# Neural Nets for Indirect Inference

*Michael Creel*

**Universitat Autònoma de Barcelona, BGSE and MOVE***

**March, 2016**

**Abstract**

This paper shows how neural networks may be used to approximate the limited information posterior mean, $E(\theta|Z_n)$, where $\theta$ is the parameter vector of a simulable model, and $Z_n$ is a vector of statistics. 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 $E(\theta|Z_n)$ with good accuracy. The output of the net can be used as an estimator of the parameter, or, following Jiang et al. (2015), as an input to subsequent classical or Bayesian indirect inference estimation. Targeting $E(\theta|Z_n)$ using neural nets is simpler, faster, and more successful than is targeting the full information posterior mean, $E(\theta|Y_n)$, where $Y_n$ is the sample data. Code to replicate the examples and to use the methods for other models is available at https://github.com/mcreel/NeuralNetsForIndirectInference.jl. This code uses the `Mocha.jl` package for the Julia language, which allows for easy access to GPU computing, which greatly accelerates training the net.
**Keywords:** indirect inference; neural networks; approximate Bayesian Computing; machine learning

## 1 Introduction

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 (1988) and Hornik et al. (1989), who establish that feed-forward neural networks can learn a nonlinear mapping to any required degree of accuracy. A survey from the econometric point of view is Kuan and White (1994). In more recent years, computational advances, such as GPU computing, that allow for use of "deep learning" nets with many layers, 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 (**?**). 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 (2014)).

This paper proposes to use neural nets for the estimation of the parameters of a simulable model, in situations where indirect inference (Gouriéroux et al. (1993)) or Approximate Bayesian computing (Beaumont et al. (2002)) might be used. Let $\theta$ 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 $(\theta^s, z_n^s)$, $s = 1, 2, ..., S$. We can consider the regression relationship $\theta = E(\theta|Z_n) + \epsilon$, where $\epsilon$ is a mean zero error. This paper proposes to use neural networks to use 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, $\theta^s$, 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

*email: michael.creel@uab.cat. Thanks to MEC grant ECO2014-52506-R for support.

that is a large enough, in terms of number of hidden layers and neurons, to predict the outputs well. Once the net is trained, the fitted quantity, $\hat{E}(\theta|Z_n = z_n)$, which is the output of the net evaluated at the actual sample realization of the statistic, $z_n$, can serve as a point estimator of the unknown parameter.

A similar proposal has been made by Jiang et al. (2015). They propose to use 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, using a net to fit $E(\theta|Z_n)$ is both much easier to do, and performs much better. Jiang et al. (2015) point out that the output of a trained net can be used 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. If this is done, the well developed theory for such methods will apply to the final estimator. This idea applies directly to the proposal of this paper.

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[1], 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.

*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. (1998)), 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. (1998), 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 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 econometrician 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. Additionally, as was originally pointed out by Jiang et al. (2015), while the output of the net can be thought of as an estimator of the parameter, it can also be viewed as a statistic which is highly informative for the parameter, and which is of minimal dimensionality. High dimensional inputs are not a serious impediment, and the output of the net can be thought of as a dimensionality reducer, for subsequent estimation using classical or Bayesian indirect inference.

*Statistical efficiency.* Given a large enough network, the posterior mean of the parameters, conditional on the statistics, $E(\theta|Z_n)$, can be learned with arbitrarily good accuracy (Gallant and White (1988), Hornik et al. (1989)). In the full information context, the posterior mean has the same first order asymptotic distribution as does the maximum likelihood estimator (Bickel and Yahav (1969)), 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.

[1] In the applications considered here, the networks that are used can be trained in less than one hour if GPU computing is used.

*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 `Mocha.jl` (https://github.com/pluskid/Mocha.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 pointers to the input data sets, and perhaps changing the number of layers 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[2] net 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 for the estimator is lacking. As will be seen below, the final output of the net is somewhat sensitive to the point at which training is terminated. However, as has been noted, it is possible to think of the output of the net not as an estimator, but simply as a new statistic, one that is very highly informative for the parameter, and which has the same dimension as the parameter. This statistic can be used as the input to one of the versions indirect inference. In the case of classic indirect inference, the resulting indirect inference estimator will be exactly identified, so the criterion function will not require estimation of a weight matrix. The neural net estimator will serve as a very good start value for minimization of the indirect inference criterion, so optimization will be quite easy, and a full asymptotic theory will apply to the final estimator. Jiang et al. (2015) give an example of using the output of a net as a statistic for Bayesian indirect estimation.

