Convolutional Neural Networks for Parameter Estimation in Nicholson's Blowfly Model

Patrick Gunn

August 2024

1 Introduction

- Introduce (C)NNs and their growing popularity etc
- Introduce chaotic ecological dynamic systems (e.g. as in Wood [15]), and describe the difficulty in trying to analyse them statistically.
- Mention intractability of above problem and subsequent need for advanced, often computationally intensive methods.
- Hence suggest CNNs as a fast and accurate alternative to existing methods.
- Explain that we are using Nicholson's Blowfly model as the example case for investigation.

2 Methodology

2.1 Parameter Estimation Framework

- Describe broad outline and purpose of the inference in this paper.
- Describe how deep NNs work as an estimation technique, i.e. how they will map from inputs to targets using weights that must be optimised, etc.

2.2 Introduction to CNNs

- Introduce 1D CNNs as a class of deep NNs, describing the motivation for this choice of network.
- Explain what the convolution layers do.
- This and the above subsection could potentially be combined into one, once both are written? Depends on the level of detail desired in each subsection.

2.3 Synthetic Likelihood

Fasiolo et al. [2] provided a comparison of various existing inferential methods for noisy nonlinear ecological dynamic systems, including Synthetic Likelihood (SL). We have chosen to use this method as a benchmark because it performed well on Nicholson's original Blowfly data sets, first recorded in Nicholson [9, 10].

The first step in the method is to transform the raw observed data, $\boldsymbol{y}$, to a vector of summary statistics, $\boldsymbol{s}$. We describe which statistics we use in our analysis in Section 3.2. The construction of a synthetic likelihood is then dependent on the multivariate normality assumption

$$\boldsymbol{s} \sim \mathrm{N}(\boldsymbol{\mu}_\theta, \Sigma_\theta), \tag{1}$$

which must itself be checked before using the synthetic likelihood for any statistical inference. For any $\boldsymbol{\theta}$ we can estimate $\boldsymbol{\mu}_\theta$ and Σ_θ by simulating from the model being analysed; in particular, we use the model to

simulate N_R replicate data sets $\boldsymbol{y}_1^*, \boldsymbol{y}_2^*, \ldots$ and convert these to replicate statistics vectors $\boldsymbol{s}_1^*, \boldsymbol{s}_2^*, \ldots$ exactly as $\boldsymbol{y}$ is converted to $\boldsymbol{s}$. Then, Wood [15] showed that estimates for the unknown mean vector and covariance matrix in (1) are given by

$$\hat{\boldsymbol{\mu}}_\theta = \sum_{i=1}^{N_r} \boldsymbol{s}_i^* \; N_r, \quad \hat{\Sigma}_\theta = SS^\top/(N_r - 1), \tag{2}$$

where $S = (\boldsymbol{s}_1^* - \hat{\boldsymbol{\mu}}_\theta, \boldsymbol{s}_2^* - \hat{\boldsymbol{\mu}}_\theta, \ldots)$. Dropping irrelevant constants, the log synthetic likelihood is

$$l_s(\boldsymbol{\theta}) = -\frac{1}{2}(\boldsymbol{s} - \hat{\boldsymbol{\mu}}_\theta)^\top \hat{\Sigma}_\theta^{-1}(\boldsymbol{s} - \hat{\boldsymbol{\mu}}_\theta) - \frac{1}{2}\log|\hat{\Sigma}_\theta|. \tag{3}$$

We then explore this synthetic likelihood using a standard Metropolis-Hastings Markov chain Monte Carlo (M-H MCMC) method to find posterior mean estimates, $\hat{\boldsymbol{\theta}}$. Starting from an initial parameter guess $\boldsymbol{\theta}^{[0]}$, iterate the following for $k = 1, \ldots, N_{MC}$:

1. Propose $\boldsymbol{\theta}^* = \boldsymbol{\theta}^{[k-1]} + \boldsymbol{\delta}^{[k]}$, where $\boldsymbol{\delta}^{[k]}$ is a random vector from a symmetric distribution (see Section 3.2 for details).

2. Set $\boldsymbol{\theta}^k = \boldsymbol{\theta}^*$ with probability $\min\left\{1, \exp\left(l_s(\boldsymbol{\theta}^*) - l_s(\boldsymbol{\theta}^{[k-1]})\right)\right\}$ and set $\boldsymbol{\theta}^k = \boldsymbol{\theta}^{[k-1]}$ otherwise.

The MCMC algorithm produces a dependent sample from the approximate posterior distributions of each parameter [2], allowing us to quantify the error in each of the point estimates (means). We describe the details of this uncertainty assessment in Section 3.2.

3 Application to Nicholson's Blowfly Model

3.1 The Blowfly Model

We consider the discretisation of Nicholson's Blowfly model for chaotic population growth, as proposed in Wood [15] and used in Llewellyn et al. [8]. We denote by $S_{1:T}$ and $R_{\tau+1:T}$ the survival and birth processes of the Blowflies respectively, with $\tau \in \mathbb{Z}^+$ the lag between population count and subsequent birth count. Population count N_t is then given by $N_t = S_t$ for $t = 1, \ldots, \tau$ and $N_t = S_t + R_t$ for $t = \tau + 1, \ldots, T$. Each adult Blowfly has an independent probability of $\exp(-\delta\epsilon_t)$ of surviving each day, where $\epsilon_t \sim \Gamma(\beta_\epsilon, \beta_\epsilon)$ is an associated environmental error term with $\beta_\epsilon > 0$, so that

