NN Research Notes

Patrick Gunn

July 2024

1 Estimating parameters of a normal distribution, $N(\mu, \sigma^2)$.

1.1 Multi-layer Perceptron (MLP)

- Fixed one of the parameters μ, σ^2 and attempted to estimate the other using a NN comprised of several simple, fully-conected layers, called an MLP.
- Obtained 95% confidence intervals using a modified bootstrap technique: obtain a point estimate of the desired parameter, use this estimate to simulate a large number of samples, then use the MLP to obtain a new estimate for each of these samples. Then take the required quantiles from the resulting range of estimates.
- The network was able to give seemingly 'accurate' intervals for $\hat{\mu}$, but intervals obtained via maximum-likelihood methods were very significantly narrower, rendering the MLP fairly useless.
- Fixing μ and training using $\log \sigma^2$ values gave worse results, with the network unable to accurately estimate σ from large samples.

1.2 Mean-Variance Estimation (MVE) Networks

- Came across MVE networks in [3]. Given initial covariates x_i, these networks assume that the input data comes from a $N(\mu(x_i), \sigma^2(x_i))$ distribution and outputs an estimate for the mean and variance functions, optimizing the estimates by minimising the negative log-likelihood.
- The paper had associated Python code that helped me reproduce an MVE network in R. As well as using the negative log-likelihood as the loss function, these networks have a few more key features:
  - The model uses two separate sub-networks for estimating the mean and variance functions, which only share the input and output layers.
  - When training the model, an initial variance estimate (mean squared error) is obtained before the variance network is frozen, allowing the mean to be optimized first, before unfreezing and allowing both sub-networks to be (simultaneously) trained further.
  - The sub-networks can use different regularisation constants.
  - The variance sub-network is trained using log-variance values (plus 10^{-7} for numerical stability), ensuring positivity of variance predictions.
- I found that these networks were able to accurately estimate μ and σ^2 for a wide range of true values of these two parameters, both constant and non-constant. My favourite example would be:
  - I simulated 1000 uniform covariates $x_i \in [0, 10]$, then simulated 1000 observations y_i from a $N(\mu(x_i), \sigma^2(x_i))$ distribution with $\mu(x_i) = x_i(9 - x_i)$ and $\sigma(x_i) = 0.9(x_i - 2)(x_i - 4) + 1$.

- I then standardise both the x_i and y_i, converting them to standard normal distributions. I found standardisation to be crucial for producing good results.
- I trained an MVE with 9 layers in total for each of the two sub-networks, with the number of units in each layer being (40, 20, 20, 20, 10, 10, 10, 10, 1) respectively.
- Further model parameters were: learning rate = 0.001, number of epochs = 500, batch size = 100.
- Through trial and error I found that the optimum L2 regularisation constants were 0.001 and 0.01 for the mean and variance portions of the network respectively. Note that the original paper actually provides proofs of the optimum values of these constants for simpler networks.

- Figure 1.2 shows the results for this example.

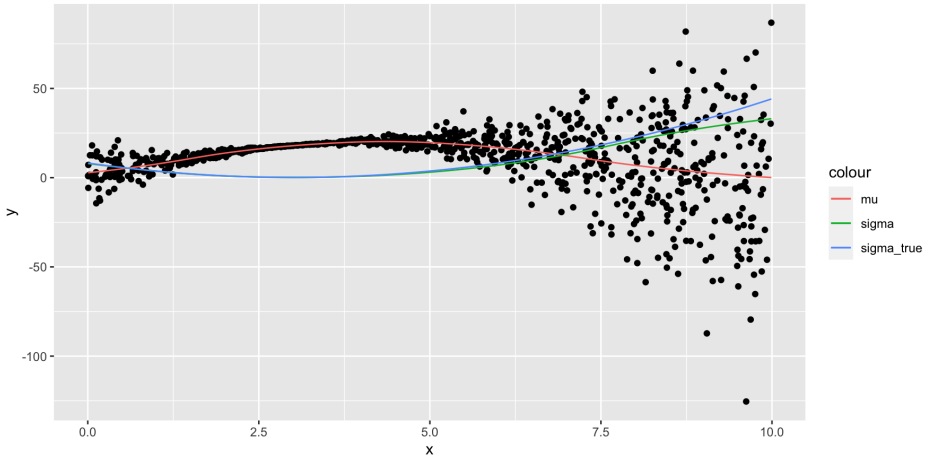

Figure 1: MVE network results

- For different parametrisations, e.g. setting $\mu(x_i)$ to be a constant value, I found that the regularisation constant needed changing each time. I also found that in some other cases a sample size of 1000 was not enough.

