IMPERIAL

MEng Individual Project

Imperial College London

Department of Computing

NeuralCLSNA: Applying Neural Parameter Calibration to CLSNA Opinion Dynamics

Author:
Oscar Abel Garcia Velasco N.

Supervisor:
Prof. Grigorios A. Pavliotis

Second Marker:
Prof. Colin Cotter

June 19, 2024

Abstract

Opinion dynamics is a rapidly growing sub-field of computational social science that studies the dynamics of opinion evolution and diffusion in agent networks. This research domain is inherently interdisciplinary, integrating behavioural insights from psychology and social sciences with the rigour of mathematical and physical modelling. State-of-the-art models are parameterised by abstract constructs (i.e., beliefs or influence) that cannot be theoretically derived nor physically determined. Instead, researchers utilise a wide range of calibration tools whereby model parameters are empirically fit to a relevant dataset. Common drawbacks of these techniques include high computational requirements and the absence of guaranteed convergence.

In this study, we successfully apply a neural network (NN) parameter calibration scheme to a co-evolving latent space network with attractors (CLSNA) opinion dynamics model originally proposed in 2022. This model was initially optimised with Monte Carlo Markov Chain (MCMC), which, despite providing valuable behavioural insight, was computationally expensive, with calibration converging in approximately 4 hours. An alternative calibration method using Stochastic Gradient Descent (SGD) was introduced in March 2024, boasting significantly faster convergence and comparable point/variance estimates. We demonstrate that neural calibration, a machine-learning-based method of approximating parameter probability densities, is suitable for this opinion dynamics model, reporting up to **93%** latency reduction compared to baseline and producing promising evaluation results across a range of different machine learning architectures (fully-connected, convolutional, and graph NNs). We then conduct a comprehensive sensitivity analysis using the Morris Method, Sobol Indices, and multi-objective optimisation, observing a persistent relationship between neural calibration variance estimation and parameter significance.

Further contributions are made by reformulating neural parameter calibration as a variational auto-encoder. We show that under this Bayesian framework, neural calibration is a tool to update prior beliefs on the underlying system dynamics. We finally adapt the alternative formulation to the CLSNA model as a proof-of-concept, highlighting the key benefits and limitations of the approach in our discussion.

Acknowledgements

First and foremost, I want to extend my gratitude to Professor G.A. Pavliotis for constantly encouraging me to explore areas of personal academic interest under his supervision. As my first experience in research, I enjoyed having a high level of autonomy that allowed me to produce a body of work I am truly proud of.

Thank you to Thomas Gaskin for providing insights and support over the last eight months. Often a few days after our meetings, I would have "a-ha" moments sparked by his critical thinking and probing questions.

Lastly, a heartfelt thank you to my friends and family. You have supported, motivated, and inspired me.

Contents

Chapter 1

Introduction

Opinion dynamics, an active area of research in computational social science, bridges mathematical modelling and behavioural analysis. The field aims to understand individual and collective mechanisms that govern social behaviour 'with the quantitative rigour of applied mathematics and physics' [1]. The motivation for investigating such models is self-evident: opinions are pervasive. In social networks, political discourse, or daily interaction, our opinions are constantly changing [2]. Ultimately, opinions drive action, so analysing how they form and evolve has prompted the development of many different models.

Due to their nondeterministic nature, opinions are ambiguously defined mathematical objects in literature [1], rendering opinion dynamics a highly subjective domain. This implies that the primary focus of prevalent model archetypes is to understand macro-trends in multi-agent opinion evolution (see Figure 1.1). Researchers have the onus of distinguishing meaningful patterns from noise in observed data when analysing results [3]. Such patterns include clustering activity (flocking, polarisation, etc.) and rates of opinion convergence.

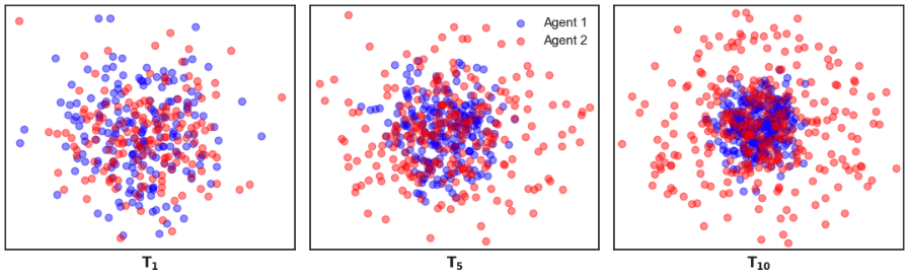


Figure 1.1: Flocking of agent class 1 in a two-dimensional latent opinion space at selected discrete time steps (T = 1, 5, 10)

A recent model, proposed by Pan et al., was designed to disentangle inter- and intra-political party behaviour between Democrats and Republicans from 2010 to 2020 [4]. This model is a coevolving latent space network with attractors (CLSNA) and is fitted using X (formerly Twitter) data. The study quantifies positive and negative polarisation in US politics. As with many computational models, the proposed CLSNA has free parameters that cannot be derived theoretically and instead are inferred from data. Originally, the authors calibrated the parameters using Monte Carlo Markov Chain (MCMC), but at the cost of significant computational demands. In an updated approach [5], the team observed comparable partisan behaviour using a faster and more efficient stochastic gradient descent (SGD) algorithm, reducing model calibration time from 1 hour to less than 5 minutes on a standard CPU.

Sophisticated calibration tools for multi-agent models have been investigated in literature. These include MCMC and SGD inference optimisation, martingale estimators [6], regression-based methods, etc. In February 2023, an artificial neural network computational scheme was proposed by Gaskin et al. [7], linking classical numerical methods and machine learning (ML). Since its intro-

duction, the neural parameter calibration approach has been applied to the diffusive Susceptible, Infectious, or Recovered (SIR) model for epidemics [7], the non-convex Harris-Wilson model of economic activity [8], and line failure inference of the British power grid [9]. There has yet to be an application to opinion dynamics models.

In this report, we look to adopt the neural calibration methodology to the state-of-the-art CLSNA dynamics model. We extend the calibration tool from a bare-bones multi-layer perception to more elaborate ML architectures specific to the CLSNA case. Furthermore, we propose a novel reparameterisation of the calibration approach as a variational auto-encoder (VAE) and demonstrate successful training regimes under this paradigm.

Due to the model-agnostic nature of neural parameter calibration, we anticipate that future research in opinion dynamics will incorporate the methodology of this work to allow for efficient and accurate parameter probability density estimation.

1.1 Contributions

This project makes the following key contributions:

1. We successfully implement the neural parameter calibration technique on the CLSNA model proposed by Pan et al., utilising various ML architectures including Multi-Layer Perceptrons (MLP), Convolutional Neural Networks (CNN), and Graph Neural Networks (GNN).[1]
2. We reinterpret Gaskin et al.'s calibration scheme within a VAE framework and demonstrate the proof-of-concept for the CLSNA. This includes the integration and validation of a Variational Graph Auto-Encoder (VGAE).
3. We use sensitivity analysis (SA) techniques to evaluate model outputs, providing an effective tool for comparing calibration techniques. In doing so, we (1) identify *significant* CLSNA parameters and (2) empirically show that probability density estimates from neural calibration can be used as a confidence-interval SA tool.

1.2 Ethical Considerations

This project has a few relevant ethical considerations: proper data handling and preventing misuse of research findings.

1.2.1 Data Handling

The data used in this project is collected from tweets from every US congressman active on the social media platform X from 2010 to 2020. The format of the collated data is time-indexed binary adjacency matrices where rows and columns are user handles. This means the congressmen are identifiable from the matrices. According to the X developer agreement [10], usage of the tweet data is justified for non-commercial research affiliated with an institution. In compliance with the General Data Protection Regulation (GDPR), the data collected for this project does not serve to profile or identify individual users. It is used exclusively to analyse the general topological properties of network structures within the dataset. We therefore adhere to the GDPR's principles of data minimisation and purpose limitation, ensuring that data processing is appropriate and limited strictly to the research's specified and legitimate purposes [11].

1.2.2 Preventing Misuse of Research Findings

While researchers in opinion dynamics often adopt a scientific "view from nowhere", analysing agents 'solely through quantitative and computational lenses' [12], it is crucial to recognise that the field is not devoid of societal implications. The influence that large corporations and government agencies wield through media channels is well-documented, and the potential for the malicious use of model simulations presents a significant ethical risk. The usage and further development of the methodologies presented must be monitored from a moral standpoint.

[1] NeuralCLSNA GitLab Repository: https://gitlab.doc.ic.ac.uk/og519/nerualclsna

Chapter 2

Background

This chapter presents academic literature on opinion dynamics models and existing parameter calibration techniques for multi-agent models. We begin by tracing the evolution of opinion dynamics models, from basic linear models to advanced non-linear formulations, concluding with the state-of-the-art CLSNA model. We then compare various parameter calibration methods, discussing their advantages and limitations as alternatives to the neural calibration method.

2.1 Evolution of Agent-Based Opinion Dynamics Models

Agent-based models (ABMs) are dynamical systems that simulate interactions of autonomous agents, thereby producing collective phenomena. They have been successful in capturing 'macro-level implications of micro-level assumptions' [13]. The ABMs for opinion dynamics have evolved significantly from their inception to the present day, introducing increasingly complex dynamics and latent parameters to simulate social behaviours more precisely [14]. These models initially focused on linear interactions but have progressed to incorporate non-linear dynamics, reflecting deeper insights into social influence and decision-making processes. In these models, opinions are represented in two ways: as discrete or continuous random variables.

Discrete representations often model opinions as n-ary outcomes, i.e. $[\mathbf{0}, \mathbf{1}, ..., \mathbf{n}]^d$, where d indicates the dimension of the opinion space. In this setting, opinions can be akin to "yes" or "no" responses in a referendum. This type of modelling is useful in scenarios where decisions are precise, mirroring situations where individuals must choose between two clear-cut options.

On the other hand, continuous representations treat opinions as variables along a continuum, denoted as $\mathbf{R}^d$. This approach is analogous to how political views lie on a spectrum without distinct boundaries. Such modelling is advantageous for complex dynamics models that require gradient calculations, such as those found in partial differential equations used to study instantaneous or iterative changes in opinions. For the remainder of this report, the focus will be primarily on continuous opinion representation.

2.1.1 Foundations of Linear Models: General Model

The most basic, tractable models are discrete-time linear models. All are based on the general model (GM), and prominent variants include the DeGroot model (DG), social network Degroot Model(s) (SNDGs), and the Friedkin and Johnson (FJ) model [15], among others.

Consider the set $\{x_i(t)\}_{i=1,...,N} \in \mathbb{R}^d$ at time t for each of the N agents. For simplicity, we focus on the case where $d = 1$, although generalisations to higher dimensions are straightforward. In these discrete-time models, time t is an element of a countable set T (i.e., $\mathbb{N}$). We define $x(t) = \{x_1(t), x_2(t), ..., x_N(t)\}$ as the opinion profile at time t. The general model assumes that 'an agent will neither share nor strictly disregard the opinion of any other agent but will take into account the opinions of others to a certain extent' [15]. To model this, there is an assignment of

different non-negative weights, $a_{ij}(t)$, representing the weight that agent i assigns to the opinions of agent j at time t. These weights are constrained to be a convex combination, allowing us to succinctly express them in the stochastic matrix $A(t, x(t)) = (a_{ij}) \in \mathbb{R}^{N \times N}$, which reflects each agent's degree of openness to other opinions in the network. It is important to note that while these weights generally depend on both time t and the current opinion profile $x(t)$, this dependency introduces non-linearity, which we will revisit in subsequent sections. Based on these definitions, the opinion formation of our agents is given by the iterative averaging equation:

$$x(t+1) = A(t, x(t))x(t) \tag{GM}$$

We will now examine variants of GM that exhibit complex long-term behaviours.

2.1.2 DeGroot, SNDG, and FJ Models

The DG is the simplest such variant. In this model, the weight matrix is constant, i.e. $A(t, x(t)) = A$, from which we see the for an arbitrary time t:

$$x(t) = A^t x(0)$$

As a result, the analysis of this model reduces to powers of the matrix A – i.e. the distribution of the opinion landscape is a Markov process. The DG model was one of the earliest proposed models, but of course, this is far too simple to describe opinion evolution, so a widely used variation of DG is known as the Friedkin-Johnson (FJ) model.