$$S_t \sim \text{Binom}(N_{t-1}, \exp(-\delta\epsilon_t)).$$

Egg production is an independent Poisson process for each female given by

$$R_t \sim \text{Po}\left(PN_{T-\tau-1}\exp\left(-\frac{N_{T-\tau-1}}{N_0}\right)e_t\right)$$

with environmental noise $e_t \sim \Gamma(\beta_e, \beta_e)$, $\beta_e > 0$. The observed population counts are themselves Poisson processes in order to incorporate some measurement noise, and are given by $y_t \sim \text{Po}(\phi N_t)$, with $\phi > 0$. We let $\tau = 14$, $N_0 = 400$, $T = 300$, $\epsilon_{1:T}$ and $e_{\tau+1:T}$ be known, so that the model parameters to be estimated are $\boldsymbol{\theta} = (\delta, P, \beta_\epsilon, \beta_e, \phi)$. Note that all parameters are positively valued.

In this work we will simulate test data using the values $\boldsymbol{\theta}^{\text{test}} = (0.16, 6.5, 1, 0.1, 1)$; Figure 1 shows data $y_{1:T}$ simulated using these parameters.

3.2 Synthetic Likelihood and Metropolis-Hastings Algorithm

We use the same summary statistics as Wood [15]. For a Blowfly population count, $y_{1:T}$, over $T = 300$ days, we use a total of 23 statistics, namely:

- The autocovariances to lag 11.
- The coefficients of the cubic regression of the ordered differences $y_t - y_{t-1}$ on their observed values.

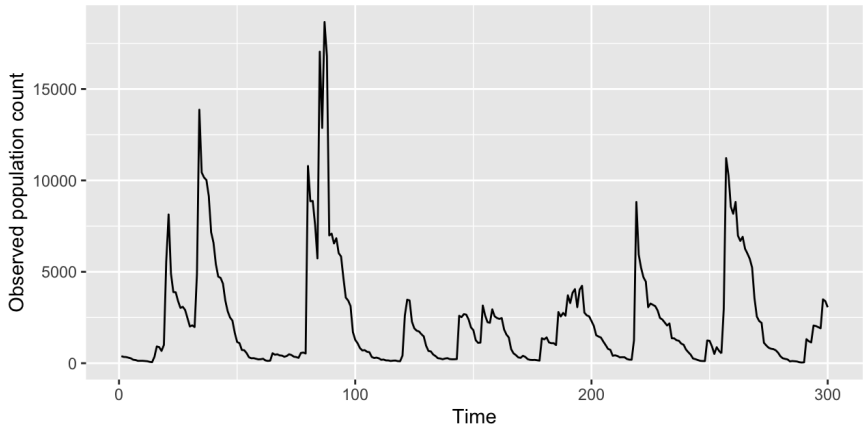


Figure 1: Simulated Blowfly population count using $\boldsymbol{\theta}^{\text{test}} = (0.16, 6.5, 1, 0.1, 1)$.

- Mean$\{y_t\}$, mean$\{y_t\}$ − median$\{y_t\}$, and the number of turning points observed.
- The estimated coefficients of the autoregression

$$y_i = \beta_0 y_{i-12} + \beta_1 y_{i-12}^2 + \beta_2 y_{i-12}^3 + \beta_3 y_{i-2} + \beta_4 y_{i-2}^2 + \epsilon_i.$$

For each proposal of the form $\boldsymbol{\theta}^* = \boldsymbol{\theta}^{[k-1]} + \boldsymbol{\delta}^{[k]}$, the $\delta_i^{[k]}$ were independent Gaussian deviates, with standard deviations equal to $(0.07, 0.1, 0.07, 0.07, 0.1)$ respectively for each parameter in $\boldsymbol{\theta}$.

Once the Metropolis-Hastings MCMC algorithm has run in full, we use a 95% highest density interval (HDI) of the approximate posteriors for each parameter to quantify the estimates provided by the method. The results produced asymmetric posterior distributions for some parameters; see Figure 5. We have therefore chosen to use HDIs over, for example, central 95% quantiles because HDIs are advantageous when dealing with skewed distributions [3].

3.3 CNN Architecture

We use a sequential 1D CNN taking Blowfly population counts over $T = 300$ days as inputs and mapping these to five scalar values as outputs, corresponding to an estimate for $\boldsymbol{\theta}$. One of the major, well-known advantages of CNNs (both 1D and 2D) is their ability to simultaneously extract important features and classify in the same network [7]. Furthermore, 1D CNNs have shown state-of-the-art results in a number of important signal processing applications [5]. Together, these motivate the choice of a 1D CNN for our estimation problem, where the network needs to be able to extract important features of the data without getting caught up in the local, noise-driven details of a population count, and where the data consists of a single variable measured over a fixed time-period.

Table 1 provides a summary of the architecture of the CNN used to estimate the Blowfly model parameters, though we exclude dropout and max pooling layers from the table. The 'output shape' column refers to the shape of the tensor which is output from that layer. For example, the first layer takes as input a tensor of shape $[-, 300]$, where '-' indicates an arbitrary number of samples and 300 corresponds to the $T = 300$ data points in one Blowfly population count, and outputs a tensor of shape $[-, 300, 64]$ where 64 represents the number of filters.