2 Nicholson's Blowfly Model

For now, I have been using the parametrisation for this model from Ruth King's paper [2], which has five, positive parameters to be estimated, namely $\boldsymbol{\theta} = (\delta, P, \beta_\epsilon, \beta_e, \phi)$. As a reminder, the various processes in this model are parametrised by the following:

$$\epsilon_t \sim \Gamma(\beta_\epsilon, \beta_\epsilon), \quad S_t \sim \text{Binom}(N_{t-1}, \exp(-\delta\epsilon_t)), \quad e_t \sim \Gamma(\beta_e, \beta_e),$$
$$R_t \sim \text{Po}\left(PN_{t-\tau-1}\exp\left(-\frac{N_{t-\tau-1}}{N_0}\right)e_t\right), \quad N_t = S_t + R_t, \quad y_t \sim \text{Po}(\phi N_t).$$

The goal is to design a deep neural network that, given an observed time series $(y_t)_{t\in 1:T}$, outputs estimates for all of the parameters $\boldsymbol{\theta}$. Throughout my experimentation I have used known model parameter values of $T = 300$, $\tau = 14$ and $N_0 = 400$, taken from Simon Wood's paper [4]. Also, to simplify things I have been working with only three parameters δ, P, ϕ, treating the two β parameters as known values for the time being. One of the biggest questions has been what prior distributions to assign for the three unknowns, and I don't have a clear answer yet. Note that I am using 'true' values of $\boldsymbol{\theta} = (0.16, 6.5, 1, 0.1, 1)$, and that

I have upped my training sample size from 500 to 2000 with better results since.

Since all the parameters are positively-valued, I have trained all models using $\log(\boldsymbol{\theta})$, i.e. using log values of the parameters. I also standardise all training samples $\boldsymbol{y} = \{y_1, \ldots, y_T\}$ to take values between 0 and 1 before passing them as inputs to the network. I will describe three different models I have tried below, but first I describe the metrics I use to compare each model:

- I use a **modified bootstrap technique** to produce 95% confidence intervals for each of the three parameters. This works by first simulating a blowfly sample using the true parameter values, from which we obtain a point estimate for each parameter using the trained network. These estimates are then used to simulate a large number of bootstrap samples (usually 500) from the blowfly model, and from each sample I obtain new parameter estimates by passing the sample through the network. I then take the central 95% interval of the 500 estimates to be the confidence interval.
- I also use **scatterplots of one estimated parameter against another** for different pairs of parameter values to investigate any dependence between parameter estimates from the model. I have been using three different test values for each parameter, e.g. for δ I use the test values $(0.10, 0.16, 0.22)$, which then gives nine plots for each parameter combination.

2.1 MLP

Using this basic deep NN produced promising results, but unsurprisingly other networks performed better overall so I will skip the details of this model for now. Note that since the observed data is time-dependent, it makes sense to use a CNN or other specialised network that can take time dependencies into account. Quick notes:

- Increasing regularisation constant makes the intervals **narrower** but less precise (i.e. less likely to contain true value). Best compromise found seems to be around 0.01.
- Swapping regularisation for dropout layers has a similar effect; higher dropout rate makes the intervals narrower. Best rate appears to be around 0.2.
- Regularisation has a stronger effect than dropout does, and when the two are combined it is changes to regularisation that then dominate changes in results.
- While regularisation appears to be useful in the '1D' bootstrap plots, it actually causes the '2D' plots to be very concentrated and inaccurate.

2.2 1D Convolutional Neural Network (CNN)

Copying Amanda's Model

I first used a copy of Amanda's 2D CNN from [1], with the 2D layers converted to 1D layers with the same number of units, and other minor changes. This model comprised of:

- Three 1D convolutional layers with 128, 128 and 16 filters respectively, each with kernel size of 5×1. Each of these layers uses leaky ReLU activation and is followed by a 1D max pooling layer.
- Four dense layers of size 4, 8, 16 and 3 respectively. The first three of these also use leaky ReLU activation, while the last must use linear activation.

The results for this model are shown in Figure 3, and they were fairly promising. However, in the training graph the validation loss started to tail off higher, indicating overfitting. To combat this I introduced L2 regularization into the second 1D convolution layer, and after a lot of experimenting I found the best value

of the regularisation constant to be 0.01; the results for this updated model are shown in Figure 3.

Completely counter-intuitively (?), introducing L2 regularisation would cause the models to predict much narrower intervals for parameters, which to me would indicate higher overfitting... see the bottom-right plot in Figure 3. I note that δ did not suffer from this issue as much as the two other parameters.