The next section discusses indirect inference methods, both classical and Bayesian, that rely on statistics, to understand how the proposed method 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 examples, and Section 5 offers conclusions and directions for further research. Code to replicate the results of the paper, and which also can serve as a template for application of the method to other 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 $\theta_0$. 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 $\theta$. 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 (1989); Pakes and Pollard (1989)) sets $Z_n$ to sample moments; indirect inference (Gouriéroux et al. (1993)) sets it to the parameter estimate of an auxiliary model, and the efficient method of moments (EMM - Gallant and Tauchen (1996)) sets it to the score of an auxiliary model. In principle, moments can be specified to use combinations of these ideas, and I refer to the general idea as an indirect inference estimator. Some variants of Approximate Bayesian computing (ABC - Beaumont et al. (2002)) seek to work with the posterior distribution of $\theta$ given $Z_n = z_n$, and thus, this form of ABC

[2] The nets used in the examples, below, are nonlinear regression models with up to 24449 parameters. This is by no means a large number of parameters for a neural net.

estimation can be thought of as Bayesian indirect inference. A common asymptotic theory applies to the classical and Bayesian forms of indirect inference (Creel and Kristensen (2011)). Two general problems affect these methods: the curse of dimensionality, and high (and complex) computational burden.

To begin with the first, these methods which rely on a statistic in general will not achieve full asymptotic efficiency, because they use sample information as filtered through the statistic, rather than 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 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 more asymptotically efficient than are 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 in by Donald et al. (2009), who also investigate methods for selecting moments out of a candidate set. In the context of ABC, Blum et al. (2013) provide a survey of methods that have been proposed for selection of statistics. The issue of selecting statistics or reducing the dimension of the input statistic has received a great deal of attention in the literature. The method proposed here, which is adapted from the ideas Jiang et al. (2015) to work with a net that takes an initial vector of candidate statistics as the input to the net, 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 very simple and quick to implement, taking advantage of the "black box" nature of neural nets: they work, 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. This point is made by Chernozhukov and Hong (2003), 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. 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 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.

## 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 Jiang et al. (2015) and Kuan and White (1994). A very useful practical guide is given by LeCun et al. (2012).

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 ($G = 3$) to 2 outputs ($K = 2$). The inputs I1, I2, and I3 to the net are scalar real numbers, as are the outputs O1 and O2. The net has a single hidden layer, formed by 5 hidden neurons, H1,H2,...,H5, and an output layer, which gives the values of the two outputs O1 and O2.

In general, a net may have any number of hidden layers, and each layer may have any number of neurons. The value taken by a neuron is the result of an "activation function" applied to a affine function of the inputs to the neuron. Thus, the value taken by the $i^{th}$ neuron in the $j^{th}$ hidden layer is

$$H_{ij} = f_j(\alpha_{ij} + \beta'_{ij} H_{j-1}) \tag{2}$$

where $f_j()$ is the activation function of the $j^{th}$ layer, $\alpha_{ij}$ is a scalar parameter, $\beta_{ij}$ is a parameter vector. The vector of inputs to the layer is the vector $H_{j-1}$, and it is also the output of the $j - 1^{th}$ layer. The output of the $j^{th}$ hidden layer is the vector $H_j = (H_{1j}, H_{2j}, ..., H_{n_j j})$, where $n_j$ is the number of neurons in the $j^{th}$ hidden layer. The input to the first hidden layer is simply the input data, so that if we were working with the net in Figure 1, we would have $H_0 = I = (I_1, I_2, I_3)$.

The output layer, when using a net for regression, is simply a $K$ vector valued affine function of the last hidden layer's values, without applying an activation function. Thus, if there are $P$ hidden layers,

$$O_i = \alpha_{iP} + \beta'_{iP} H_P.$$

The reason that an activation function is used to compute the values of the neurons in 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 $f(x) = \tanh(x)$ (see LeCun et al. (2012) for discussion of activation functions).

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 regression of $\theta$ 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. (2012), attempting to train a net when the inputs are correlated is a much more difficult problem that is training a net with uncorrelated inputs. With time series data, the sample $Y_n$ will consist of a vector of dependent elements, so these considerations may be relevant in explaining the results obtained by Jiang et al. (2015), who use $Y_n$ 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 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 $40 \times 300$ parameters in the $\beta_{i,1}$ parameter vectors of the first layer, and there are 40 $\alpha_{i,1}$ parameters. Similarly, in the second hidden layer, there are $300 \times 40 + 40$ parameters. There are $40 \times 9 + 9$ parameters that generate the final outputs. Thus, the total number of parameters is 24449. One can see that 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"?