The proposed network consists of two 1D convolution layers, each with 64 filters, followed by three fully-connected dense layers with 64, 32 and five nodes respectively. Both convolution layers are immediately followed by max pooling layers, using a pool size of 2×1 on each 5×1 kernel. The second of these layers is also followed by a 20% dropout layer to prevent against overfitting of the model to the training data. The first four trainable layers of the CNN use the Leaky Rectified Linear Unit (ReLU) activation function, while the last layer uses linear activation in order to output an estimate for each parameter. The alpha value (the coefficient of the gradient for inactive units) for the Leaky ReLU activation is set to 0.1, and the CNN

Layer Type	Output Shape	Filters	Kernel Size	Parameters
1D conv	$[-, 300, 64]$	64	5×1	384
1D conv	$[-, 150, 64]$	64	5×1	20544
dense	$[-, 64]$			307264
dense	$[-, 32]$			2080
dense	$[-, 5]$			165
Total Trainable Weights:				330,437

Table 1: Summary of the 1D CNN.

Parameter	Prior Distribution
δ	$\log(\delta_i^{\text{train}}) \sim \text{Unif}(-3.7, 0)$
P	$\log(P_i^{\text{train}}) \sim \text{Unif}(0.9, 2.9)$
β_ϵ	$\log(\beta_{\epsilon_i}^{\text{train}}) \sim \text{Unif}(-1.5, 1.5)$
β_e	$\log(\beta_{e_i}^{\text{train}}) \sim \text{Unif}(-3.5, -1.1)$
ϕ	$\log(\phi_i^{\text{train}}) \sim \text{Unif}(-1.5, 1.5)$

Table 2: Prior distributions used to train the CNN on the Blowfly model.

weights are initialised randomly. Note that our model is implemented in the `R` software using Keras [4], a high-level neural networks API.

3.4 Training the CNN

We train the network on 7000 samples, simulated from combinations of different values of $\boldsymbol{\theta}$. That is, for $i = 1, \ldots, 7000$, each sample $\boldsymbol{y}_i^{\text{train}}$ is simulated using random values of $\boldsymbol{\theta} = (\delta, P, \beta_\epsilon, \beta_e, \phi)$, denoted by $\boldsymbol{\theta}_i^{\text{train}}$. We use uniform distributions to generate training values of each parameter, with the bounds for each distribution shown in Table 2. Since all parameters are positively-valued, we train the network on the log values of each parameter, hence the prior distributions are uniform and (roughly) symmetric in the log scale about the values $\boldsymbol{\theta}^{\text{test}} = (0.16, 6.5, 1, 0.1, 1)$ that we use in Section 4.

We train the network using the Adam optimizer, a popular optimizer when working with CNNs [12], with a learning rate of 0.01. It is trained for 75 epochs using a batch size of 40, meaning that, within each epoch, the CNN weights are updated after every 40th training sample has been used as input, until the CNN has run through all 7000 samples. The various CNN hyperparameters - number of epochs, learning rate, batch size - were optimised by hand until the desired results and learning accuracy were achieved. However, there exist theoretical methods to automate the optimisation of these parameters, including Bibaeva [1] and Snoek et al. [13] - these are left to be explored in future work.

3.5 Uncertainty Assessment of the CNN Estimates

In order to assess the uncertainty in the estimates provided by the CNN, we use a modified parametric bootstrapping procedure, as described by Lenzi [6]. Once an estimate $\hat{\boldsymbol{\theta}}^{\text{CNN}}$ has been obtained, the procedure works by the following:

1. Use the estimate $\hat{\boldsymbol{\theta}}^{\text{CNN}}$ to simulate $N_{\text{bs}} = 1000$ replicate data sets from the Bowfly model, denoted by $\boldsymbol{y}_i^{\text{bs}}$ for $i = 1, \ldots, 1000$.

2. Feed each $\boldsymbol{y}_i^{\text{bs}}$ into the trained CNN to obtain a bootstrapping data set $\hat{\boldsymbol{\theta}}_i^{\text{bs}}$, $i = 1, \ldots, 1000$.

3. Compute the 2.5% and 97.5% quantiles of the bootstrapping data set, resulting in the construction of a 95% bootstrap confidence interval.

This method allows us to efficiently compute the uncertainty in a CNN estimate, without having to train the CNN more than once. In the second step, prediction by the CNN is almost instantaneous (< 0.1 seconds), which allows the method to use a large bootstrapping sample size of 1000, allowing for a more robust uncertainty quantification.

4 Results

In this section, we use the values $\boldsymbol{\theta}^{\text{test}} = (0.16, 6.5, 1, 0.1, 1)$ to compare the accuracy of the CNN and SL methods against each other. Since these values are roughly at the centre (in log scale) of the simulations used to train the CNN, it is important to investigate whether the CNN has simply learned to output the mean values of its training targets each time. The plots at the end of this section demonstrate that this is not the case, and that the network is instead directly learning the mapping between Blowfly population observations, y_t, and the parameters explaining the dynamics of these observations, $\boldsymbol{\theta}$.

4.1 CNN Training and Estimates

We trained the CNN using 7000 simulated training samples, with parameter values sampled according to Table 2. The total time taken to simulate the training samples and subsequently train the CNN was *one minute 44 seconds*. Using the test parameters $\boldsymbol{\theta}^{\text{test}} = (0.16, 6.5, 1, 0.1, 1)$, we simulate a set of observations to use as test data, which we denote by $\boldsymbol{y}^{\text{test}} = \{y_1^{\text{test}}, \ldots, y_{300}^{\text{test}}\}$. Feeding $\boldsymbol{y}^{\text{test}}$ as input into the CNN, we obtained point estimates $\hat{\boldsymbol{\theta}}^{\mathbf{CNN}}$, where we recall that using the trained CNN to predict from new data sets is almost instantaneous. Following this, we used the modified bootstrapping sample of Section 3.5 to obtain 95% bootstrap intervals for each of the five parameter estimates. Figure 2 shows the estimates alongside these intervals, as well as the bounds used in the uniform priors for each parameter, and the 'true' parameter values (that is, the values $\boldsymbol{\theta}^{\text{test}}$). We observe that the CNN is able to accurately predict which parameter values generated the data $\boldsymbol{y}^{\text{test}}$, with bootstrap intervals that all contain the truth, demonstrating promising results. In Section 4.3, we compare these with the results from the M-H MCMC method.