Designing a New Model

To combat the initial overfitting I was worried about, I reduced the complexity of Amanda's model and introduced dropout layers. After much experimentation and fine-tuning of parameters, I came up with the following architecture:

- The model design begins with two 1D convolution layers, each with 64 units. Both use leaky ReLU activation with an alpha value of 0.1, and both are followed by a 1D max pooling layer.
- The second of these convolution layers is followed by a 20% dropout layer, then flattened.
- The rest of the model consists of three dense layers of sizes 64, 32, 3. The first layer uses L2 regularization with a constant of 0.001. The first two use leaky ReLU activation, while the final one must use linear activation.
- The results for this model are shown in Figure 3.

The problem I am having is that both the custom model and Amanda's model (with regularization) have strengths and weaknesses; Amanda's model has a much stronger bootstrap CI for the ϕ parameter, but then its scatterplots show that it responds worse to changes in parameter values. On the other hand, the custom CNN has scatterplot results that respond to these changes better but with wider confidence intervals.

Prior Bounds

I note here that during my experimentation with the various 1D CNN designs I was also trying to find the best method for designating prior distributions for δ, P and ϕ. Since the models are trained on the log values of these parameters, it makes sense for the prior bounds to be uniform on the log scale. After some trial and error, I have found that the following priors work well:

$$\begin{aligned}
\log(\delta) &\sim \text{Unif}\left(2\log(0.16), 0\right) \approx \text{Unif}\left(-3.67, 0\right), \\
\log(P) &\sim \text{Unif}\left(0.5\log(6.5), 1.5\log(6.5)\right) \approx \text{Unif}\left(0.94, 2.81\right), \\
\log(\phi) &\sim \text{Unif}(-1.5, 1.5).
\end{aligned}$$