First, the loss function must be chosen, and a common choice is the least squares loss function, which is what we use here. Thus, 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. (2012)), 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 regions 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. 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. Given a trained net, let $\hat{E}(\theta_j|Z_n = z)$ be the output of the net for some input value $z$. We use $\hat{\theta} = \hat{E}(\theta_j|Z_n = z_n)$ as an estimator of the parameter, where $z_n$ is the realized value of the statistic in the real, not simulated, data. We can see that the outcome of this process, $\hat{\theta}$, 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. The output of the trained net will be a good predictor of the target, if the net is well chosen and well trained, but it will not be an extremum estimator, so we will not be able to apply extremum estimator theory directly, for inference about the target. As has been mentioned, Jiang et al. (2015) propose to use the output of the trained net as a statistic for ABC estimation. The same idea applies to the output of the trained net when the input is $Z_n$, rather than $Y_n$, and one could use the net's output as a statistics for either classical or Bayesian forms of indirect inference. This has the advantage that the indirect inference estimator is supplied with a highly informative statistic of the minimal dimension that maintains identifiability. A minimal dimension statistic will help to ameliorate the difficulties that one may encounter with indirect inference, which were mentioned in the previous section.

## 4 Examples

This section presents two examples which illustrate the proposed method. This first is an example of a problem of real research interest, which shows the feasibility and good performance of the method. The second is a simple example which replicates one of the designs of Jiang et al. (2015), which allows us to explore the trade-offs between working with $E(\theta|Z_n)$ or $E(\theta|Y_n)$. Code to replicate the results of this section is available at https://github.com/mcreel/NeuralNetsForIndirectInference.jl.

## 4.1 DSGE model

Creel et al. (2015) use ABC methods to estimate a small dynamic stochastic general equilibrium (DSGE) model. The model generates $n = 160$ time series observations on 5 variables: output, consumption, hours, the wage rate, and the interest rate. Estimation is based upon a vector of 22 statistics that was chosen from a set of 40 candidate statistics, using the method described in Creel and Kristensen (2015). The candidate statistics include means, standard deviations, estimated VAR(1) coefficients, estimated VAR(1) shock covariances, and several other statistics from auxiliary models. The parameter estimates reported in Tables 3 and 4 of Creel et al. (2015) were computed using an adaptive importance sampling algorithm, and nonparametric regression with tuned bandwidths. The time needed to perform the selection of statistics is several hours, and the time to do a single estimation using the adaptive importance sampling method is roughly 3 minutes, using a 32 core server. Additionally, several hours are needed to perform tuning of bandwidths. The time to perform an entire set of 2000 Monte Carlo replications of the estimator is roughly 3 days on the 32 core server.

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 of different sizes were trained and tested using 500000 training observations of the statistics, and 50000 validation observations (generating these training and test sets takes roughly one hour of time on the 32 core server). One of the reasons why neural nets may be exceptionally well suited to simulation based estimation is that it is possible to generate very large training and validation sets, and this allows a neural net to be trained very well. The training was done using the `Mocha.jl` package for the Julia language (https://github.com/pluskid/Mocha.jl)[3]. 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 `Mocha.jl` package 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. One simple and economical way of accessing a powerful GPU is to use the Amazon Elastic Compute Cloud (EC2: https://aws.amazon.com/ec2/). The net was trained using a single g2.2xlarge instance EC2, in about 40 minutes, at a cost of 70 cents. The output of the training process is a closed form expression, $\hat{E}(\theta|Z_n)$, which is the neural net's approximation to $E(\theta|Z_n)$. 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 1. To run this, one simple enters "`julia 300_40.jl`" at the system command prompt. This will train the net, as described above, and the parameters of the trained net will be contained in the most recent file in the directory `./300_40_snapshots`. While an interested reader will need to consult the `Mocha.jl` 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 `Mocha.jl`. The effect of options such as batch size, learning policy parameters, and regularization are explained in the `Mocha.jl` documentation. Some study and experimentation will allow a user to choose these options effectively, but this script should provide a useful starting point.