The FJ model assumes that each agent i has a predetermined "self-confidence" $\beta_i \in (0, 1)$ [16]. With these beta parameters defined, we modify each step of opinion updating in the following way:

$$x_i(t+1) = \beta_i x_i(t) + \frac{\sum_{j \neq i} a_{ij} x_j(t)}{1 - \beta_i}$$

This beta parameter measures agent free will or "novelty" in opinion formation, a notion heavily debated in philosophical literature. It is important to note that by incorporating β to the update rule, we introduce N new parameters that must be fit. A natural question arises: How are these parameters determined? As we transition to more complicated models, the number of parameters will grow and this question becomes pertinent.

Another variation of DG is the social network DeGroot model (SNDG) developed by Ding et al. [17]. This model focuses on the role that agent leadership plays in opinion dynamics. Instead of adding more parameters to the DG model, this model makes a pivotal structural assumption: that at least one agent, the "leader", has a nonzero influence on all others, formalised by the condition:

$$\exists i \in T \mid a_{ji} > 0 \forall j \neq i$$

This means at least one column of the weight matrix, excluding diagonal entries, is nonzero, ensuring that the leader's opinion significantly impacts the group's consensus process. Adding this assumption and defining this new type of agent, the "leader", it is shown to be necessary and sufficient to reach consensus among all agents [17].

The introduction of such basic yet structurally distinct variants of the general model highlights that complex models rely heavily on empirically fitted parameters, influenced by prior belief and not theoretical derivation.

2.1.3 Transition to Non-linear Models

The shift from linear to non-linear models marks a critical evolution in modelling opinion dynamics, accommodating non-linear dependencies and feedback loops inherent in real-world social processes. These models are often simulated due to their complex nature.

Bounded Confidence Model

The bounded confidence model (BCM), also known as the Hegselmann-Krause model, is a pivotal non-linear model that incorporates the idea of confidence bounds within which agents influence each other [15]. This model introduces the concept of bounded confidence through the formula:

$$x_i(t+1) = \frac{1}{|I(i, x(t))|} \sum_{j \in I(i,x(t))} x_j(t) \tag{2.1}$$

where $I(i, x(t))$ is the set of agents whose opinions are within an ϵ_i-neighbourhood of agent i's opinion at time t. The introduction of confidence levels as parameters reflects the selective exposure in opinion formation, fundamental to observing polarisation phenomena in social networks. Figure 2.1 depicts a polarisation regime under the BC model.

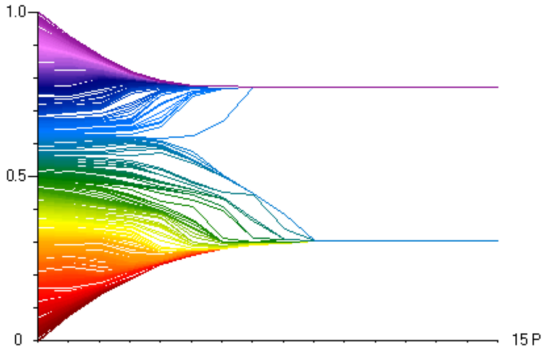


Figure 2.1: Bi-polarisation of uniformly distributed opinions in BC model. Figure from Hegselmann and Krause [15]

In summary, the developments in opinion ABMs outlined in Section 2.1 highlight an increasing emphasis on latent-variable parameterisation, like ϵ confidence neighbourhoods, to introduce complex dynamics regimes.

2.2 CLSNA: Positive and Negative Partisanship Model

The Coevolving Latent Space Network with Attractors (CLSNA) was first presented by Zhu et al. in August 2022 [4] and later refined earlier this year [5] to allow for a dynamic number of agents. The goal of the model is to disentangle positive and negative partisanship observed from congressmen's social media behaviour. Negative partisanship, originally defined in multiparty voting studies, is the negative evaluation of the opposing out-group party, as opposed to a positive assessment of one's party. The rise of negative partisanship in US politics is well documented, where approval ratings for the country's president from opposing parties have seen an average decrease surpassing 20% since the turn of the century [18]. Zhu and colleagues quantified this observed trend in their findings.

Latent variables are unobserved and typically arise in social science where concepts are mere constructs rather than directly measurable variables [19]. This makes them apt for opinion modelling where the latent variables constitute a simplified topology of the opinion landscape. It has been shown that the co-evolution of opinion and network dynamics in agent-based network models govern the appearance of realistic emerging structures [14]. The CLSNA model will be defined generally in the following subsections, but it is important to note that the original paper specifically investigated the polarisation between Republicans and Democrats from 2010 to 2020.

2.2.1 Assumptions

The positive and negative partisanship CLSNA model makes four critical assumptions:

1. The nodes exist and evolve in an abstract latent space. By definition, a latent space is an embedding of a set of items in an abstract multi-dimensional space. In this case, the latent space is $\mathbb{R}^d$.

2. Each node is classified into one of two distinct groups, with labels $\pi(i) \in \{1, 2\}$ for node i. These map bijectively with either Democrat or Republican.

3. Attractors are incorporated at the latent level to represent the attractive and repulsive forces between nodes.

4. Edges between nodes are represented by binary adjacency matrices at each time t, where $y_t \in \{0, 1\}^{NxN}$, for N the number of nodes. This is the only observed data in the model.

2.2.2 Model Definition

We now provide a detailed presentation of the dynamic node model. To ensure consistency, we adopt the notation from Zhu et al. [4] and Pan et al. [5], and we fully acknowledge their contributions to the development of the model. In what follows, capitalised letters denote random variables and lowercase variables represent realisations.

Let $G_t = (V_t, E_t)$ represent a graph network that evolves in discrete time. The number of nodes at time t is given by the cardinality of V_t. Let Y_t be the random adjacency at time t corresponding to G_t. Data comes in the form of a time series of $\{y_t : t = 1, ..., T\}$ where $y_{t,ij} = 1$ if an edge exists between node i and node j at time t and 0 otherwise. Let $z_i(t) \in \mathbb{R}^d$ be the time-indexed latent position for node i at time t. The model is defined as follows:

$$Y_{t,ij} \mid p_{t,ij} \sim \text{Bernoulli}(p_{t,ij}) \tag{2.2}$$

where at time $t = 1$,

$$\text{logit}(p_{t,ij}) = \alpha - s(z_{t,i}, z_{t,j}), \tag{2.3}$$

$$Z_{t,i} \sim \text{Normal}(0, \tau^2 I_p) \tag{2.4}$$

and at time $t > 1$, if node i is absent at time $t - 1$,

$$\text{logit}(p_{t,ij}) = \alpha + \delta Y_{t-1,ij} - s(z_{t,i}, z_{t,j}) \tag{2.5}$$

$$Z_{t,i} \sim \text{Normal}(\mu_{t,i}, \phi^2 I_p), \tag{2.6}$$

$$\mu_{t,i} = \bar{z}^{\pi(i)}_{t-1,i} \tag{2.7}$$

or else at time $t \geq 2$, if node i is present at time $t - 1$,

$$\text{logit}(p_{t,ij}) = \alpha - s(z_{t,i}, z_{t,j}) \tag{2.8}$$

$$Z_{t,i} \mid Z_{t-1,i} = z_{t-1,i} \sim \text{Normal}(\mu_{t,i}, \sigma^2 I_p) \tag{2.9}$$

$$\mu_{t,i} = z_{t-1,i} + \gamma^w_{\pi(i)} A^w_i(z_{t-1}, Y_{t-1}) + \gamma^b_{\pi(i)} A^b_i(z_{t-1}, Y_{t-1}) \tag{2.10}$$

In equations (2.3), (2.5), and (2.8), $s(\cdot, \cdot)$ represents a similarity function for p-dimensional vectors (L2 distance, for example), logit: $(0, 1) \to \mathbb{R}$ is defined as

$$\text{logit}(x) = \ln(\frac{x}{1 - x})$$

and $A^{p/n}_i$ are positive and negative attractors for node i in Y_{t-1}, defined as follows:

$$A^p_i(z_{t-1}, Y_{t-1}) = \bar{z}^1_{t-1,i} - z_{t-1,i}, \ \bar{z}^1_{t-1,i} = \frac{1}{|S^{(i)}_1|} \sum_{j \in S^{(i)}_1} z_{t-1,j} \tag{2.11}$$

$$A^n_i(z_{t-1}, Y_{t-1}) = \bar{z}^2_{t-1,i} - z_{t-1,i}, \ \bar{z}^2_{t-1,i} = \frac{1}{|S^{(i)}_2|} \sum_{j \in S^{(i)}_2} z_{t-1,j} \tag{2.12}$$

The sets $S_1^{(i)}$, $S_2^{(i)}$ represent the local neighbourhood of node i defined as follows:

$$S_1^{(i)} = \{j \neq i \mid Y_{ij} = 1, \pi(i) = \pi(j)\} \tag{2.13}$$

$$S_2^{(i)} = \{j \neq i \mid Y_{ij} = 1, \pi(i) \neq \pi(j)\} \tag{2.14}$$

The novelty of this model is the idea of a *coevolving* network. In traditional latent-space models, the latent representations update according to a Markov process, but in the positive and negative partisanship model the temporal evolution of y_{t+1} is a function of both the current opinion landscape and latent space, i.e. $z_{t+1} = f(y_t, z_t)$ as in Figure 2.2.

Note: The original **fixed node** formulation is ecovered when $V_t = V$ is held constant. Doing so makes (2.5-2.7) redundant.

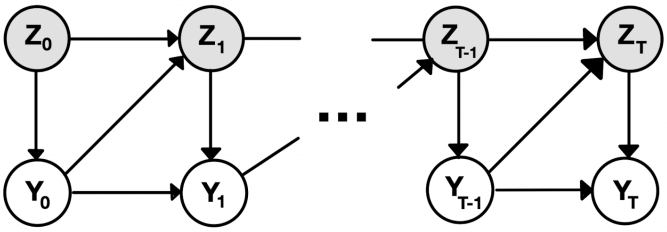


Figure 2.2: Directed graphical model of CLSNA depicting dependence of Y_{t+1} on Z_t and Y_t

2.2.3 Interpreting Model Parameters

In defining this model multiple unit-less parameters were introduced that have semantic interpretation. Namely, the dimension p for the latent opinion landscape, baseline connectivity (α), edge persistence (δ), positive-partisanship node attraction ($\gamma_{1,2}^p$), negative-partisanship node attraction (γ^n), and standard deviation measures (τ, σ, and ϕ). By simply tuning these parameters, we can observe a rich set of behaviours.

The end goal of the CLSNA model is to empirically determine both the sign and magnitude of the attraction parameters: γ_1^p, γ_2^p and γ^n. Different configurations of these parameters yield vastly different opinion evolution. For example, when both γ_1^p and γ_2^p are positive values and γ^n is negative, we observe the nodes drift in expectation towards the opinions of other nodes in their same classification and away from nodes of opposite classification, i.e. polarisation. On the other hand, if $\gamma_1^p, \gamma_2^p, \gamma^n > 0$, flocking is observed. The magnitude of these parameters represents the degree of inter-/intra- party attraction/repulsion, strongly linked with the rate of latent position convergence or divergence. Another parameter of interest is δ, corresponding to edge persistence. One may expect a priori the presence of opinion "inertia" in which connections or lack thereof in observed data $\{y_t\}$ tend to remain.

2.2.4 Metropolis Hasting MCMC

The value of Zhu et al.'s work primarily stems from the semantic interpretation of the fitted parameters, and meticulous calibration of parameters is crucial to realising the model's potential. The parameters for the model in the original paper are set by Bayesian inference using a Metropolis-Hastings (MH) within the Gibbs MCMC framework. In the first paper [4], the team utilised Bayesian inference to estimate model parameters by using the Metropolis-Hastings (MH) algorithm within a Gibbs sampling Markov Chain Monte Carlo (MCMC) framework.

At its core, the MH algorithm constructs a Markov chain that has the target distribution as its equilibrium distribution, thus ensuring that as the number of iterations increases, the distribution of the chain's states converges to the target distribution [20]. One of the primary challenges in implementing the MH algorithm is the selection of an appropriate proposal distribution, but Zhu et al. derived an identifiable posterior distribution in the appendix of [4]. This required a prior on the

parameters λ, and were chosen to be $\alpha, \delta \sim N(0, 100), \gamma_1^w, \gamma_2^w \sim N(0.5, 100), \gamma^b \sim N(-0.5, 100)$, where $N()$ is a normal distribution. The ability of this calibration to converge with high prior variance is a major benefit of this approach as it allows for efficient traversal of the parameter space. However, a significant drawback is the considerable computational demand limiting its scalability.