2.3 Long Short-Term Memory (LSTM) Network

3 Figures

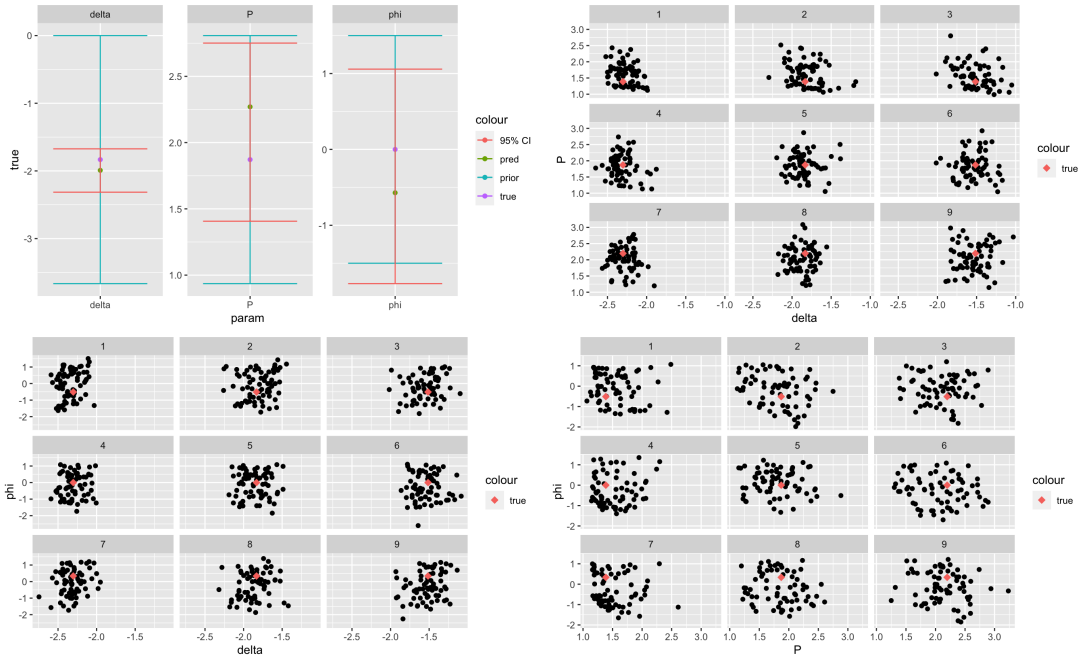

Figure 2: Initial 1D CNN results

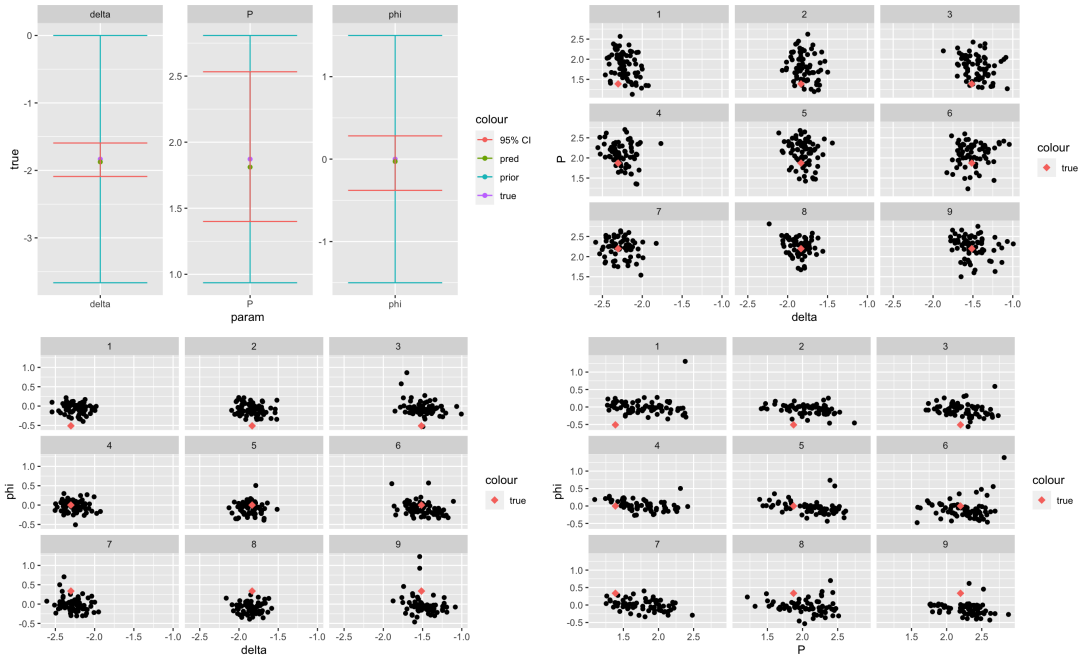

Figure 3: Results when adding L2 regularization

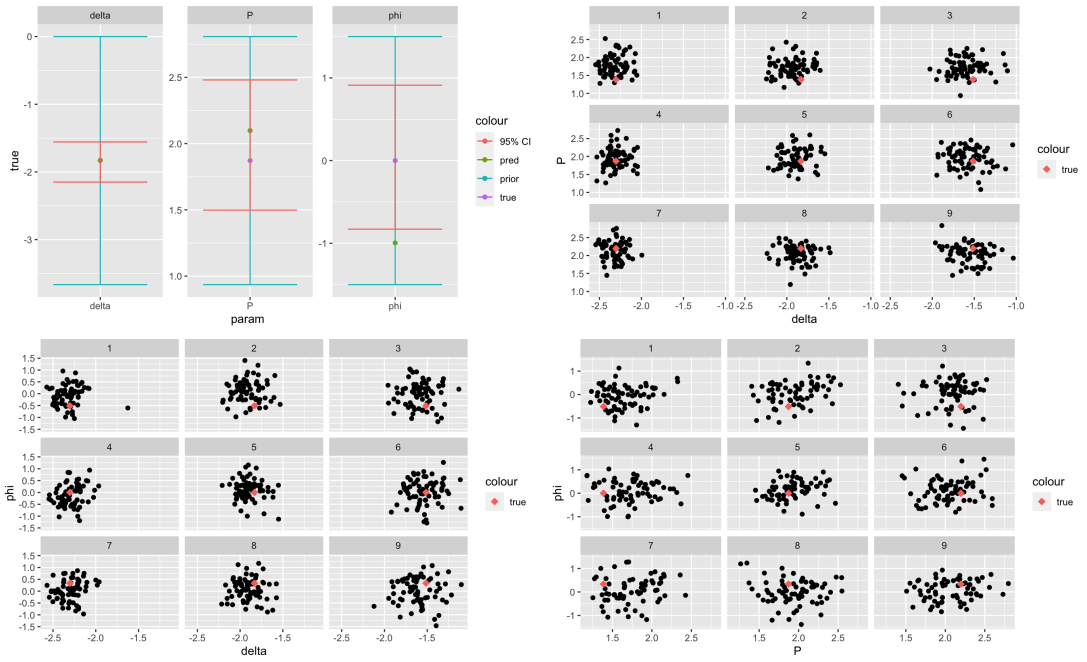

Figure 4: Custom 1D CNN results

4 To Do

- Comment all code (done for 3 param)
- Use same `y_test` for model predictions and MCMC inference.
- See if you can speed up simulating from NBF.
- Calculate credible intervals from MCMC algorithm.
- Design a network that assumes each model comes from a normal distribution, and tries to estimate the mean and variance of each prior? Use MVEs and code in email as start point.

References

[1] 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.

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

[3] Laurens Sluijterman, Eric Cator, and Tom Heskes. *Optimal Training of Mean Variance Estimation Neural Networks*. 2023. arXiv: 2302.08875 [stat.ML]. URL: https://arxiv.org/abs/2302.08875.

[4] 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.