Once the net is trained, it may be used to make predictions of the outputs, given the inputs. This was done using the same statistics and true parameters as in Creel et al. (2015), so the results are completely comparable with the results of that paper. The only difference is that in that paper, 1000 Monte Carlo replications were done, while here, 10000 are done, because it is essentially costless to use a high number of replications, once the net is trained. We compare to the results in Table 4 of Creel et al. (2015), which is based on kernel bandwidths that were tuned in an approximately optimal way. Specifically, we compare to the results of the local linear (LL) version of the estimator that they present, as the bandwidths were tuned to minimize RMSE of that estimator. Table 1 gives the results. For the neural net, two sets of results are given, those using the final parameters after 200000 iterations of training, and those using the parameters of iteration 199000. We can see that the two

[3] Mocha and Julia are freely available under the MIT license, and run on all of the popular operating systems

methods (ABC and neural net) give quite similar results. Both have little bias, relative to the true parameter values, for all parameters. Considering RMSE, we can see that the two methods again have very similar overall performance. For all of the model's parameters, the differences in the RMSEs between the two methods, relative to the true parameter values, is fairly minor. It seems fair to say that the two methods perform more or less equally well. The two sets of results for the neural net method serve to illustrate one of its drawbacks: it is not an extremum estimator, and the final results depend upon how training is done and when it is terminated. For example, for the parameter $\alpha$, bias increases roughly threefold from training iteration 199000 to iteration 200000, which for the parameter $\gamma$, the opposite is true, it drops by a factor of 3. The differences are not important, relative to the magnitudes of the parameters, but the lack of a definitive final result is something that may disconcert an econometrician who expects a single stable point estimate as the result of an estimation process. It is also not clear how one could test hypotheses using the neural net results, given that the results vary depending on the stopping point, and that there is no asymptotic theory for the net's final output.

In summary, the ABC and neural net methods give estimates that are of very similar quality. An important difference is that the neural net results can be obtained much more quickly, and the process is considerably more simple, in that tuning decisions are easy to explore and implement using the Mocha package. Though not reported in detail here, qualitatively similar results are obtained when larger or smaller nets are used, or different learning rates, batch sizes, regularizations, etc., are used. The provided software will allow an interested reader to explore these possibilities and confirm the claim that these details are not of much importance. A second important difference is that asymptotic theory and quite accurate confidence intervals are available for the ABC estimator (see Table 4, Creel et al. (2015)), while no such results are available for the neural net method. However, the output of the net is a new statistic that has the same dimension as the parameter vector which we seek to estimate, and as we see in Table 1, it is a highly informative statistic. This supports the idea of using the output of the neural net as the input statistic to the ABC method, as proposed by Jiang et al. (2015). They showed in an example that an ABC estimator which uses the neural net output as a statistic performs almost as well as an ABC estimator that is based on a sufficient statistic, and better than alternatives that use other summary statistics. While Jiang et al. (2015) illustrate this possibility using ABC, the same possibility could be exploited for indirect inference estimation. The benefit is that the resulting estimator would enjoy the theoretical properties that have been proven for ABC or indirect inference estimators, and these estimators would be based a just identifying statistic that is highly informative.

An interesting question is *why* the neural net estimator, which takes the entire set of 40 candidate statistics as its input, is able to perform as well as the ABC estimator, which uses the 22 informative statistics which were chosen using a selection procedure? Also, how is it possible for a nonlinear regression model with 24449 parameters to perform well at all? To address this, first, note that $\beta_{ij}$ in eqn. 2 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 $\theta$. Others, such as statistics 7, 14, 26, 27, 31 and 36 are clearly important. If one wishes to compare to the selected statistics used in Creel et al. (2015), they the ones with the numbers highlighted in yellow 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 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. (2015) is a moving average model of order 2 (MA(2)). Data is generated by the model

$$\begin{aligned} y_t &= u_t + \theta_1 u_{t-1} + \theta_2 u_{t-2} \\ \theta_1 &\in [-2, 2] \\ \theta_2 &\in [-1, 1] \\ \theta_1 \pm \theta_2 &\geq -1 \end{aligned} \tag{3}$$

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, $\theta_1$ and $\theta_2$, 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[4]. The AR(10) model is

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

This auxiliary model is fit by ordinary least squares, and the statistic $Z_n = [\hat{\rho}_0, ..., \hat{\rho}_{10}]$. The training set is $9 \times 10^5$ 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. (2015), 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 targeting $E(\theta|Z_n)$ leads to MSEs which are less than half of those obtained by targeting $E(\theta|Y_n)$, for both parameters. This is in spite of the fact that the network that is used has less than 2% as many parameters, and that $E(\theta|Y_n)$ has full asymptotic efficiency (Bickel and Yahav (1969)), while $E(\theta|Z_n)$ does not, as it looses information. Several factors may explain this difference. First, the net

[4] 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. (1993).