2.2.5 SGD Variational Inference

In March of 2024, Pan et al. presented a novel stochastic Gradient Descent (SGD) based variational inference method for the CLSNA model. This method significantly enhanced the scalability and computational speed of the preceding MCMC. The approach utilises a two-stage algorithm that **first** employs SGD to compute initial point estimates for model parameters. **Then** it refines the estimates by calculating marginal posterior standard deviations using quadratic approximations of the log-posterior density.

The main advantage of this method lies in its ability to transform variance estimation into an optimisation problem of the posterior log-likelihood. Concretely, to calculate point estimates, the algorithm starts by initialising parameter set λ, then progresses by randomly selecting indices of simpler summands from the log-posterior distribution and updates the parameters in an SGD step. This approach trades a small amount of precision in the posterior calculation for significantly reduced computation. Variance inference is similarly computed with SGD by employing an approximate Bayesian inference method grounded in Laplace's approximation, which is further detailed in [4].

2.3 Neural Parameter Calibration

Gaskin et al.'s neural parameter calibration technique presents a novel approach to recovering probability densities from noisy time-series data as summarised in Figure 2.3. Conceptually, the method trains a neural network to determine a set of parameters $\hat{\lambda}$ that best reproduces observed time series data $T = (\phi_1, \ldots, \phi_L)$. The loss values obtained at different training epochs e_i, J_{e_i}, are then used to define joint and marginal empirical parameter density functions.

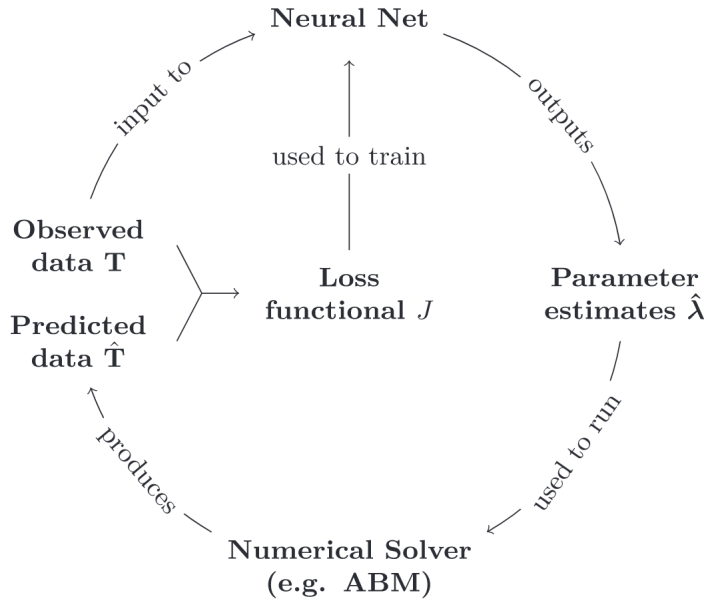


Figure 2.3: Neural parameter calibration pipeline. Diagram by Gaskin et al. [7].

2.3.1 Methodology

Neural parameter calibration starts with defining a neural network as the function $\mu_\theta : \mathbb{R}^{N \times q} \to \mathbb{R}^p$ where θ are the learned NN parameters, N is the number of observed input samples, $q \geq 1$ is the

number of input time-steps, and p the number of parameters calibrated. The output of this neural network are the estimated parameters $\hat{\lambda} = \mu_{\hat{\theta}}(X)$. At each training epoch, the output gets passed into the numerical ABM solver parameterised by λ to produce an estimated time-series output $\hat{T}(\hat{\lambda}) = (\hat{\phi}_k, \ldots, \hat{\phi}_{k+B})$. These estimations serve as a basis for training the neural network's internal parameters Θ, which include learnable weights and biases. This is accomplished through a user-defined loss objective function $J(\hat{T}, T)$ to guide the optimisation of Θ.

Calculating the gradient of the loss function $\nabla_\Theta J$ is a chain rule operation, first differentiating the predicted time series $\hat{T}$ and subsequently, the system equations concerning λ. This differentiation is integral as it incorporates the dynamics of the model into the training regime. The optimisation of Θ employs back-propagation and may utilise various optimisation algorithms (SGD, Adam, etc.).

Once the ground-truth data are re-inputted into the model, a new set of parameter estimates $\hat{\lambda}$ is generated, beginning the iterative process again. Given the complexity of general numerical solvers, the differentiation details are managed by the autodifferentiation engines of ML libraries. The stochastic nature of the CLSNA simulation is handled by treating random vectors as constants in gradient calculation.

The optimisation process is further extended to inferential statistics. While this optimisation does not yield confidence intervals directly, it facilitates the generation of probability densities through the computation of loss values across the parameter space $\mathbb{R}^p$. Specifically, each loss estimates $\hat{J}$ contributes to constructing an approximation of the observed data distribution. By Bayes formula we know the parameter posterior is proportional to the prior multiplied by the distribution of the observed data, expressed as:

$$\pi(\hat{\lambda}|T) \approx \exp(-J(\hat{T}, T))\pi^0(\hat{\lambda}), \tag{2.15}$$

where $\pi^0(\hat{\lambda})$ denotes the prior distribution. The marginal densities of the parameters are subsequently proportional to the integral of the exponential loss, given by:

$$\rho(\lambda_i) \sim \int \exp(-J) d\lambda_{-i}, \tag{2.16}$$

with λ_{-i} indicating integration over all parameters except λ_i.

The quality of the density estimation approximation improves as the dimension of the parameter space increases and the coverage of the parameter space increases. This framework benefits from the ability to aggregate stored loss values across numerous random parameter initialisations followed by their associated training phases. Applying parallel processing to this method significantly bolsters the robustness and precision of the confidence intervals obtained from the density estimates.

The calibration framework serves a dual purpose: optimising model parameters and providing a statistical framework for inference; the empirical parameter densities generate meaningful confidence intervals that reflect the uncertainty inherent in real-world data modelling.

2.4 Benefits of the Neural Calibration Approach

The neural parameter calibration scheme boasts a series of benefits. Perhaps the most significant is its model of agnosticism. This feature ensures the method is adaptable to stochastic and non-stochastic multi-agent models. Users only need to define a loss functional particular to the numerical solver, embedding specific calibration objectives in the process. Such flexibility allows for broader application to scientific and engineering fields.

Additionally, the neural network component can be easily replaced with more sophisticated deep-learning architectures tailored to specific cases. This adaptability enhances the model's performance and ensures that it keeps up with advancements in ML technologies. Unlike conventional ML methods that frequently encounter issues with generalisability due to overfitting or underfitting, the neural calibration method inherently focuses on the non-traditional setting of parameter

calibration. Depending on the use case of the ABM being fit (CLSNA, for ex.), having high predictive power on unseen data is *not* necessarily the goal. There are examples of ML methods deployed in literature where minimising training loss is strictly prioritised over model robustness, like Burgette and Reiter on multiple imputation in missing data via sequential regression trees [21] and Brookhart et al. performing variable selection in propensity score models [22]. The challenges of neural calibration mainly lie in ensuring that number parameters remain tractable and computations are expedient. The neural method has shown to calibrate models orders of magnitude more accurately than classical techniques while still running between '195 and 390 times faster' [7].

Functionally, the method operates similarly to an encoder-decoder model. It encodes the input data into a parameter space and decodes it to update prior beliefs about parameter estimates. This mechanism is crucial for continuously refining the model's accuracy and reliability, especially in dynamic environments where the underlying data patterns fluctuate frequently. Reparametrising this method in a formal VAE setting is the core of Chapter 5.

2.5 Concluding Remarks

In this chapter, we explored the evolution of agent-based models (ABMs) for opinion dynamics, emphasising an observed correlation between model parameterisation and the complexity of dynamics regimes. We then formally presented the CLSNA positive and negative partisanship model, stating the pros and cons of the calibration methods used thus far. The limitations of computational overhead and the requirement of calculating explicit formulations of log-likelihood posteriors motivate applying the neural calibration approach to the CLSNA model for two main reasons:

1. Successful adaptation of the neural calibration approach serves as a proof-of-concept for the viability of the methodology in social sciences settings.

2. Neural calibration circumvents the need for computing log-likelihood posteriors by utilising empirical loss density approximations instead. Evaluating this effect on both computational efficiency and quality of parameter estimation is of interest.

Chapter 3

Preliminaries

This chapter introduces the fundamental concepts that underpin the work in subsequent chapters. We present preliminaries on ML architectures and activation functions, evaluation metrics, and modern methods for sensitivity analysis.

3.1 Machine Learning Architectures

The neural parameter calibration technique defined in 2.3 requires defining an artificial neural network with learnable parameters θ to minimise a loss objective. Although all applications of this approach up to writing have defined this network as an MLP, it is easily extensible to arbitrary architectures. In this section, we outline the architectures used in 4 and 5.

3.1.1 Multi-Layer Perceptron and Backpropagation

A multi-layer perceptron (MLP) is a fundamental component of a basic neural network, consisting of several layers of nodes: an input layer, one or more hidden layers, and an output layer. In each layer, every node is fully connected to all nodes in the preceding layer, performing a linear combination per node that aggregates inputs using adjustable weights and biases. This is then crucially followed by applying a nonlinear activation function to produce the output. Figure 3.1 illustrates a single perceptron node. The forward propagation process in an MLP and is given by:

$$
\begin{aligned}
\mathbf{z}^{(1)} &= \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \\
\mathbf{a}^{(1)} &= \sigma(\mathbf{z}^{(1)}) \\
\mathbf{z}^{(2)} &= \mathbf{W}^{(2)}\mathbf{a}^{(1)} + \mathbf{b}^{(2)} \\
\mathbf{a}^{(2)} &= \sigma(\mathbf{z}^{(2)}) \\
&\vdots \\
\mathbf{z}^{(L)} &= \mathbf{W}^{(L)}\mathbf{a}^{(L-1)} + \mathbf{b}^{(L)} \\
\mathbf{y} &= \sigma(\mathbf{z}^{(L)})
\end{aligned}
$$

where $\mathbf{x}$ is the input vector, $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the weights and biases of the l-th layer respectively, $\mathbf{z}^{(l)}$ is the weighted sum of inputs to the l-th layer, $\mathbf{a}^{(l)}$ is the activation of the l-th layer, and σ is the activation function applied element-wise. MLPs build complexity through layer composition and can universally approximate unknown continuous functions of the form $f : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$ arbitrarily well [23].

Rumelhart et al. (1986) presented the backpropagation algorithm as a fundamental iterative technique for learning the weights and biases of MLPs [25]. At its core, backpropagation involves the computation of the gradient of a cost objective function concerning the network's learnable parameters. This gradient, which indicates the direction in which the weights adjust to minimise error, is calculated in a single backward pass from the output layer to the input layer. This pass uses the chain rule of calculus to flow loss error gradients back through the network, thus enabling

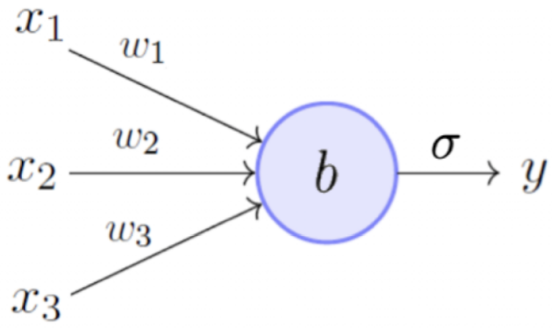


Figure 3.1: Perceptron with learnable weights $w1$, $w2$, $w3$, b and non-linear activation σ. Diagram inspired by Minsky and Papert [24]

the updating of weights not directly connected to the output. The backpropagation algorithm is expressed as follows:

$$\text{Output error: } \delta^{(L)} = \nabla_a C \odot \sigma'(z^{(L)})$$

$$\text{Backpropagated error: } \delta^{(l)} = ((w^{(l+1)})^T \delta^{(l+1)}) \odot \sigma'(z^{(l)})$$

$$\text{Gradient of the cost w.r.t. biases: } \frac{\partial C}{\partial b_j^{(l)}} = \delta_j^{(l)}$$

$$\text{Gradient of the cost w.r.t. weights: } \frac{\partial C}{\partial w_{jk}^{(l)}} = a_k^{(l-1)} \delta_j^{(l)}$$

where C is the cost function, σ' is the derivative of the activation function, $z^{(l)}$ is the input to layer l after applying weights, and $a^{(l)}$ is the output from the activation function at layer l. Autodifferentiation engines built into libraries like *torch.autograd* abstract away long gradient computation chains [26].