4.2 Synthetic Likelihood Results

Figure 3 shows plots to assess the multivariate normality assumption in Equation (1). The first (left-hand-side) plot shows that it is an acceptable approximation until the higher tails of the distribution, and the dotted line in this plot shows the Mahalonobis distance for the observed statistics; this value does not indicate a significant problem with the normality assumption. The central plot again shows that the assumption fails only in the far tails, whilst the final plot does not suggest any issues with the assumption.

We used the SL method within a standard Metropolis-Hastings MCMC algorithm designed to target the posterior distributions of each parameter. The algorithm took approximately *one hour and three minutes* to run, which we note is very significantly longer than the CNN took to train and predict. Using $N_{\text{MC}} = 50000$ iterations, where in each step we used $N_R = 500$ replicates to evaluate the synthetic likelihood at the MCMC estimate for that iteration, the MCMC chains converged quickly with an acceptance rate of 32.1%. Figure 4 shows the (log scale) evolution of the chains for each parameter, as well as the evolution of the log synthetic likelihood. The algorithm performs well in estimating each parameter and converges to regions near to the truth in each case, though the width of these regions (that is, the sizes of the deviations of the chains about the truth) appear large relative to the CNN bootstrap intervals. In Section 4.3 we quantify these observations and compare them to the results of the CNN.

In Figure 5 we have plotted histograms of the MCMC chain evolutions, which represent samples from the approximate log posterior distributions of each parameter. The distributions are mostly centered on the true values of each parameter, demonstrating the relative success of the Metropolis-Hastings algorithm, with δ being the most overestimated parameter. This could suggest that the summary statistics fail to capture the influence of δ on observed data $y_{1:T}$ as much as they do for the other parameters, though it is worth noting that the estimate is still relatively accurate.

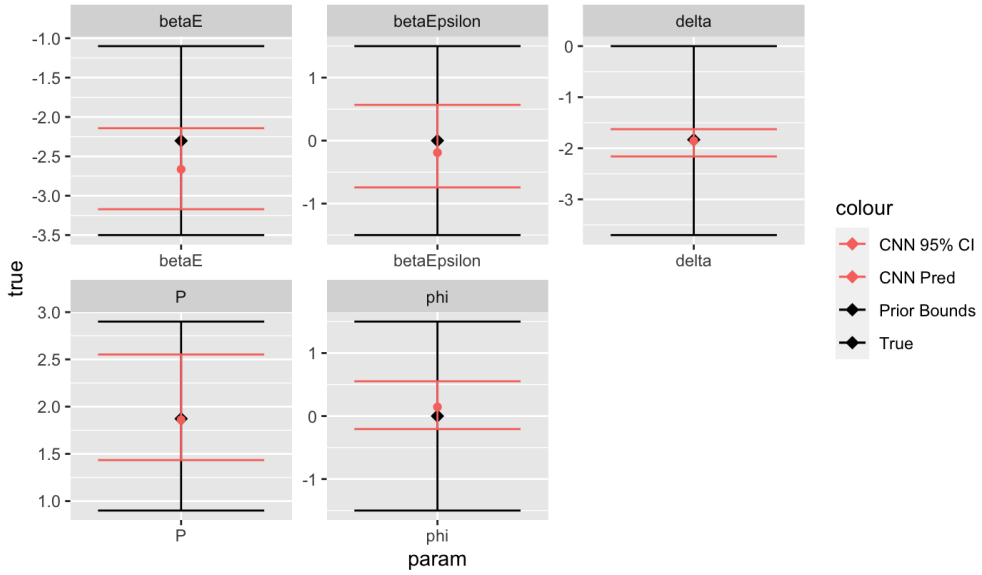


Figure 2: Point estimates and bootstrap intervals for the CNN analysis.

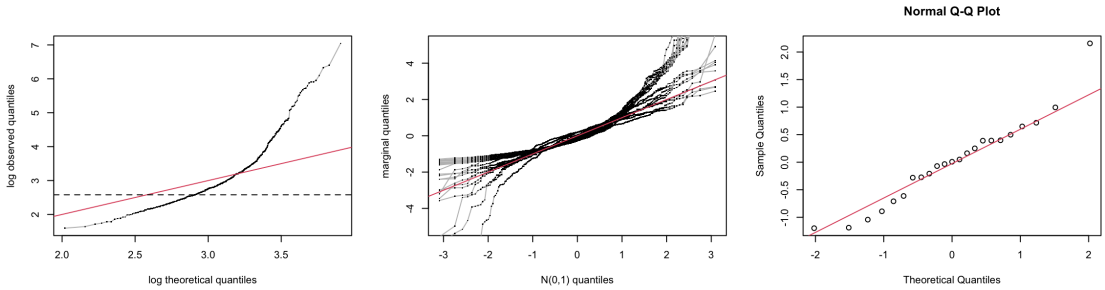


Figure 3: Plots to assess the multivariate normality assumption $\boldsymbol{s} \sim \mathrm{N}(\boldsymbol{\mu}_\theta, \Sigma_\theta)$.

