provide examples of the methods may be implemented, and these materials can easily be adapted to other estimation problems.

Topics for future work include exploring the possibility of performing statistical inference the direct output of a trained net. The direct estimator, $\hat{\theta}(z_n)$, is obtained from a closed form expression, so it is very fast to compute. In contrast, classical indirect inference requires iterative minimization, which may fail. Bayesian indirect inference (ABC) requires some tuning decisions, and both II and ABC may be time consuming. The tuning decisions required to use the proposed neural net estimator, such as the number of neurons to use in a layer, or the number of layers, are simple questions, and may be resolved automatically through machine learning methods. It would convenient if reliable methods of inference could be developed for the direct estimator. Bootstrapping or related methods may be a promising possibility. Another direction for work might be to seek means of providing informative statistics as inputs in a systematic, more automatic way. Simulable models are specified in great detail, which allows for detailed analysis, through which an econometrician may be able to propose informative statistics. Nevertheless, this process depends in part on the econometrician. One of the goals of Jiang et al. [19] is to provide an ABC estimator with an automatically selected statistic. They do this by providing the neural net with the entire sample as the input. This removes the subjectivity associated with forming a candidate set of statistics, but it is probably too extreme a solution, because even a large neural net is not able to learn the posterior mean well enough to compensate for the less structured information that the net takes as its input. If an intermediate solution were available

that could find informative statistics in an automatic manner, perhaps using statistics derived from the characteristic function, as in Carrasco et al. [5], then the limited information posterior mean, being based on informative statistics, would be almost as accurate a point estimator as the full information posterior mean. It could be accurately learned using a moderately sized net, has has been illustrated in this paper. Finally, it would achieve these benefits in a way that eliminates or reduces the need for art or intuition on the part of the econometrician.

Acknowledgements

Financial support was given by Spanish MEC grant ECO2014-52506-R and the Severo Ochoa Programme for Centres of Excellence in R&D grant SEV-2015-0563.

Appendix

Listing 1
300_40.jl MXNet input file for the DSGE model.

```
# This script trains and saves the 300-40 MLP for the DSGE example
using MXNet
# size of layers
layer1 = 300
layer2 = 40
outputs = 9
include("dataprep.jl") # load training and testing data
# set up data providers
batchsize = 2048 # can adjust this later, but must be defined now for next line
trainprovider = mx.ArrayDataProvider(:data => X, batch_size=batchsize, shuffle=false, :label => Y)
evalprovider = mx.ArrayDataProvider(:data => XT, batch_size=batchsize, shuffle=false, :label => YT)
# set up 2 layer MLP with 2 outputs
data = mx.Variable(:data)
label = mx.Variable(:label)
net  = @mx.chain  mx.FullyConnected(data = data, num_hidden=layer1) =>
                  mx.Activation(act_type=:relu) =>
                  mx.FullyConnected(num_hidden=layer2) =>
                  mx.Activation(act_type=:relu) =>
                  mx.FullyConnected(num_hidden=outputs)
# squared error loss is appropriate for regression, don't change
cost = mx.LinearRegressionOutput(data = net, label=label)
# final model definition, don't change, except if using gpu
model = mx.FeedForward(cost, context=mx.cpu())
# set up the optimizer: select one, explore parameters, if desired
# optimizer = mx.SGD(lr=0.01, momentum=0.9, weight_decay=0.00001)
optimizer = mx.ADAM()
# train, reporting loss for training and evaluation sets
# initial training with small batch size, to get to a good neighborhood
batchsize = 256
mx.fit(model, optimizer, initializer=mx.NormalInitializer(0.0,0.05), eval_metric=mx.MSE(), trainprovider, eval_data=evalprovider, n_epoch = 100)
# more training with larger sample, saving the final fitted model
batchsize = 2048
mx.fit(model, optimizer, eval_metric=mx.MSE(), trainprovider, eval_data= evalprovider, n_epoch = 20, callbacks=[mx.do_checkpoint("dsge30040", frequency=20, save_epoch_0=false)])
```

References

[1] Beaumont, M. A., Zhang, W., Balding, D. J., 2002. Approximate Bayesian computation in population genetics. Genetics 162 (4), 2025–2035.
URL http://www.genetics.org/content/162/4/2025

[2] Bickel, P., Yahav, J. A., 1969. Some contributions to the asymptotic theory of Bayes solutions. Z. Wahrsch. Verw. Gebiete (11), 417–426.