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, and this is something that is difficult for the net to learn (LeCun et al. (2012)). 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 successful use of neural nets for parameter estimation may require that we make use of this knowledge to the degree possible, by providing inputs 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. 3), 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 Hornik et al. (1989)'s 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.

Figure 3 shows the maxima of the absolute values of the $\beta_{i1}$ 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. In general, 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, but the topic of how to choose the candidate set is outside the scope of this paper, though it is an interesting and important question.

## 5 Conclusions

This paper has modified the proposal of Jiang et al. (2015) 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 $E(\theta|Z_n)$ can give very accurate predictions. For the DSGE example, the results obtained here are comparable with those of Creel et al. (2015), 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. For the MA(2) model, the predictions are much more accurate that those obtained by Jiang et al. (2015), when the full sample is the input. The size of the nets required to achieve the results are fairly modest, and the nets can be trained in little time using freely available software that accesses state of the art computational resources. The provided script in Listing 1 gives an example of how to implement the method which can easily be adapted to other estimation problems. This script, as well as all other code to replicate the results or implement the

method is available at https://github.com/mcreel/NeuralNetsForIndirectInference.jl.

Topics for future work include following Jiang et al. (2015)'s lead by using the output of the DSGE net as an input to the Bayesian indirect inference estimator of Creel et al. (2015). A second idea is to explore the possibility of performing inference using the output of a trained net directly. The first option would use the net's output indirectly, as a means to compute another estimator for which asymptotic theory is already available, while the second option, if possible, would allow for direct for econometric inference. It would be interesting to compare the two possibilities to see which gives most reliable inference.

## References

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

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

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

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

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

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

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

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

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.

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

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

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

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

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

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

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

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

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

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

## Tables, Figures and Listings

Tab. 1: DSGE model. Monte Carlo results. ABC results are taken from Creel et al. (2015), Table 4, using the local linear (LL) results, for 1000 Monte Carlo replications. Neural Net results are for 10000 Monte Carlo replications.

| | | Bias | | | RMSE | | |
|---|---|---|---|---|---|---|---|
| Parameter | True value | ABC | NN iter 199000 | NN iter 200000 | ABC | NN iter 199000 | NN iter 200000 |
| $\alpha$ | 0.330 | 0.003 | -0.001 | 0.003 | 0.013 | 0.001 | 0.003 |
| $\beta$ | 0.990 | 0.001 | -0.001 | -0.000 | 0.003 | 0.001 | 0.001 |
| $\delta$ | 0.025 | 0.001 | -0.001 | -0.000 | 0.004 | 0.001 | 0.001 |
| $\gamma$ | 2.000 | 0.036 | -0.095 | -0.033 | 0.103 | 0.148 | 0.114 |
| $\rho_z$ | 0.900 | -0.002 | 0.003 | -0.007 | 0.010 | 0.014 | 0.013 |
| $\sigma_z$ | 0.010 | -0.001 | 0.000 | -0.000 | 0.002 | 0.001 | 0.001 |
| $\rho_\eta$ | 0.700 | -0.012 | 0.013 | -0.018 | 0.046 | 0.051 | 0.052 |
| $\sigma_\eta$ | 0.01 | 0.000 | 0.002 | -0.001 | 0.002 | 0.002 | 0.001 |
| $\bar{n}$ | 1/3 | 0.001 | 0.000 | -0.003 | 0.004 | 0.003 | 0.004 |

Tab. 2: MA(2) model. Test set results. $\hat{E}(\theta_j|Y_n)$ results are from Jiang et al. (2015), pg. 18.

| | MSE | |
|---|---|---|
| Parameter | $\hat{E}(\theta_j\|Y_n)$ | $\hat{E}(\theta_j\|Z_n)$ |
| $\theta_1$ | 0.021 | 0.010 |
| $\theta_2$ | 0.024 | 0.011 |