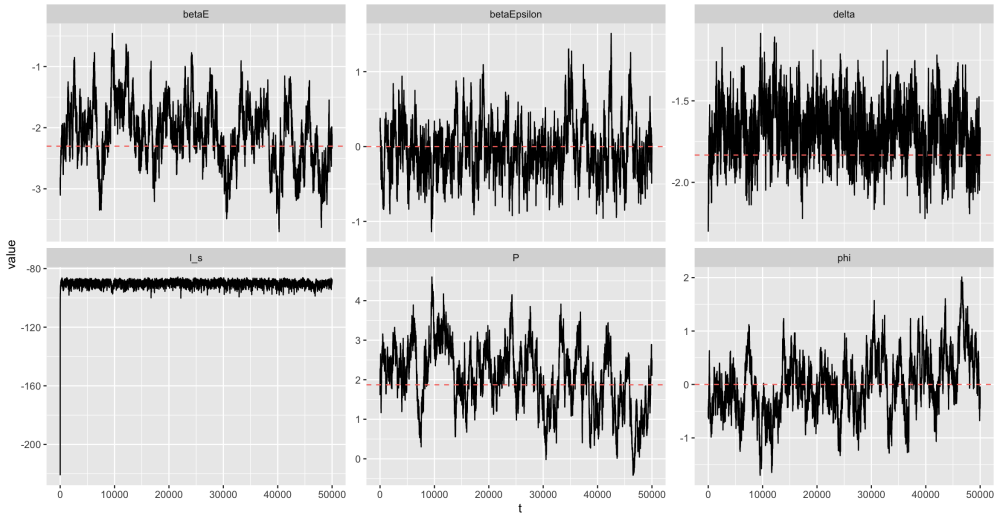


Figure 4: Evolution of the MCMC chains and log synthetic likelihood, with true parameter values shown by the dashed red lines.

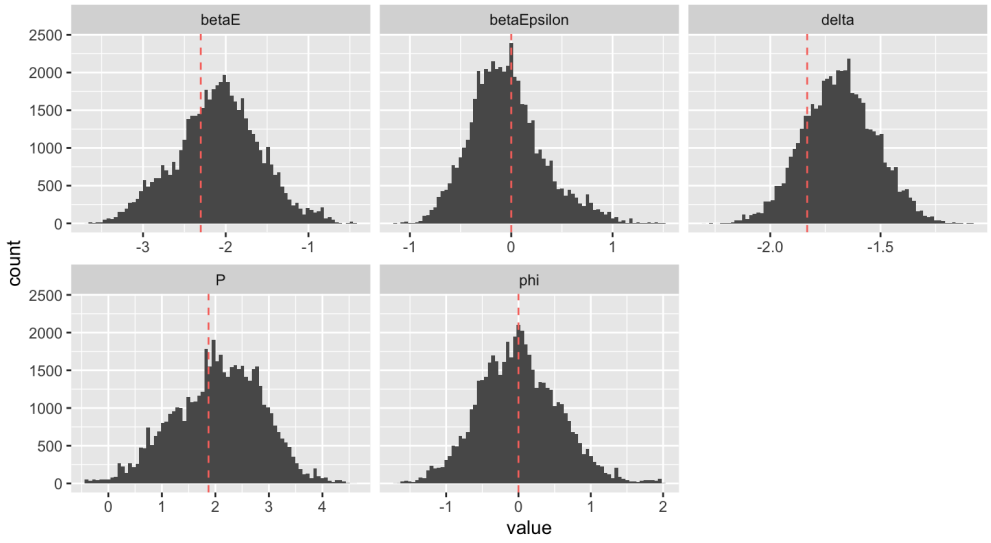


Figure 5: Histograms of the MCMC chains for each parameter, representing approximate posterior distributions. Dashed red lines denote the true values of each parameter.

Parameter	CNN Estimate	SL Estimate
δ	**0.1568**	0.1851
P	**6.433**	7.857
β_ϵ	0.8266	**0.9559**
β_e	0.06963	**0.1235**
ϕ	1.155	**1.004**

Table 3: Comparing the point estimates provided by the CNN and SL methods, all exponentiated and given to 4 significant figures. For each parameter, the bold value indicates which of the two methods provided an estimate closer to the truth, $\boldsymbol{\theta}^{\text{test}} = (0.16, 6.5, 1, 0.1, 1)$.

4.3 Comparing the CNN and SL Results

Let us denote the (exponentiated) mean values of the posterior distributions shown in Figure 5 by $\hat{\boldsymbol{\theta}}^{\mathbf{SL}}$, so that the mean values represent point estimates of $\boldsymbol{\theta}$ provided by the M-H MCMC algorithm. Table 3 compares these values with the estimates provided by the CNN, $\boldsymbol{\theta}^{\text{CNN}}$. We see that the SL method provided more accurate point estimates than the CNN for three out of the five parameters in $\boldsymbol{\theta}$, though in all cases this was by a relatively small margin. However, given that it takes under two minutes to obtain the CNN estimates, while using SL within the M-H MCMC algorithm takes more than one hour, this provides the first strong evidence that CNNs are an effective tool within this particular parameter estimation framework.

In Section 4.1, we described how the 95% bootstrap confidence intervals for the CNN estimates, $\boldsymbol{\theta}^{\text{CNN}}$, were obtained. In order to provide a useful comparison with the MCMC algorithm, we have used 95% HDIs of the approximate posteriors shown in Figure 5 to quantify the uncertainty in the estimates provided by the SL method. Figure 6 plots the two different intervals against each other, alongside the point estimates provided by both methods and the bounds of the prior uniform distributions of each parameter. Overall, using the CNN results in significantly narrower (but still accurate) intervals when compared to those from the SL method. In particular, the network is able to provide much more certain estimates for the P, β_e and ϕ parameters, demonstrating that the CNN is able to learn the relationship between the observed data and these parameters better than the M-H algorithm is able to infer.