3.1.2 Convolutional Neural Network (CNN)

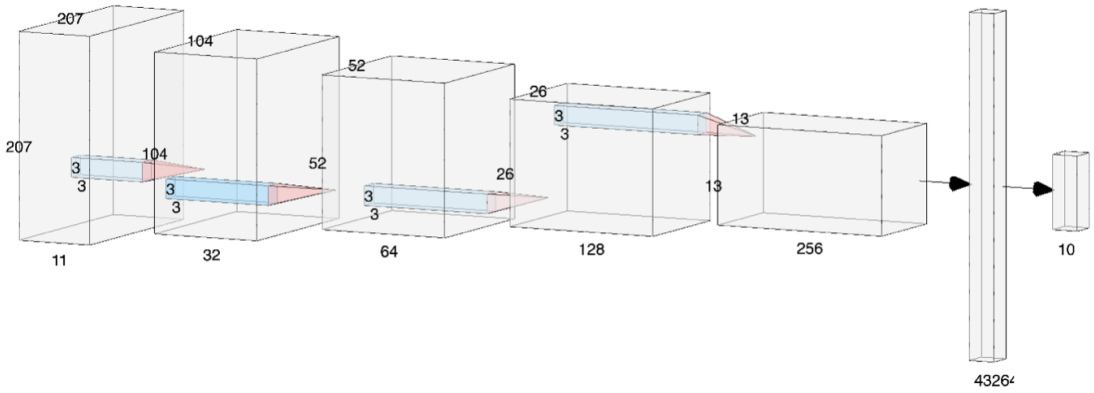


Figure 3.2: CNN with input tensor (11, 207, 207), four convolutional layers, and a fully-connected linear layer. Each convolutional layer has a (3,3) kernel and bias parameter.

CNNs were first introduced by LeCun et al. in the breakthrough paper "Handwritten Digit Recognition with a Back-Propagation Network" [27]. MLPs suffer from the curse of dimensionality where the number of parameters explodes due to the fully connected layer connections between nodes; this increases gradient computation exponentially and limits the ability to generalise to unseen input in image recognition tasks. CNNs offer a viable alternative that employs a restricted connection scheme whereby tensor model inputs are convolved with fixed-sized feature maps (i.e. 3x3

kernels) to generate output maps. As with the MLP, non-linear activation functions are applied to the outputs to allow for universal function approximation.

A by-product of the convolution operation is a notion of weight-sharing where the number of network parameters is a function of convolution kernel dimension and *not* of input dimension. Figure 3.2 is an example of a CNN used in this project. Although originally devised for image recognition, CNNs, also known as space-invariant artificial neural networks (SIANN), have proven to be effective in a larger subset of deep learning tasks that require topological feature extraction due to their capacity to extract spatial patterns from input tensors [28].

The three main building blocks of CNNs are convolution, pooling/aggregation, and fully connected layers. By composing convolutional layers, CNNs automatically and adaptively learn low and high-level spatial hierarchies. The learning process for CNNs is equivalent to an MLP where gradients for kernel parameters/biases are calculated by backpropagation during SGD optimisation.

3.1.3 Graph Convolutional Network (GCN)

Graph Convolutional Networks (GCNs) have become a leading framework in deep learning for learning graph representations, achieving high performance across various applications [29]. First introduced by Kipf and Welling [30], GCNs are designed to integrate topological information through the underlying *adjacency matrix*. The core idea is to perform convolution operations on the graph, restricting feature aggregation to only neighbouring nodes. The forward pass of a GCN layer is formulated as follows:

$$\mathbf{H}^{(l+1)} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right)$$

where:

- $\mathbf{H}^{(l)}$ is the feature matrix at layer l, with $\mathbf{H}^{(0)} = \mathbf{X} \in \mathbb{R}^{n\times d}$, the input feature matrix where the columns of $\mathbf{X}$ represent the d features.
- $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix with added self-loops (1's along the diagonal introduced by identity matrix $\mathbf{I}$).
- $\tilde{\mathbf{D}}$ is the degree matrix of $\tilde{\mathbf{A}}$.
- $\mathbf{W}^{(l)}$ is the trainable weight matrix of layer l.
- σ is a non-linear activation function.

This formulation is derived as a first-order approximation of spectral graph convolutions, ensuring that the feature transformation and propagation steps are intertwined. This allows the model to capture both node features and graph structure effectively. The degree matrix $\tilde{\mathbf{D}}$ is employed to normalise the influence of each node's neighbours, preventing potential issues with unevenly distributed node degrees. GCNs are particularly 'powerful in the setting where the adjacency matrix contains information to present in the model input X' [30], and it is this core idea that motivates their use in the CLSNA model.

3.2 Activation Function Saturation

Activation functions (AFs) introduce non-linearity into ML architectures, enabling them to learn complex, non-convex patterns. However, when the inputs to these functions fall into extreme ranges (i.e. the input signal maps to an asymptotic value), the derivatives of the activation functions approach zero, leading to a phenomenon known as saturation or vanishing gradient where the gradient does not flow backwards [31]. The property of function saturation is expressed as:

$$\lim_{|v|\to\infty} |\nabla f(v)| = 0.$$

Saturation impedes the training of deep networks, as the gradients become too small to effect meaningful weight updates in update steps. This makes achieving model convergence in training

a delicate matter, sensitive to weight initialisation, learning rate, and optimiser strategy. This challenge is particularly acute in deep networks where multiple layers exacerbate the issue, leading to slow or failed convergence.

Rakitianskaia and Engelbrecht [32] propose a novel simple measure for quantifying saturation levels of arbitrary bounded functions, which we employ when considering the CLSNA numerical solver as an activation function in 5. This metric is calculated from the frequency distribution (like in Figure 3.3) of the activation function outputs, $g(net)$, over a valid input domain.

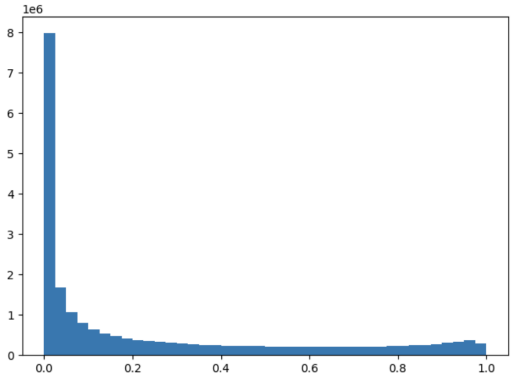


Figure 3.3: Frequency histogram of a positively-skewed, bounded, and saturated NN unit

The measure is defined as follows. First, the average output signal value for each bin b is calculated as follows:

$$\bar{g}_b = \begin{cases} \frac{\sum_{k=1}^{f_b} g(\text{net}_k)}{f_b} & \text{if } f_b > 0 \\ 0 & \text{otherwise} \end{cases}$$

In this equation, f_b denotes the count of output signals $g(\text{net})$ falling into bin b. When the values of g are centred around zero, the absolute average $\bar{g}_b$ tends to be higher for bins nearer to the asymptotic values and lower for bins closer to the midpoint. If g spans the range $[g_L, g_U]$, the value $\bar{g}_b$ can be normalized to the interval $[-1, 1]$ using the following transformation:

$$\tilde{g}'_b = \frac{2(\bar{g}_b - g_L)}{g_U - g_L} - 1$$

We then compute a non-negative weighted mean of absolute $|\tilde{g}'_b|$:

$$\varphi_B = \frac{\sum_{b=1}^{B} |\tilde{g}'_b| f_b}{\sum_{b=1}^{B} f_b}$$

where B is the total number of bins, and f_b assigns a probability weight of each bin. This measure φ_B serves as an indicator of saturation, with values approaching 1 indicating high saturation and values nearing zero indicating low saturation.

3.3 Loss Functions and Evaluation Metrics

3.3.1 Binary Cross-Entropy (BCE) and Weighted BCE

Binary Cross-Entropy (BCE) is a standard metric in information theory, used to quantify the difference between two probability distributions. In machine learning, particularly for binary classification tasks, it is the primary loss function to compare the ground truth labels with the predicted probabilities. The average BCE loss for a dataset containing N instances is calculated as follows [33]:

$$\text{BCE}_{\text{avg}} = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

where:

- y_i is the true binary label (0 or 1) for instance i.
- $\hat{y}_i$ is the predicted probability of instance i being in class 1.

In many real-world datasets, class imbalances are common, where one class is significantly more frequent than the other. This imbalance can cause the standard BCE loss to be biased towards the majority class and lead to degenerate behaviour [34]. To address this issue, we use a weighted version of BCE. The Weighted BCE loss introduces a weight for each class, allowing higher penalisation for misclassification of the less sampled class, thereby encouraging the model to learn both classes in a more balanced manner. The weighted BCE loss is given by [33]:

$$\text{Weighted BCE}_{\text{avg}} = -\frac{1}{N}\sum_{i=1}^{N}[w_1 y_i \log(\hat{y}_i) + w_0(1-y_i)\log(1-\hat{y}_i)]$$

where w_1 is the weight for the positive class (class 1) and w_0 is the weight of the negative class (class 0). Choosing appropriate weights w_1 and w_0 can help mitigate the effect of class imbalance, ensuring that the loss function gives equal importance to both classes. A common approach to choosing these weights is inversely proportional to the class frequencies:

$$w_1 = \frac{N}{2N_1}, \quad w_0 = \frac{N}{2N_0}$$

where N_1 are the counts of class 1 and 2 instances, respectively. Using Weighted BCE helps in scenarios where the data is highly imbalanced by ensuring that the minority class contributes more significantly to the loss, leading to better performance in classification tasks.

Precision measures the proportion of correctly identified positive cases out of all instances that were predicted as positive. It is given by:

$$\text{Precision} = \frac{TP}{TP + FP}$$

where TP (True Positives) is the count of correctly predicted positive cases, and FP (False Positives) is the count of negative instances incorrectly predicted as positive.

Recall (also known as Sensitivity) is the ratio of actual positive cases correctly identified by the model. It is defined as:

$$\text{Recall} = \frac{TP}{TP + FN}$$

where FN (False Negatives) is the count of incorrectly classified positive instances.

F1 Score is the harmonic mean of Precision and Recall, providing a single metric that balances precision and recall. It is given by:

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

3.3.2 Topological Graph Metrics

For the CLSNA model and similar matrix prediction tasks, graph metrics can provide deeper insights by analysing the structural properties of graphs. This section presents the measures used in Sections 4 and 5.

Clustering Coefficient

The clustering coefficient quantifies the tendency of nodes in a graph to cluster together, reflecting the presence of tightly knit sub-groups. This is both a local node and a general graph metric. For a given node v, the local clustering coefficient is the ratio of the number of triangles (closed triplets of nodes) to the number of potential triangles through that node:

$$C_v = \frac{2\Delta_v}{k_v(k_v - 1)}$$

where Δ_v represents the number of triangles through node v, and k_v is the degree of node v (number of neighbours). For graphs with weighted edges, the clustering coefficient can modified to consider non-binary edge weights [35].

Degree Centrality

Degree centrality is a basic measure of a node's importance in a graph, defined by the number of connections (edges) a node has. For a node v, the degree centrality $D(v)$ is expressed as:

$$D(v) = k_v$$

where k_v is the degree of node v. In normalized form, degree centrality is given by:

$$D(v) = \frac{k_v}{N-1}$$

where N is the total number of nodes in the graph.

Closeness Centrality

Closeness centrality is a distance metric that measures how near a node is to all other nodes in a graph, summarising the node's efficiency in disseminating information to others and is frequently used for identifying influential nodes in social networks [36]. The closeness centrality $C(v)$ of a node v is the inverse of the sum of the shortest path distances from v to all other nodes:

$$C(v) = \frac{N-1}{\sum_{u \in V} d(v,u)}$$

where N is the number of nodes in the graph, V denotes the set of all nodes, and $d(v,u)$ is the shortest path distance between nodes v and u.

3.4 Sensitivity Analysis (SA)

In Saltelli et al.'s book "Sensitivity Analysis in Practice", SA is defined as 'the study of how the uncertainty in the output of a model (numerical or otherwise) is apportioned to different sources of uncertainty in the model input' [37]. This definition shifts the focus on SA as a local measure of effects on given inputs on given outputs (i.e. with partial derivatives) to a global notion that incorporates higher-order parameter interactions. These techniques are extensively employed in the scientific literature for higher-level model evaluations that include validation, optimisation, and decision-making processes.

The properties of an ideal sensitivity analysis method are [37]:

- Ability to handle the influence of model scale and shape when producing analysis metrics.
- Inclusion of multidimensional averaging to evaluate interactions between model parameters.
- Model agnosticism.
- Ability to group parameters and evaluate them as individual factors.