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

```
# set up environment
ENV["MOCHA_USE_CUDA"] = "true"
#ENV["MOCHA_USE_NATIVE_EXT"] = "true"
using Mocha,JLD
srand(12345678)
backend = DefaultBackend()
init(backend)
snapshot_dir = "300_40_snapshots"
batchsize = 1024
maxiters = 200000
# Load and pre-process the data
include("dataprep.jl")
# specify sizes of layers
Layer1Size = 300
Layer2Size = 40
# create the network
data = MemoryDataLayer(batch_size=batchsize, data=Array[X,Y])
h1 = InnerProductLayer(name="ip1",neuron=Neurons.Tanh(), output_dim=Layer1Size, tops=[:pred1
    ], bottoms=[:data])
h2 = InnerProductLayer(name="ip2",neuron=Neurons.Tanh(), output_dim=Layer2Size, tops=[:pred2
    ], bottoms=[:pred1])
output = InnerProductLayer(name="aggregator", output_dim=9, tops=[:output], bottoms=[:pred2]
    )
loss_layer = SquareLossLayer(name="loss", bottoms=[:output, :label])
common_layers = [h1,h2,output]
net = Net("dsge-train", backend, [data, common_layers, loss_layer])
# create the validation network
datatest = MemoryDataLayer(batch_size=50000, data=Array[XT,YT])
accuracy = SquareLossLayer(name="acc", bottoms=[:output, :label])
net_test = Net("dsge-test", backend, [datatest, common_layers, accuracy])
test_performance = ValidationPerformance(net_test)
# Solve
lr_policy=LRPolicy.Inv(0.01, 0.001, 0.8)
method = SGD()
params = make_solver_parameters(method, regularization_type="L1", regu_coef=0.0001,
    mom_policy=MomPolicy.Fixed(0.9), max_iter=maxiters, lr_policy=lr_policy, load_from=
    snapshot_dir)
solver = Solver(method, params)
add_coffee_break(solver, TrainingSummary(), every_n_iter=1000)
add_coffee_break(solver, test_performance, every_n_iter=1000)
add_coffee_break(solver, Snapshot(snapshot_dir), every_n_iter=1000)
solve(solver, net)
Mocha.dump_statistics(solver.coffee_lounge, get_layer_state(net, "loss"), true)
destroy(net)
destroy(net_test)
shutdown(backend)
```

Fig. 1: A simple neural net

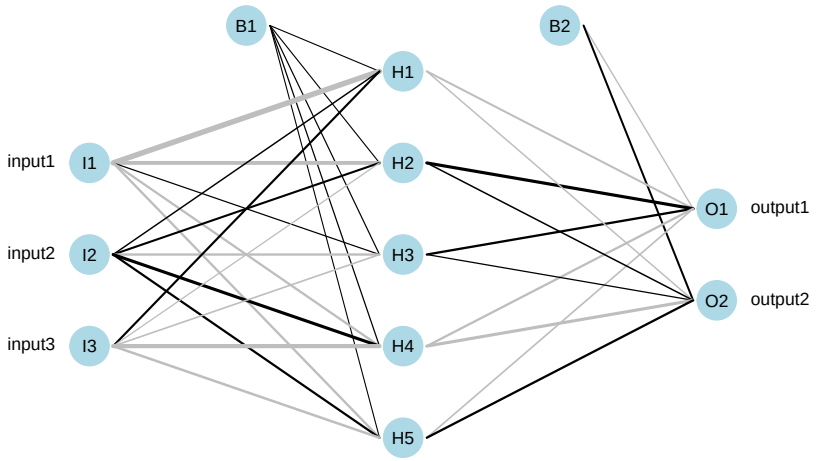


Fig. 2: DSGE model, identification of important statistics. For each input statistic $Z_n^{(g)}$, $g = 1, ...., 40$, the maximum over $i = 1, ..., 300$ of $\left|\beta_{i1}^{(g)}\right|$.

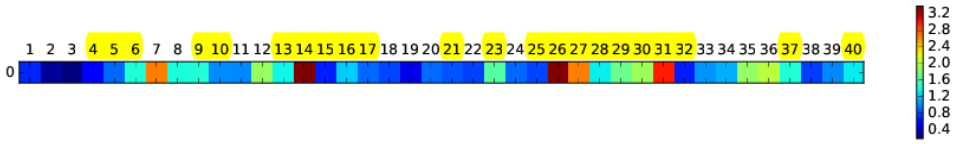


Fig. 3: MA2 model, identification of important statistics. For each input statistic $Z_n^{(g)}$, $g = 1, ...., 11$, the maximum over $i = 1, ..., 100$ of $\left|\beta_{i1}^{(g)}\right|$.