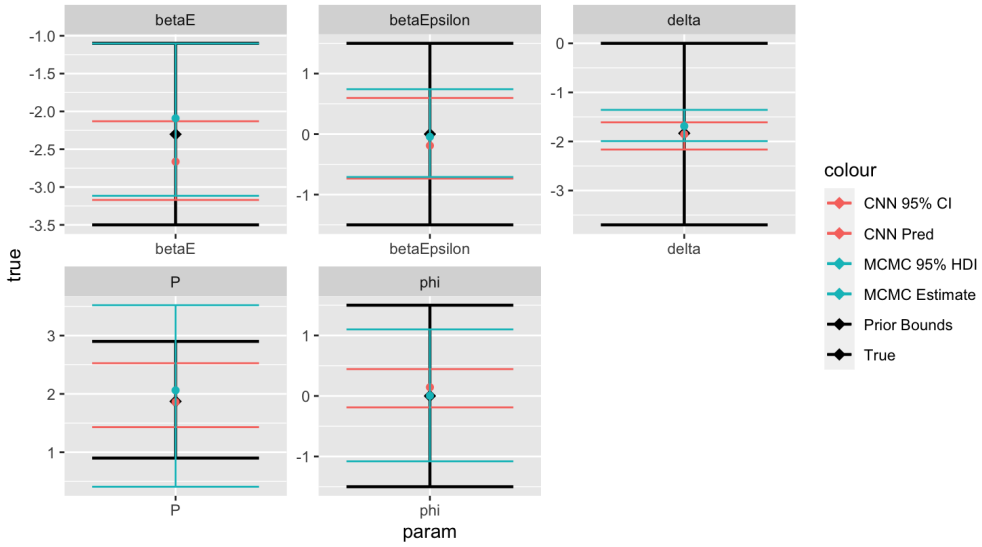


Figure 6: Point estimates and intervals of interest for both the CNN and SL methods.

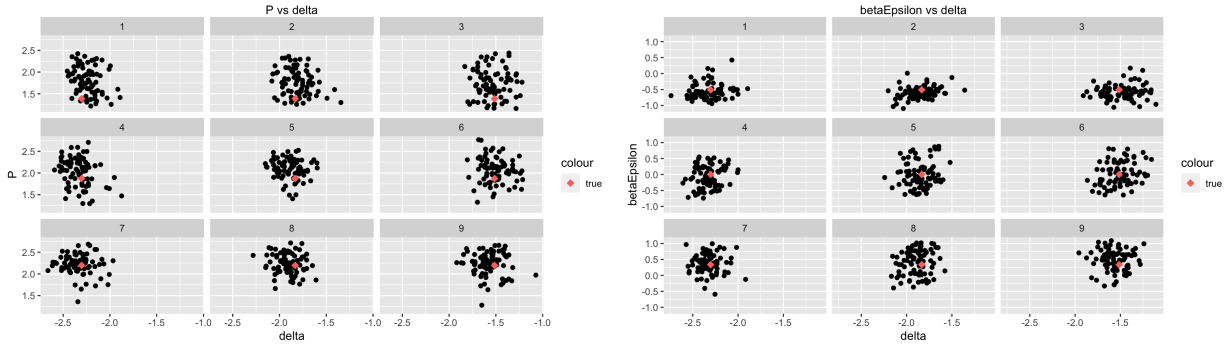
4.4 Adaptability of the CNN to Different Parameter Values

Whilst the CNN has outperformed the M-H MCMC algorithm for data generated by the values $\boldsymbol{\theta}^{\text{test}} = (0.16, 6.5, 1, 0.1, 1)$, we must also investigate whether the network can perform well for other parameter values too. Given also that the values $\boldsymbol{\theta}^{\text{test}}$ lie at the very centers of the priors used for the training target values $\boldsymbol{\theta}_i^{\text{train}}$, $i = 1, \ldots, 7000$, it becomes very important to check that the CNN can produce accurate estimates for other values within the bounds of the $\boldsymbol{\theta}_i^{\text{train}}$. Table 4 shows the various test values that we used for this purpose. For each pairwise combination of parameters (e.g. δ vs. P, δ vs. β_ϵ, etc), we used three test values of each parameter, giving a total of nine test combinations for each pair. Then, for each of these combinations, 80 samples were simulated (where the remaining three parameters had their values set to those in $\boldsymbol{\theta}^{\text{test}}$) and fed into the CNN. Each set of 3x3 plots in Figure 7 shows the estimates produced by the CNN for the various samples generated by the test values, with the 10 sets of plots shown in total corresponding to the 10 possible permutations of two parameters in $\boldsymbol{\theta}$.

The plots show that the CNN is able to accurately estimate a wide range of parameter values, not just those described in $\boldsymbol{\theta}^{\text{test}}$. They suggest that the CNN is, to a significant degree, able to directly learn the mapping from observed data, $y_{1:T}$, to the model parameters, $\boldsymbol{\theta}$, thus bypassing many of the difficulties associated with existing information-reduction or state-space methods. However, plots (d), (g), (i) and (j) indicate that the CNN is overestimating small values of ϕ, potentially suggesting overfitting of the network to estimates for this parameter. Though the CNN design already includes some basic preventative measures against this, several methods to improve them (and other techniques altogether) exist to combat overfitting; see Santos et al. [11] and Wang et al. [14] for examples. These are encouraged to be explored in future work. Despite this, the overall performance of the CNN is very promising.