3.4.1 Morris Method

The Morris Method (MM) or Elementary Effects Method is a screening technique designed to pinpoint the parameters that significantly impact model outputs. This method focuses on local sensitivity by computing approximations of output f partial derivatives at different points in the parameter space.

The three primary steps of the Morris Method are detailed below [38]:

- Discretising the Input Space: The input sense must be sensibly chosen to lie in a realistic subset within the parameter space to limit wasted computation and yield the best results.

- Sampling the input space: The *One-At-a-Time (OAT)* sampling strategy is used. Starting from an initial randomly chosen point in the input space, one input parameter is changed to the next "level" up or down in the grid while holding all other parameters constant, generating a new point in the input space. This process is sequentially repeated for each input parameter, resulting in *trajectories* through the input space that explores one dimension at a time.
- Elementary Effect (EE) Calculation: The EE of an input parameter is calculated as the change in the model's output divided by the change in the input parameter. This is done for each step in the trajectory. Mathematically:

$$\mathrm{EE}_i = \frac{f(\mathbf{x}_i^+) - f(\mathbf{x})}{\Delta}$$

where $\mathbf{x} = (x_1, x_2, ..., x_k)$ is an arbitrary point in input space such that $\mathbf{x}_i^+ := \mathbf{x} + \mathbf{e}_i\Delta$ is still in input space. Δ is the change in the input parameter.

The mean of the absolute values of the elementary effects ($\mu^*_{|EE_i|}$) indicates the overall impact of an input parameter on the output. Larger μ_{EE} values suggest that the parameter has a significant influence on the output. The standard deviation of the elementary effects (σ_{EE}) reflects the non-linearity and interaction effects with other parameters; a high value indicates that the parameter's effect varies greatly under different conditions and strongly interacts with other parameters.

The Morris Method is useful for a specific sensitivity analysis task known as factor fixing (FF). In this paradigm, the objective is to identify a subset of factors $\lambda' \subset \lambda$ that can be fixed at any value over a range of uncertainty without significantly affecting the output variance.

3.4.2 Variance-Based Methods: Sobol Indices

Variance-based methods decompose the model variance into components attributable to different inputs, helping to pinpoint which parameters significantly impact output uncertainty. These methods are advantageous because they consider higher-order interactions. Two factors are said to interact when their combined effect on the output cannot be linearly decomposed to single effects on output [37]. Formally, the interaction effect between two factors, X_i and X_r, on the output Y is defined by the conditional expectation variance:

$$V_{ir} = V(E(Y \mid X_i, X_r)) - V(E(Y \mid X_i)) - V(E(Y \mid X_r)). \tag{3.1}$$

Here, $V(E(Y \mid X_i, X_r))$ represents the joint effect of the factors X_i and X_r on Y. Equation 3.1 is known as a second-order effect. Analogous formulae are derived for higher orders.

The goal in this context is to rank the factors based on how much of the output variance is eliminated when the true value of a particular input factor X_i is known. To do this, factors are ranked according to $V(Y \mid X_i = x_i^*)$, which is the variance when X_i is fixed at its true, but unknown, value x_i^*. Since x_i^* is unknown, we use empirical averages over all possible values x_i^* of X_i, that is, $E(V(Y \mid X_i))$. Given that $V(Y)$ is constant and $V(Y) = V(E(Y \mid X_i)) + E(V(Y \mid X_i))$, minimizing $E(V(Y \mid X_i))$ is equivalent to maximizing $V(E(Y \mid X_i))$, which is what we aim to compute [37].

One common approach is the Sobol method, which calculates the total variance of the output and partitions it into fractions that is attributed to individual inputs or combinations of inputs using Monte Carlo estimates [39]. The Sobol indices (first and higher-order are defined as follows):

$$S_{ir} = \frac{V_{ir}}{V(Y)}$$

where S_{ir} is Sobol' sensitivity index for the i, r-th input factor. Finally, the total-effect index S_{Ti} for each factor X_i includes all variance contributions involving X_i, making it a comprehensive measure of the input factor's influence. This is used when the number of parameters is large, say d, because calculating all higher-order indices would require $2^d - 1$ computations.

3.4.3 Multi-Objective Optimisation (MOO) and NSGA-II

Multi-Objective Optimisation (MOO) is a branch of optimisation that simultaneously minimises (or maximises) multiple objective functions to be optimised. Unlike single-objective optimisation which looks for a single best solution, MOO results in a set of solutions that balance trade-offs between competing objectives, known as *Pareto optimal solutions*. Mathematically, MOO is formalised as:

$$\begin{aligned} &\min \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})) \\ &\text{subject to } \mathbf{x} \in \Omega \end{aligned}$$

where $\mathbf{f}(\mathbf{x})$ is a vector of objective functions, and Ω represents the feasible region defined by the constraints [40]. The Pareto front, or boundary, is the set of non-dominated solutions in the objective space. A solution is non-dominated if no other solution can improve one objective without performing worse in at least one other objective:

$$\text{Pareto Front} = \{\mathbf{f}(\mathbf{x}) \mid \mathbf{y} \in \Omega \text{ such that } \mathbf{f}(\mathbf{y}) \preceq \mathbf{f}(\mathbf{x}) \wedge \mathbf{f}(\mathbf{y}) \neq \mathbf{f}(\mathbf{x})\}$$

where $\preceq$ denotes the component-wise comparison for dominance.

In traditional optimisation, multiple objectives aggregate into a single objective, typically through convex combinations in a process known as objective weighting (OW) or through min-max formulations aimed at minimising the maximum deviation from the optimal value of each objective. The most significant drawback of these approaches is the requirement of thorough prior knowledge regarding the different objectives (i.e. to determine weights in OW) [41]. Recent alternatives use non-dominated, sorting-based evolutionary algorithms (MOEAs) such as NSGA-II [42]. In this method, a population of solutions evolves over multiple generations to approximate the Pareto front. The NSGA-II algorithm preserves a diverse set of solutions by combining non-dominated sorting with a crowding distance mechanism, ensuring genetic diversity among the solutions.

The genetic algorithm in NSGA-II involves several steps:

- **Initialisation:** A population of potential solutions is randomly generated.
- **Selection:** The existing population is divided into different fronts based on Pareto dominance, with solutions in the first front being non-dominated. Being on a higher front ensures a higher probability of survival for the next generation.
- **Crossover and Mutation:** The Genetic operators of crossover and mutation are applied to selected solutions to produce new offspring. During crossover, components of 2 parent solutions combined to create new solutions. Mutation involves random modifications to individuals to ensure genetic diversity is maintained.
- **Elitism:** NSGA-II uses an elitist strategy where the best solutions (based on Pareto dominance and crowding distance) carry over to the next generation.

Chapter 4

CLSNA Neural Calibration

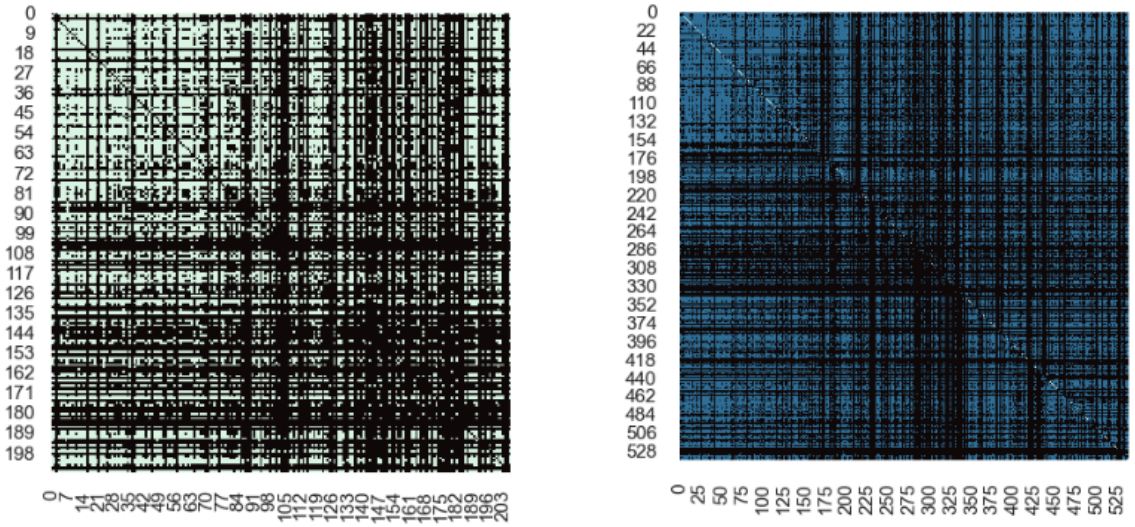
In this chapter, we outline the implementation details and evaluate the results of neural parameter calibration on the CLSNA opinion dynamics model. Concretely, the **problem statement** is to learn empirical probability density functions for the five model parameters $\lambda = \{\alpha, \delta, \gamma_w^1, \gamma_w^2, \gamma_b\}$. The three variance parameters of CLSNA are fixed ($\sigma^2 = 1, \tau^2 = 10, \phi^2 = 10$) and the latent space dimension (d) is set to 2 to facilitate fair results comparison between our solution and Zhu et al.'s work.

Important Note: Throughout this chapter, we discuss the implementation of neural parameter calibration on two versions of the CLSNA model: Zhu et al.'s fixed node formulation [4] and Pan et al.'s dynamic node model [5]. The differences are outlined in Section 2.2. In what follows, the fixed node variant will be referred to as the **207** model and the dynamic node variant the **616** model for reasons made clear in Section 4.1.

4.1 Dataset(s): Tweet Hashtag Network

The data was provided by Zhu et al. [4] and Pan et al. [5]. In this section, we formally present the structure of this data and perform preliminary exploratory analysis.

4.1.1 Structure



(a) 207 Model Adjacency Matrix at $t = 10$

(b) 616 Model Adjacency Matrix at $t = 10$

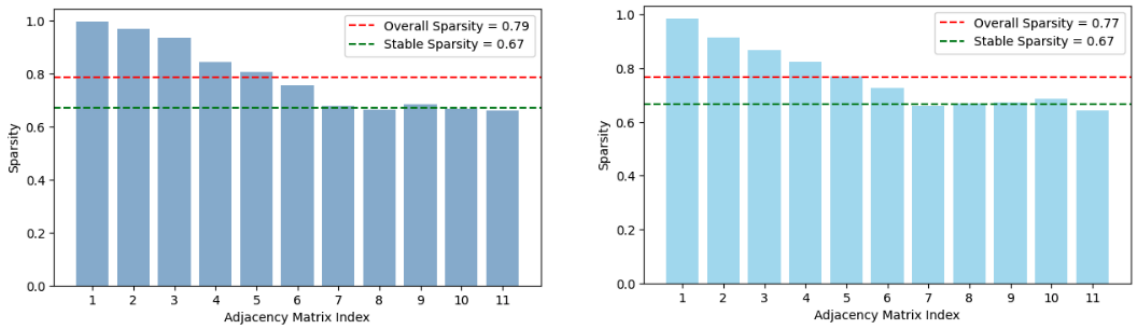
Figure 4.1: Adjacency matrices at $t = 10$ for the **207** model and **616** model. In both matrices, black corresponds to a 0 entry. See Appendix B.1 for full dataset plots.

For both the **207** and **616** implementations, the dataset comprises of tweets posted by each US member of Congress (MOC) with a valid handle from 2010 to 2020, resulting in 796 accounts, 843,907 tweets, and 1,252,455 instances of hashtag sharing (after excluding retweets and tweets without hashtags). Then we created time-indexed binary symmetric adjacency matrices for each year, where $t \in \mathcal{T} := \{0, \ldots, 11\}$ and each $Y_t \in \{0,1\}^{N_t \times N_t}$ for N_t nodes at time t. Additionally, for each time index t, a binary list P_t was generated that maps each node index to a political party (1 for Democrat, 0 for Republican). Consequently, for each time index t, we obtain a tuple consisting of a matrix and a party list (Y_t, P_t). The nodes in these matrices represent sitting members of Congress, and edges denote that the number of common hashtags tweeted by both members of Congress that year was above the network average. The rationale behind using hashtag sharing is to measure polarisation in terms of *indirect engagement*, where a connection signifies participation in similar topics [4].

For the fixed node dataset, we filtered the members of Congress to include only those in office throughout the 10-year period. This filtering reduced the dataset to 207x207 adjacency matrices, with 131 Democrats and 76 Republicans. Meanwhile, the **616** model has a varying number of nodes per time index, with a maximum N_t of 616, which occurred in the year 2019 ($t = 9$). Due to the dynamic nature of the adjacency matrices, the dataset also includes user-handle dictionaries that map index to handle. This information is strictly necessary for the implementation of `simulate_clsna_dynamic` algorithm (see Appendix A).