[3] Blum, M. G. B., François, O., 2010. Non-linear regression models for Approximate Bayesian Computation. Statistics and Computing 20 (1), 63–73.
URL http://dx.doi.org/10.1007/s11222-009-9116-0

[4] Blum, M. G. B., Nunes, M. A., Prangle, D., Sisson, S. A., may 2013. A comparative review of dimension reduction methods in approximate Bayesian computation. Statistical Science 28 (2), 189–208.
URL http://dx.doi.org/10.1214/12-sts406

[5] Carrasco, M., Chernov, M., Florens, J.-P., Ghysels, E., 2007. Efficient estimation of general dynamic models with a continuum of moment conditions. Journal of Econometrics 140 (2), 529 – 573.
URL http://www.sciencedirect.com/science/article/pii/S0304407606001473

[6] Chernozhukov, V., Hong, H., aug 2003. An MCMC approach to classical estimation. Journal of Econometrics 115 (2), 293–346.
URL http://dx.doi.org/10.1016/s0304-4076(03)00100-3

[7] Creel, M., Gao, J., Hong, H., Kristensen, D., 2015. Bayesian indirect inference and the ABC of GMM, arXiv:1512.07385v1.
URL http://arxiv.org/abs/1512.07385v1

[8] Creel, M., Kristensen, D., 2011. Indirect likelihood inference, working paper, available at http://pareto.uab.es/wp/2011/87411.pdf.
URL http://pareto.uab.es/wp/2011/87411.pdf

[9] Creel, M., Kristensen, D., 2015. {ABC} of sv: Limited information likelihood inference in stochastic volatility jump-diffusion models. Journal of Empirical Finance 31, 85 – 108.
URL http://www.sciencedirect.com/science/article/pii/S0927539815000031

[10] Creel, M., Kristensen, D., 2015. On selection of statistics for approximate Bayesian computing (or the method of simulated moments). Computational Statistics & Data Analysis.
URL http://dx.doi.org/10.1016/j.csda.2015.05.005

[11] Donald, S. G., Imbens, G. W., Newey, W. K., 2009. Choosing instrumental variables in conditional moment restriction models. Journal of Econometrics 152 (1), 28–36, recent Adavances in Nonparametric and Semiparametric Econometrics: A Volume Honouring Peter M. Robinson.
URL http://www.sciencedirect.com/science/article/pii/S0304407609000566

[12] Gallant, A. R., White, H., July 1988. There exists a neural network that does not make avoidable mistakes. In: Neural Networks, 1988., IEEE International Conference on. pp. 657–664 vol.1.

[13] Gallant, R., Tauchen, G., 1996. Which moments to match? Econometric Theory 12, 363–390.
URL http://dx.doi.org/10.2139/ssrn.37760

[14] Girosi, F., Jones, M., Poggio, T., 1995. Regularization theory and neural networks architectures. Neural Computation 7, 219–269.

[15] Gouriéroux, C., Monfort, A., Renault, E., 1993. Indirect inference. Journal of Applied Econometrics, S85–S118.
URL http://dx.doi.org/10.1002/jae.3950080507

[16] Grazzini, J., Richiardi, M., 2015. Estimation of ergodic agent-based models by simulated minimum distance. Journal of Economic Dynamics and Control 51, 148 – 165.
URL http://www.sciencedirect.com/science/article/pii/S0165188914002814

[17] Haykin, S., 1998. Neural Networks: A Comprehensive Foundation, 2nd Edition. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[18] Hornik, K., Stinchcombe, M., White, H., Jul. 1989. Multilayer feedforward networks are universal approximators. Neural Netw. 2 (5), 359–366.
URL http://dx.doi.org/10.1016/0893-6080(89)90020-8

[19] Jiang, B., Wu, T., Zheng, C., Wong, W., 2015. Learning summary statistic for approximate Bayesian computation via deep neural network, arXiv:1510.02175.

[20] Kuan, C.-M., White, H., 1994. Artificial neural networks: an econometric perspective. Econometric Reviews 13 (1), 1–91.
URL http://dx.doi.org/10.1080/07474939408800273

[21] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE 86 (11), 2278–2324.
URL http://dx.doi.org/10.1109/5.726791

[22] LeCun, Y. A., Bottou, L., Orr, G. B., Müller, K.-R., 2012. Efficient BackProp. In: Lecture Notes in Computer Science. Springer Science Mathplus Business Media, pp. 9–48.
URL http://dx.doi.org/10.1007/978-3-642-35289-8_3

[23] McFadden, D., 1989. A method of simulated moments for estimation of discrete response models without numerical integration. Econometrica 57 (5), 995–1026.
URL http://www.jstor.org/stable/1913621

[24] Pakes, A., Pollard, D., 1989. Simulation and the asymptotics of optimization estimators. Econometrica 57 (5), 1027–1057.
URL http://dx.doi.org/10.2307/1913622

[25] Varian, H. R., 2014. Big data: new tricks for econometrics. Journal of Economic Perspectives 28 (2), 3–28.
URL http://www.aeaweb.org/articles.php?doi=10.1257/jep.28.2.3

[26] Winker, P., Maringer, D., 2009. The convergence of estimators based on heuristics: Theory and application to a garch model. Computational Statistics 24 (3), 533–550.
URL http://dx.doi.org/10.1007/s00180-008-0145-5