Parameter	Test Values			Prior Bounds
δ	-2.30,	-1.83,	-1.50	(-3.7, 0)
P	1.39,	1.87,	2.20	(0.9, 2.9)
β_ϵ	-0.51,	0,	0.34	(-1.5, 1.5)
β_e	-2.81,	-2.30,	-1.97	(-3.5, -1.1)
ϕ	-0.51,	0,	0.34	(-1.5, 1.5)

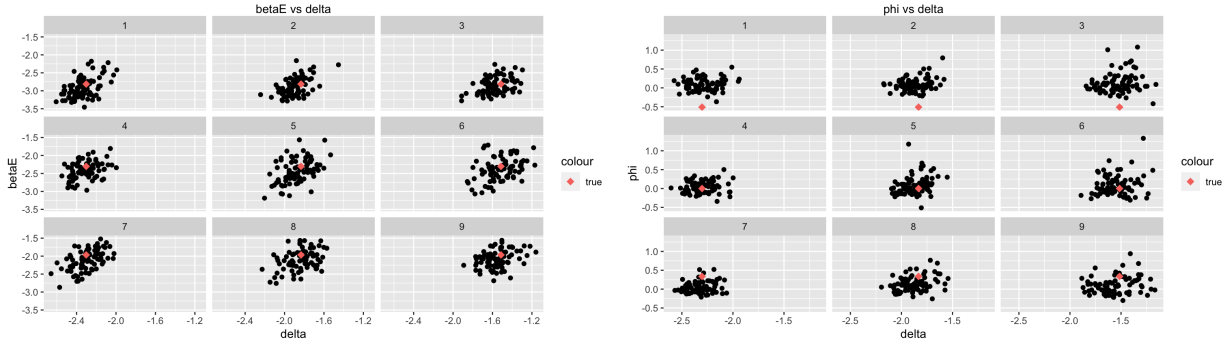
Table 4: Test values used to assess adaptability of the CNN. Notice that all values are still within the bounds used on the training data.


P vs delta
P
delta
colour
true

(a)

betaEpsilon vs delta
betaEpsilon
delta
colour
true

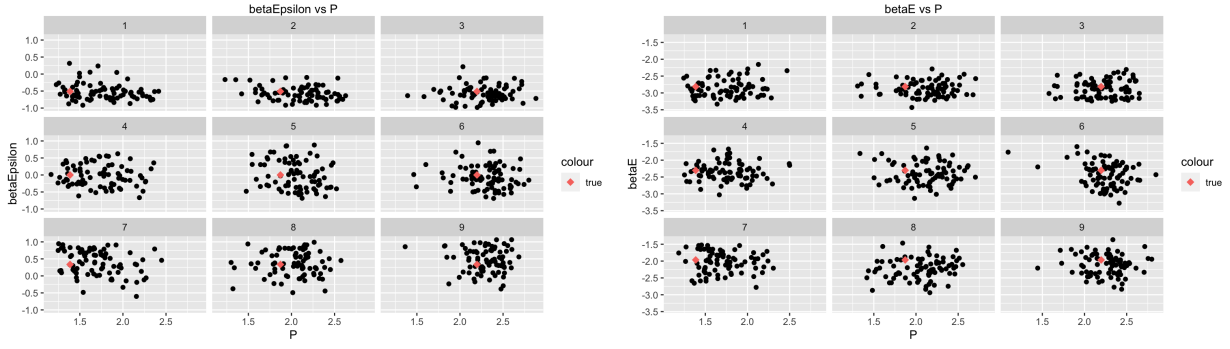
(b)


betaE vs delta
betaE
delta
colour
true

(c)

phi vs delta
phi
delta
colour
true

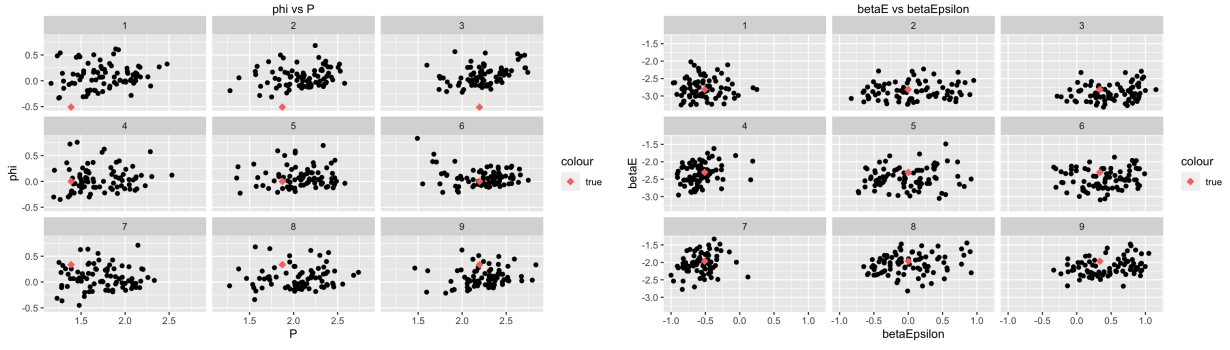
(d)


betaEpsilon vs P
betaEpsilon
P
colour
true

(e)