4.1.2 Exploratory Analysis

The focus of this exploratory analysis is on the topological structure of the **207** and **616** datasets. In Figure 4.7, we notice that in both datasets, graph sparsity starts very high (nearing 1.0) and soon stabilises to equilibrium; in fact, the evolution of sparsity is strikingly similar. The reason for high initial sparsity is unclear as we do not expect vastly different platform behaviour ceteris paribus in earlier and later years. In the **207** case, we see a stable matrix sparsity of 0.67, calculated as the average sparsity over the final 5 adjacency matrices. In the **616** dataset, the stable sparsity is exactly equal to the former (calculated the same way) and is in line with the intuition that with a relatively large number of nodes, we expect convergent behaviour. Due to the sparse nature of the dataset and the fact that the ML task at hand is the binary classification of the adjacency matrices, there is a clear **class imbalance** that we must consider when designing an architectural solution.



(a) 207 Model Data Sparsity

(b) 616 Model Data Sparsity

Figure 4.2

Furthermore, the average clustering coefficient over the 11 207x207 adjacency matrices in the fixed node dataset is 0.42, jumping to 0.657 when only considering the final five data entries. This clustering coefficient is significant and implies that the dataset exhibits small-world characteristics. Namely, the short mean path length amongst connected components and clustering coefficient is greater than 0.6. In the **616** dataset, we observe a similar structure with an average clustering coefficient of 0.468 that rises to 0.667 over the final five matrices.

Clearly, from both the high clustering coefficient(s) and horizontal and vertical patterns visually apparent in the matrices of Figure 4.1, a significant spatial component exists in the dataset(s)

along with the assumed temporal relationship; incorporating this information into our solution would allow us to capture more semantically meaningful parameter calibrations. An initial hypothesis is that the datasets follow the universal power law described in Barabasi and Albert's influential paper "Emergence of Scaling in Random Networks" [43]. The Barabasi-Albert model states that the probability $P(k)$ of a node in the network having k connections follows a power law distribution, $P(k) \sim k^{-\gamma}$, where γ is typically between 2 and 3. However, upon analysing our datasets, it became clear that the Barabasi-Albert model was **not** suitable because many nodes had a high degree, suggesting that additional factors affect the network structure.

One such factor is the selection bias inherent in our dataset, as it includes members of Congress whose digital behaviour is governed by political agendas and public responsibility. This bias impacts the network's structure, making it crucial to consider the political context of connections. Instead of a universal power law, the connection patterns are better understood by distinguishing between inter-party and intra-party connections. Figure 4.3 shows a (207,207) binary matrix where a value of 1 for index (i, j) indicates party of N_i and N_j is equal and 0 otherwise. We can see coarse visual patterns in line with what is observed in the ground truth data.

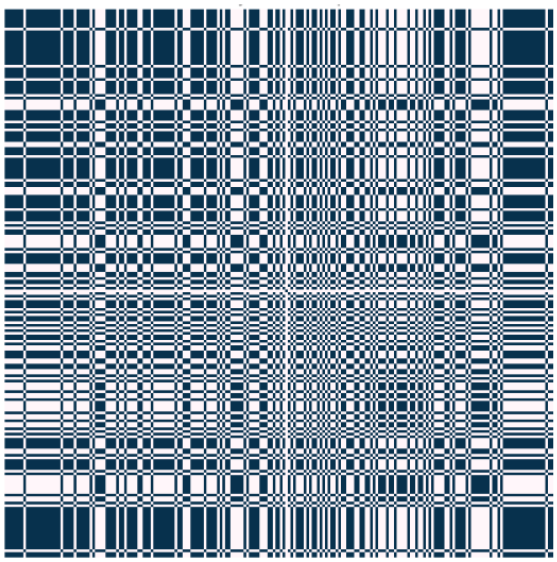


Figure 4.3: Political Party Adjacency Matrix in **207** dataset

4.2 Implementation Overview

In this section, we present the implementation of neural parameter calibration for both the **207** and **616** datasets. Our approach follows a systematic approach aligned with the neural parameter calibration methodology. The main steps of the implementation are as follows:

1. **Define the model architectures:** We define the architectures of the calibration models, specifically detailing the layers and components used to capture the spatial and temporal dynamics of the networks.
2. **Define the numerical solver:** We implement the algorithm that takes $\hat{\lambda}$ as input and outputs predicted adjacency matrices $\{Y_t : t \in \mathcal{T}\}$ governed by dynamics of CLSNA. This can be used in downstream tasks (i.e., calculating loss function). Albeit a challenging task, the implementation details are not directly relevant to neural calibration and thus are detailed in Appendix A.
3. **Define the loss Function:** We outline the loss functions used to train the models, motivating the use of various regularisation techniques to guide training.
4. **Establish a training procedure:** We describe the training process, including the optimisation techniques, hyperparameter tuning, and the training function.

5. **Define performance Metrics:** We establish the metrics used to evaluate the models' performance, ensuring a comprehensive assessment of their effectiveness.

6. **Evaluate Results:** We analyse the results obtained from the models, comparing their performance and discussing the insights gained from the evaluation.

Throughout the subsequent sections, we will clearly outline any differences in implementation between the **207** and **616** models. Efforts have been made to abstract these differences whenever possible, ensuring the implementation is modular and extensible. [1]

4.3 Model Architectures

The first step of neural calibration is defining the ML architectures that take as input the observed, ground-truth data Y_t and output predicted parameters $\hat{\lambda}$.

4.3.1 207 Dataset

For the **207** dataset, we have implemented three different architectures: a basic NN, a CNN, and a GNN. The fixed number of nodes allows us to reshape the input into a (11, 207, 207) tensor, making it feasible to apply convolutional operations for the latter architectures.

Basic Neural Network

The NN architecture is fully connected, taking as input the concatenation of Y_t with shape (234531,). As Y_t is symmetric, we only vectorise the upper triangular representations of $U(Y_t)$ to minimise latency overhead. The NN is (hyper-) parameterised as follows:

- **Number of Layers:** 5
- **Nodes per Layer:** 96
- **Activation Functions:** Sigmoid for hidden layers, linear (id) for the last layer
- **Dropout Rate:** 0.163256
- **Learning Rate:** 0.0002
- **Optimiser:** RMSProp [44] with weight decay 0.011258

The determination of these values is discussed in Section 4.5. Additionally, we define an auxiliary lightweight NN that initialises the main model's outputs to normal priors with a mean of 0.0 and a standard deviation of 0.5. This NN trains quickly (17.3 $\pm$ 1.6s) and assumes white-noise inputs. The inspiration for this type of weight initialisation is from Gaskin et al.'s NeuralABM GitHub repository [45]. This model has **22,552,320** parameters and a storage requirement of **21.508MB**. Compared to traditional NNs, this is a larger model with high capacity. Due to reasons outlined in Section 2.4, we do not mind the high parameter count as long as execution time is kept low (i.e., with GPU parallelisation).

Convolutional Neural Network

The convolutional neural network (CNN) architecture incorporates spatial and temporal structure information of the adjacency matrices by applying a series of 3x3 convolutions over input/hidden tensor maps. The CNN is (hyper-) parameterised as follows:

- **Input Channels:** 11
- **Number of Convolutional Layers:** 5
- **Kernel Shape:** (3, 3)
- **Convolution Stride:** 2

[1] NeuralCLSNA GitLab Repository: https://gitlab.doc.ic.ac.uk/og519/nerualclsna

- **Hidden Dimension$_0$:** 32, which duplicates in each subsequent layer
- **Activation Function:** ReLU for hidden layers, linear for output
- **Learning Rate:** 1e-5
- **Optimiser:** Adam [46]

In this architecture, we excluded pooling layers between convolutional layers as it worsened the predictive power of the trained models; instead, a stride of 2 allows for height and width tensor downsampling when encoding input to parameter values. Intuitively, this model resembles the first half of the widely recognised U-net architecture commonly used in graph-reconstruction tasks [47]. This CNN model has **2,003,525** parameters and a storage requirement of **1.911MB**. Compared with the previously defined NN, it has 91.12% fewer parameters and a 91.149% lower storage requirement due to weight-sharing.

Graph Neural Network

The graph neural network (GNN) architecture builds off the key insight from the exploratory analysis (Section 4.1.2) that the spatial patterns observed in the data roughly mirror the partisan relationships between nodes. We concretise this notion by defining A_{party} as a binary adjacency matrix of dimension 207×207 where:

$$A_{party}(i,j) = \begin{cases} 1 & \text{if party}(i) = \text{party}(j) \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

By enforcing the ML model training regime to diffuse information through K-neighbourhoods built from prior knowledge A_{party}, we expect improved performance compared to the learned relationships of the CNN.

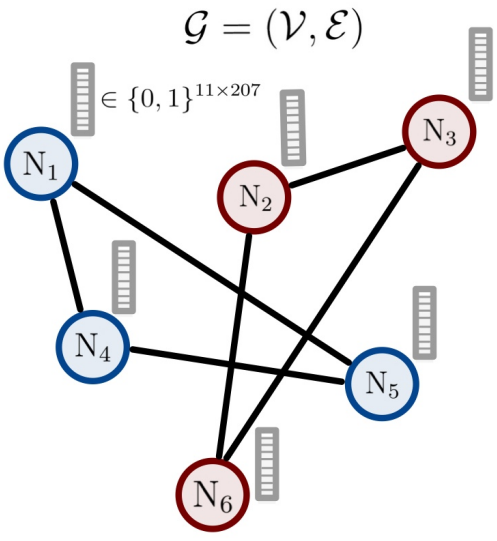


Figure 4.4: 6 Node CLSNA where nodes in class 1 are blue and in class 2 are red. The (unweighted) edges are present when nodes belong to the same class. Each node has a node embedding in $\{0,1\}^{11\times207}$.

For the GNN implementation, we reshape the (11, 207, 207) ground-truth tensor into a (207, 11 * 207) 2D tensor using a vectorisation operation. This transformation is identical to the one used in Isallari and Rekik's brain-graph super-resolution architecture [48] that assigns each node a node-embedding in $\{0,1\}^{11\times207}$. By doing this, we enable matrix multiplication with the (207, 207) normalised adjacency matrix $\tilde{A}$ in the graph convolutional network (GCN) layers. This approach encodes temporal (11-time steps) and spatial information into node features, meaning that even

with an unweighted $\tilde{A} \in \{0,1\}^{N \times N}$, each node can still incorporate information from all nodes, not just from those within the same political party. The adjacency matrix enforces a topological structure that dictates how graph information diffuses across node neighbourhoods. Figure 4.4 is a toy-example of the CLSNA $\mathcal{G}$ with six nodes.

The GNN is (hyper-) parameterised as follows:

- **Input Channels:** 11 * 207
- **Number of Layers:** 3 GCN layers
- **Hidden Dimension:** 128
- **Activation Function:** ReLU for hidden layers, linear for output
- **Learning Rate:** 1e-5
- **Optimiser:** Adam

The convolution operation in the GCN layers is defined as:

$$\text{ReLU}(\tilde{A} \cdot H \cdot W) \tag{4.2}$$

where $\tilde{A}$ is the normalised adjacency matrix, H is the input feature matrix, and W represents the learnable weights. The output of the final GCN layer is then averaged across the node dimension to produce the final output in the desired form $\hat{\lambda}$. This averaging operation ensures that the output captures the aggregated information from all 207 nodes. This GNN model has **308,480** parameters and a storage requirement of **0.294MB**. This is by far the most lightweight of the three presented, with orders of magnitude lower resource demands compared to the vanilla NN and CNN.

4.3.2 616 Dataset

For the **616** model, the only viable implementation was a basic NN because at each time index, there is a variable number of nodes, making convolutional operations impractical. The traditional workarounds of zero-padding resizing [49], interpolation scale-up or cropping is more applicable in computer vision tasks like image segmentation or classification but less so when the entire graph topology needs consideration. The network takes as input the concatenation of Y_t with a shape of (1382495,). The NN is (hyper-) parameterised as follows:

- **Number of Layers:** 5
- **Nodes per Layer:** 32
- **Activation Functions:** Sigmoid for hidden layers, linear (id) for the last layer
- **Dropout Rate:** 0.163256
- **Learning Rate:** 1e-4
- **Optimiser:** Adam with weight decay 0.011258

Similar to the vanilla NN implementation for the **207** model, we define an auxiliary lightweight NN to initialise the main model's outputs to normal priors with a mean of 0.0 and a standard deviation of 0.5. However, this NN does not train as quickly as the fixed-node counterpart, taking approximately 43.2 ± 3.4 seconds. This model has **44,244,096** parameters and a storage requirement of **42.194MB**. This highlights a significant issue with the basic NN approach: scalability.

Consider an input size of N. In a fully connected network, the input vector must be 1D, causing the number of input nodes to grow linearly with N. Consequently, the number of model parameters becomes dominated by the first layer weights that connect all input nodes to a hidden dimension, which scales as $\mathcal{O}(N^2)$. In the hyperparameter sweep conducted to optimise the **616** model, the search space extended up to 128 nodes per layer, resulting in models with **177,025,536** parameters! Unlike fully connected NNs, CNNs and GNNs do not suffer from a parameter dependency on input size. Transforming the **616** data into a semantically meaningful format compatible with these architectures remains an area for future work.

4.4 Loss Function

The primary loss function is the weighted mean BCE loss, accounting for class imbalance:

$$L_{\text{BCE}} = -\frac{1}{N}\sum_{n=1}^{N} w_n \left[y_n \log x_n + (1-y_n)\log(1-x_n)\right] \tag{4.3}$$

where y_n is the target, x_n is the predicted probability, N is the number of samples, and w_n is the weight set to 3.0. Additionally, we define two regularisation terms:

1. L2 prior regularisation term inspired by the work of Zhu et al., which represents belief a priori regarding the underlying system dynamics. In this case the priors were 0.0 for both $\{\alpha, \delta\}$, 0.5 for $\{\gamma_w^1, \gamma_w^2\}$, and -0.5 for γ_b. Formally:

$$L_2 = \sum_i \left(\frac{(\lambda_i - \lambda_{\text{prior}}[i])^2}{2}\right) \tag{4.4}$$

2. A regularisation term that emphasises the persistence of 1s or 0s in the adjacency matrix over time, maintaining correlations from time t to $t+1$:

$$L_3 = \frac{1}{N^2 T}\sum_{t=1}^{T-1} \text{XOR}(Y_t, Y_{t+1}) \tag{4.5}$$

where XOR is the logical XOR operation. The total loss is a weighted sum of the BCE loss and the regularisation terms:

$$L_{\text{total}} = L_{\text{BCE}} + \gamma L_2 + \beta L_3 \tag{4.6}$$

where γ and β are coefficients for the regularisation terms. In our implementation, we set $\gamma = 1.0$ and $\beta = 1.5$, determined by hyperparameter sweep. This loss function ensures that the neural parameter calibration not only fits the observed data but also maintains temporal consistency and updates prior beliefs.

4.5 Training Procedure

The training of our neural parameter calibration model is unique due to the nature of the data available. We only have one realisation of a 10-year time-series $\{Y_t\}$, which requires injection of regularisation noise during training to assure convergence. Fortunately, the CLSNA model is probabilistic, sampling from a Bernoulli distribution for each element $Y_{t,ij}$, thereby inherently propagating noise in the backpropagation gradient flow. In addition, we also have employed traditional ML regularisation techniques such as weight decay and node dropout.

Weight decay, also known as L2 regularisation, is a technique that penalises large node weights by incorporating an additional term into the loss function that is proportional to the sum of the squared weights. This helps prevent overfitting by discouraging the model from becoming too complex and sensitive to the training data [50]. Node dropout is another regularisation technique where, during training, a random subset of nodes are ignored or shut off. This forces the network to learn redundant representations, which improves generalisation by preventing the model from relying too heavily on any particular node [51].

Lastly, for the training of the CNN and GNN in the 207 model, we have applied gradient clipping to control and smoothen our model's traversal of the parameter space. This was similarly done by Pan et al. in their SGD calibration approach [5] where the gradient descent update rule is:

$$\theta^{(k+1)} = \theta^{(k)} - C\rho\,\text{sign}\left(\frac{\partial \log \pi_{\text{SGD}}(\theta^{(k)}, Z_{1:T}^{(k)} \mid Y_{1:T})}{\partial\theta}\right) \tag{4.7}$$

where ρ is the learning rate and C clipping constant is set to 0.25. This adjustment is employed because the magnitude and variance of the related gradient terms are much larger than the magnitude and variance of the gradients of the latent positions.

Formally, Algorithm 1 outlines the key steps of the training procedure. After execution of the training loop, the ML model (NN, CNN, or GNN) f_θ will have updated parameter weights and bias state. In practice, we additionally store and return the loss values calculated at each epoch to perform density estimation as outlined in Section 2.3.1.

Algorithm 1 Training Loop

Input: Network time-series $Y_{1:T}$. Initial value θ_0 of ML model, f_θ. M: Number of epochs. λ^{prior}: Prior values for L2 λ-regularisation. ρ: Learning Rate.

Output: Parameter estimates $\hat{\lambda} := (\hat{\alpha}, \hat{\delta}, \hat{\gamma_w^1}, \hat{\gamma_w^2}, \hat{\gamma_b})$.

for $k \leftarrow 0$ to M **do**
  $\hat{\lambda}_k \leftarrow f_\theta(Y_{1:T})$
  $(\hat{Y_{1:T}}, \hat{P_{1:T}}) \leftarrow$ `simulate_clsna`$(\hat{\lambda}_k)$ ▷ Or `simulate_clsna_dynamic` for 616 model
  Calculate $L_{total} \leftarrow L_1 + \gamma \times L_2 + \beta \times L_3$ ▷ L_{total} function of $(\hat{Y_{1:T}}, \hat{P_{1:T}})$
  Calculate gradient $\nabla_\theta L_{total}$
  Take an optimiser step: ▷ Apply gradient clipping [Eqn. 4.7] for CNN/GNN f_θ

$$\theta^{(k+1)} = \theta^{(k)} - \rho \left(\frac{\partial \log \pi_{\text{SGD}}(\theta^{(k)}, Z_{1:T}^{(k)} | Y_{1:T})}{\partial \theta} \right)$$

end for
Predict Final λ estimate: $\hat{\lambda}_M \leftarrow f_\theta(Y_{1:T})$
return $\hat{\lambda}_M$

Training the models with the `simulate_clsna` numerical solver proved to be extremely sensitive to weight initialisation (θ_0). As outlined in the basic NN model architecture (Section 4.3.1), the auxiliary NN that was trained to map random white-noise model inputs to initial parameter priors had to be configured to mean $\mu_{\lambda_0} = (0.0, 0.0, 0.0, 0.0, 0.0)$ and standard deviation $\sigma_{\lambda_0} = (0.5, 0.5, 0.5, 0.5, 0.5)$ to avoid degenerate training behaviour. If the standard deviation was increased to 1.0, divergent behaviour was observed in $\frac{8}{30}$ training instances and for values greater than 2.0, we saw divergence frequency greater than 0.5.

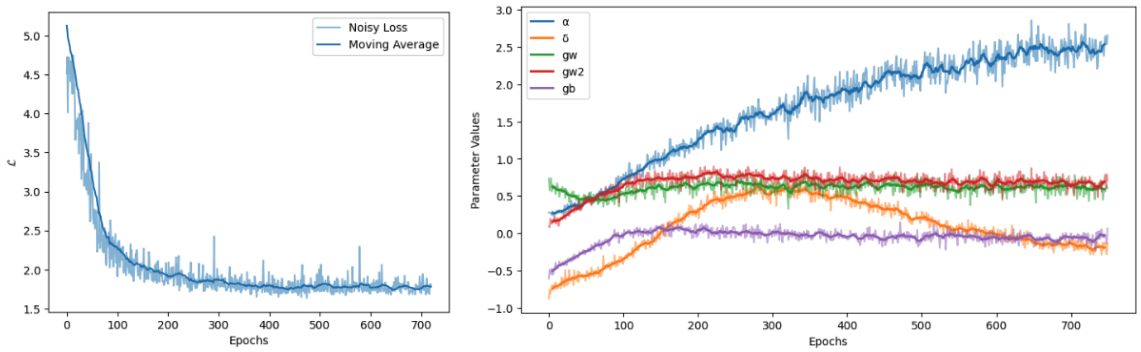
The reason for this can be traced back to a key observation made by Zhu et al. who in their CLSNA simulations saw that setting γ_w^1, γ_w^2 too large often did lead to divergent behaviour, indicating 'an interesting balancing relation between different parameters of the model' [4]. The fact that our neural calibration was so sensitive to initial prior variance is in contrast to the original MCMC calibration model used by Zhu et al. where they report consistent calibration convergence with priors of $\mathcal{N}(0, 100)$ for α, δ, $\mathcal{N}(0.5, 100)$ for γ_w^1, γ_w^2, and $\mathcal{N}(-0.5, 100)$ for γ_b. Keeping the variances high means they could keep the prior beliefs uninformative – a desirable property in objective Bayesian statistics to enforce minimal influence on the posterior [52]. Determining the root cause of this divergent behaviour and mitigation solutions is left for further research.

As shown in Figure 4.5, the loss curves for both the **207** and **616** models are akin to typical ML loss curves. The parameter evolution plots indicate that the parameters tend to stabilise as training progresses, and their trends are relatively consistent across different training instances. More plots for all model architectures can be found in Appendix B.2.

4.5.1 Obtaining Density Estimates

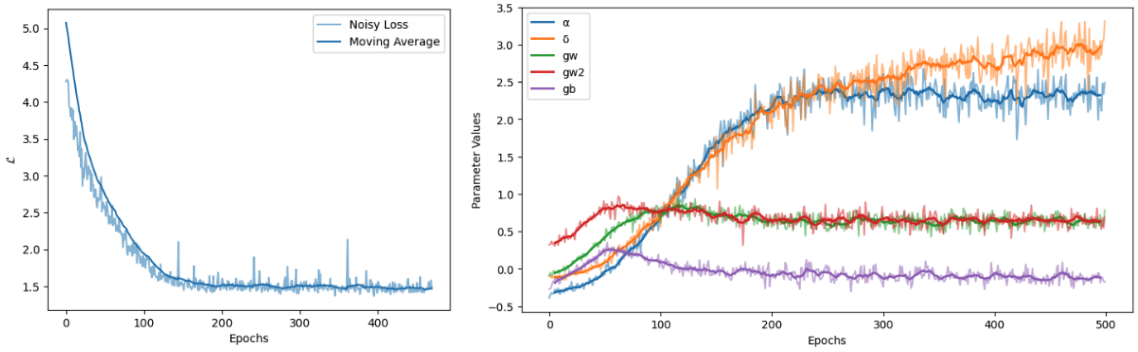
Density estimations can be obtained by running multiple training instances in parallel, storing loss and parameter values over epochs in each training run. Once complete, we compute the parameter density marginals by solving the discrete integral given by Equation 2.16. The quality of these density estimations inevitably improves with the number of trained models because of increased parameter space coverage. Therefore, there is a clear engineering trade-off between computational cost and estimation quality. In this report, all PDFs are computed from 30 training instances. This number was practical and yielded valid results.

The marginal densities in Figure 4.6 are smoothed with a 1D Gaussian kernel. We note that the modes correspond to the unique value of the parameter that minimises the loss when integrating out the other parameters. The mean is interpreted as the parameter value that, when considered



(a) Training Loss Curve for 207 Model

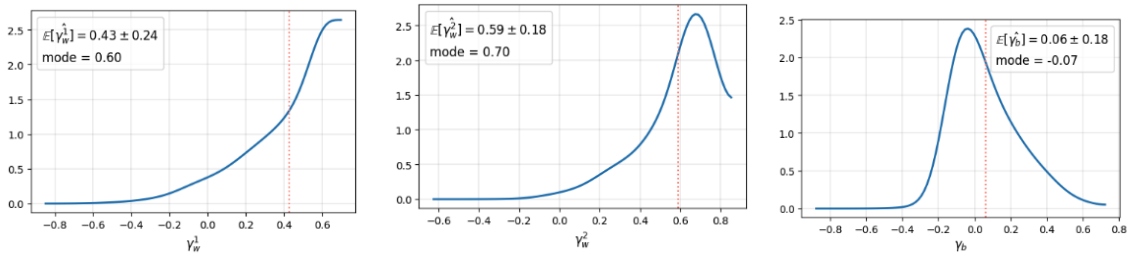
(b) Parameter Evolution in 207 training instance



(c) Training Loss Curve for 616 Model

(d) Parameter Evolution in 616 training instance

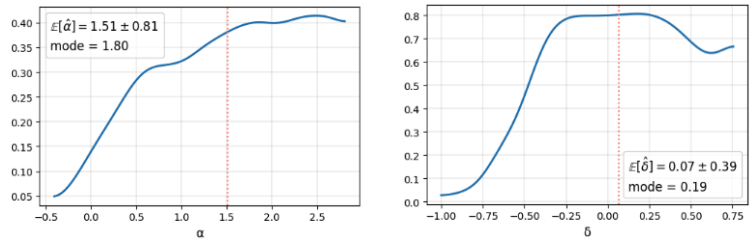
Figure 4.5: Single Training Instances of NN for both the **207** dataset (a,b) and **616** dataset (c, d). Darker lines correspond to moving averages.