betaE vs P
betaE
P
colour
true

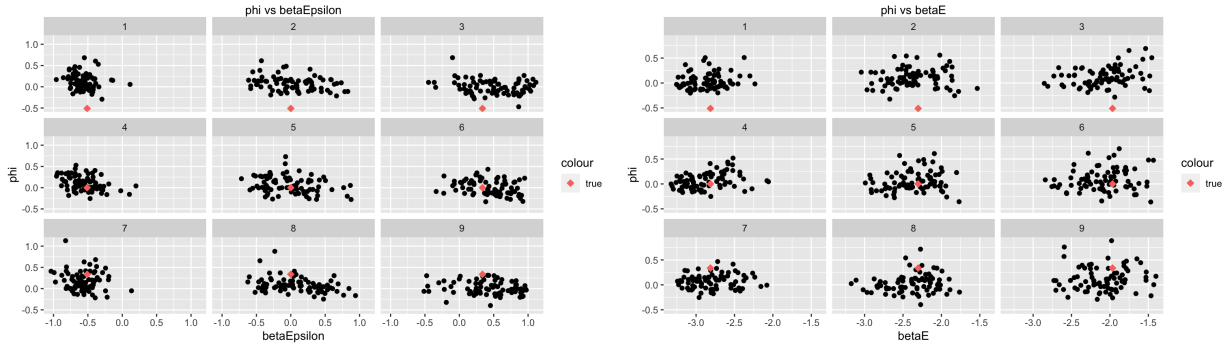
(f)


phi vs P
phi
P
colour
true

(g)

betaE vs betaEpsilon
betaE
betaEpsilon
colour
true

(h)



(i) (j)

Figure 7: Pairwise plots showing the results of the CNN's estimating a wide range of parameter values.

5 Discussion

- Summarise what has been done in this work
- Explain the benefits of using CNNs over the SL method and other methods in general; in particular, emphasise the hugely superior **time difference!**.
- Furthermore, remind that once a CNN has been trained, it can be used for near-instantaneous predictions as often as desired.
- Describe any drawbacks that the CNN has exhibited, perhaps that it is harder to quantify errors and make inferences other than point estimates? Maybe Bayesian NNs could help with this.
- Encourage exploration of other Deep NNs, as this paper only explored one CNN design.

References

[1] Victoria Bibaeva. "Using Metaheuristics for Hyper-Parameter Optimization of Convolutional Neural Networks". In: *2018 IEEE 28th International Workshop on Machine Learning for Signal Processing (MLSP)*. 2018, pp. 1–6. DOI: 10.1109/MLSP.2018.8516989.

[2] Matteo Fasiolo, Natalya Pya, and Simon N. Wood. *A comparison of inferential methods for highly non-linear state space models in ecology and epidemiology*. 2015. arXiv: 1411.4564 [stat.ME]. URL: https://arxiv.org/abs/1411.4564.

[3] Luiz Hespanhol et al. "Understanding and interpreting confidence and credible intervals around effect estimates". In: *Brazilian Journal of Physical Therapy* 23 (Dec. 2018). DOI: 10.1016/j.bjpt.2018.12.006.

[4] Keras. *Home - Keras Documentation*. Keras.io, 2019. URL: https://keras.io/.

[5] Serkan Kiranyaz et al. "1-D Convolutional Neural Networks for Signal Processing Applications". In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2019, pp. 8360–8364. DOI: 10.1109/ICASSP.2019.8682194.

[6] Amanda Lenzi et al. *Neural Networks for Parameter Estimation in Intractable Models*. 2021. arXiv: 2107.14346 [stat.ME]. URL: https://arxiv.org/abs/2107.14346.

[7] Shan Sung Liew et al. "Gender Classification: A Convolutional Neural Network Approach". In: *Turkish Journal of Electrical Engineering and Computer Sciences* 24 (Mar. 2016), pp. 1248–1264. DOI: 10.3906/elk-1311-58.

[8] Mary Llewellyn et al. "A point mass proposal method for Bayesian state-space model fitting". In: *Statistics and Computing* 33.5 (2023). ISSN: 1573-1375. DOI: 10.1007/s11222-023-10268-6. URL: http://dx.doi.org/10.1007/s11222-023-10268-6.

[9] A. J. Nicholson. "The Self-Adjustment of Populations to Change". In: *Cold Spring Harbor Symposia on Quantitative Biology* 22 (Jan. 1957), pp. 153–173. DOI: 10.1101/sqb.1957.022.01.017. (Visited on 09/15/2024).

[10] AJ Nicholson. "An outline of the dynamics of animal populations." In: *Australian Journal of Zoology* 2.1 (1954), pp. 9–65. URL: https://doi.org/10.1071/ZO9540009.

[11] Claudio Filipi Gonçalves Dos Santos and João Paulo Papa. "Avoiding Overfitting: A Survey on Regularization Methods for Convolutional Neural Networks". In: *ACM Comput. Surv.* 54.10s (Sept. 2022). ISSN: 0360-0300. DOI: 10.1145/3510413. URL: https://doi.org/10.1145/3510413.

[12] Sena Yağmur ŞEN and Nalan ÖZKURT. "Convolutional Neural Network Hyperparameter Tuning with Adam Optimizer for ECG Classification". In: *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)*. 2020, pp. 1–6. DOI: 10.1109/ASYU50717.2020.9259896.

[13] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. *Practical Bayesian Optimization of Machine Learning Algorithms*. 2012. arXiv: 1206.2944 [stat.ML]. URL: https://arxiv.org/abs/1206.2944.

[14] Yi Wang et al. "Convolutional Neural Networks With Dynamic Regularization". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.5 (2021), pp. 2299–2304. DOI: 10.1109/TNNLS.2020.2997044.

[15] Simon N. Wood. "Statistical inference for noisy nonlinear ecological dynamic systems". In: *Nature* 466.7310 (2010), pp. 1102–1104. DOI: 10.1038/nature09319. URL: https://doi.org/10.1038/nature09319.