(a) γ_w^1 Estimate

(b) γ_w^2 Estimate

(c) γ_b Estimate



(d) α Estimate

(e) δ Estimate

Figure 4.6: Parameter PDF estimates for 207 GNN model over 30 training instances. See Appendix B.3 for all model PDF estimates.

in conjunction with the other parameter means, yields a lower loss in expectation. Different parameter interactions will be investigated in Section 4.7.

Additionally, we observe a peculiar difference in distribution shape between the three parame-

ters $\{\gamma_w^1, \gamma_w^2, \gamma_b\}$ and $\{\alpha, \delta\}$. The variance of the latter two parameters is much higher than that of the first three. This will be investigated further in the sensitivity analysis conducted in Section 4.8. A higher variance can indicate that those parameters are not as influential in contributing to the loss value calculation. This difference in shape is consistent in both the **207** and **616** solutions and among all the model architectures implemented.

4.5.2 Bayesian Optimisation (HEBO)

In this section, we outline the hyperparameter sweep conducted using the Heteroscedastic Evolutionary Bayesian optimisation (HEBO) algorithm for Bayesian optimisation [53]. Detailed information about the HEBO algorithm can be found in Appendix C. We choose Bayesian optimisatio over grid search for hyperparameter tuning due to its efficiency in exploring the parameter space. Unlike grid search, which exhaustively evaluates all possible combinations of hyperparameters, Bayesian optimisation constructs a probabilistic model (in HEBO, a Gaussian process) of the objective function and uses it to select the most promising hyperparameters to evaluate. This approach is beneficial as training each model (particularly for the **616** dataset) involves expensive function evaluations, and Bayesian optimisation requires fewer iterations to find optimal or near-optimal hyperparameters.

A significant benefit of using HEBO is its capability for multi-objective optimisation, where the objective functions need not be differentiable. This is advantageous for our models as we optimise based on multiple criteria: BCE loss and F1 score to emphasise recall, and topology regularisation. The topology regularisation considers the absolute difference between the clustering coefficient and the number of edges in the ground truth versus the predicted adjacency matrices. Concretely, the composite loss function used in the HEBO optimisation is given by:

$$L_{\text{composite}} = L_{BCE} \times (1 - \text{F1}) + \omega \times L_{TOP} \tag{4.8}$$

where L_{BCE} is the weighted BCE loss as in 4.3. The topology regularisation term is:

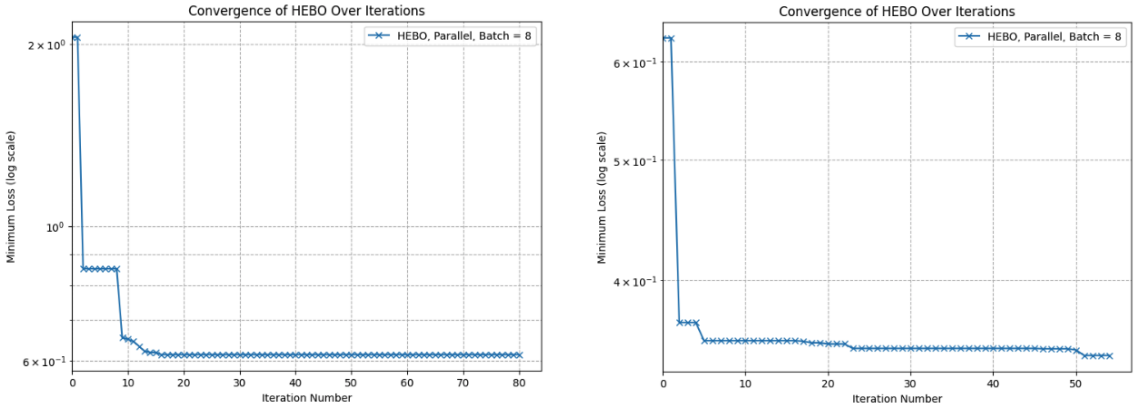
$$L_{TOP} = |C_{\text{avg}}(\hat{Y_{1:T}}) - C_{\text{avg}}(Y_{\text{GT}})| + \frac{|E(\hat{Y_{1:T}}) - E(Y_{\text{GT}})|}{E(Y_{\text{GT}})} \tag{4.9}$$

where C_{avg} denotes the average clustering coefficient and E denotes the number of edges, both computed for the predicted adjacency matrix $Y_{1:T}$ and the ground truth Y_{GT}.

Hyperparameter	**Type**	**Range**
num_layers	Integer	[3,5]
hidden_size	Integer	[16,128]
LR	Float	[1e-5,1e-3]
dropout_rate	Float	[0.0,0.5]
activation	Categorical	[Relu, Tanh, Sigmoid]
optimiser	Categorical	[SGD, Adam, RMSprop]
bce_weight	Float	[2.0,4.5]
weight_decay	Float	[1e-4,1e-1]
gamma	Float	[0.0,2.0]
beta	Float	[0.5,2.5]

Table 4.1: Parameter space configuration for HEBO search.

Due to the high computational cost, we only performed Bayesian optimisation for the basic NN models of both datasets. The hyperparameters for the CNN and GNN models were inferred based on the results from the **207** NN model. We executed the hyperparameter sweep on a Google Colab **Tesla T4 GPU** and took 4.23 hours for the **207** case, while the **616** NN HEBO took around 23.055 hours.



(a) HEBO Evaluations 207

(b) HEBO Evaluations 616

Figure 4.7: HEBO Evaluation Single-objective Pareto-Front for both 207 and 616 Datasets

4.6 Results

4.6.1 Parameter Estimates

Table 4.2 presents the parameter calibration point and variance estimates for various models applied to the **207** dataset. Specifically, the table includes results for the SGD and MCMC models from [4, 5] and the neural calibration NN, CNN, and GNN models. We obtain a few insights from the table which are elaborated on in Section 4.7.

The neural calibration methods (NN, CNN, GNN) exhibit higher variance than MCMC. While the SDs for the inter- (γ_w^1) and intra-party (γ_w^2) variables are similar between SGD and neural calibration, they are significantly larger for α and δ compared to SGD. This increased variability can be attributed to the indirect update mechanism in neural calibration, where model parameters θ of f_θ are updated instead of the estimates $\hat{\lambda}$. The flexibility of NNs to capture diverse behaviours in data also contributes to this variability.

Model	$\hat{\alpha}$	$\hat{\delta}$	$\hat{\gamma_w^1}$	$\hat{\gamma_w^2}$	$\hat{\gamma_b}$
SGD[2][5]	2.241 (0.158)	1.464 (0.156)	0.075 (0.174)	0.406 (0.206)	-0.182 (0.127)
MCMC[3][4]	2.809 (0.022)	1.500 (0.018)	0.493 (0.026)	0.105 (0.025)	-0.155 (0.014)
NN	2.011 (0.820)	-0.093 (0.791)	0.627 (0.335)	0.594 (0.231)	-0.172 (0.250)
CNN	1.672 (0.742)	0.013 (0.256)	0.612 (0.137)	0.705 (0.138)	-0.018 (0.131)
GNN	1.823 (0.811)	0.194 (0.392)	0.607 (0.244)	0.707 (0.187)	-0.075 (0.183)

Table 4.2: Parameter estimates and standard deviations (SD) for the **207** dataset.

The γ_b parameter for all three neural calibration models (NN, CNN, GNN) is negative, consistent with the SGD and MCMC models. This reflects a polarisation trend observed between 2010-2020 among the Democratic and Republican parties. For neural calibration models, γ_w^1 and γ_w^2 are much larger and similar in magnitude, unlike SGD and MCMC, which show different behaviours between parties. This uniformity in neural models may indicate a more consistent capture of inter- and intra-party dynamics. A significant difference is present with the δ parameter estimates. The δ parameter, representing edge persistence, is calibrated to small or even negative values in neural models compared to SGD and MCMC. Neural calibration appears to enforce defined behaviour in

[2]Trained for a total of 59K epochs (12K for point estimate calibrations 1 and 2, and 35K for variance estimation
[3]50K MCMC Iterations, burn-in of 15K samples

the latent space, reducing the need for edge persistence captured by δ. NNs may inherently capture higher-order interactions and temporal dynamics, relying less on the persistence parameter.

Model	$\hat{\alpha}$	$\hat{\delta}$	$\hat{\gamma_w^1}$	$\hat{\gamma_w^2}$	$\hat{\gamma_b}$
SGD[4][5]	3.223 (0.104)	1.085 (0.101)	-0.113 (0.115)	0.328 (0.138)	-0.214 (0.091)
NN	2.113 (0.743)	2.421 (1.091)	0.647 (0.252)	0.569 (0.238)	-0.152 (0.254)

Table 4.3: Parameter estimates and standard deviations (SD) for the **616** dataset.

Table 4.3 presents the parameter calibration results for the **616** dataset. Notably, the $\hat{\delta}$ parameter is much larger for the NN model than SGD, indicating a stronger emphasis on edge persistence. Unlike the SGD model, which has a negative $\hat{\gamma_w^1}$ value, the NN model exhibits positive inter-group ($\hat{\gamma_w^1}$) and negative intra-group ($\hat{\gamma_w^2}$) parameters, aligning with a true flocking behaviour.

4.6.2 Execution Time

One of the major benefits of neural calibration was achieving orders of magnitude faster calibration, which we observed in our results. During implementation, great care was taken to parallelise operations by executing them on GPU, often offering 7-10x latency reductions. In Table 5.3, we present the execution times for different models on the **207** dataset on both a **Tesla T4** GPU available on Google Colab and a **2 GHz Quad-Core Intel Core i5** CPU. From the results, it is clear that neural calibration methods (NN, CNN, GNN) achieve 2-3 orders of magnitude latency improvements compared to the original MCMC and a 1 order of magnitude reduction compared to SGD. For instance, while the MCMC model takes approximately 240 minutes on the CPU, the NN model completes the task in 7.95 minutes on the CPU for the **207** dataset; similarly, the CNN model achieves an average execution time of 0.896 minutes on the GPU compared to SGD's $\approx$ 10-minute latency.

Dataset	Model	GPU	CPU
207	**SGD**	<10	42.92 (1.208)
	MCMC	N/A	$\approx$ 240
	NN	1.173 (0.109)	7.95 (0.233)
	CNN	**0.896** (0.093)	2.72 (0.106)
	GNN	1.205 (0.098)	2.63 (0.160)
616	**SGD**	<20	78.89 (1.452)
	NN	**3.665** (0.138)	42.79 (0.528)

Table 4.4: Execution times reported in **minutes**. Standard deviations (SD), when available, are listed in parentheses. Blue highlights the best GPU training time in each dataset, respectively.

4.6.3 Evaluation Metrics

Table 4.5 presents the performance metrics and standard deviations (SD) for various models on the **207** dataset. The Δ metrics are the differences from ground truth values over the last five timesteps. For instance, Δ clustering coefficient represents the difference in average clustering coefficient between the ground truth and predicted values for the last five timesteps.

[4]Trained for a total of 16K epochs (3K for point estimate calibrations 1 and 2, and 10K for variance estimation

Metric	MCMC	SGD	NN	CNN	GNN
Weighted BCE$_\ell$	1.441 (0.058)	1.017 (0.043)	0.719 (0.043)	**0.634** (0.020)	*0.647* (0.026)
Precision$_h$	0.190 (0.006)	**0.271** (0.015)	0.252 (0.014)	0.261 (0.006)	*0.265* (0.008)
Recall$_h$	0.209 (0.013)	0.256 (0.028)	0.272 (0.029)	*0.357* (0.015)	**0.359** (0.019)
F1$_h$	0.199 (0.007)	0.263 (0.021)	0.261 (0.010)	*0.301* (0.006)	**0.304** (0.007)
AUC$_h$	0.433 (0.010)	0.509 (0.018)	0.576 (0.023)	*0.610* (0.009)	**0.613** (0.013)
Δ Clustering Coeff$_\ell$	**0.117** (0.013)	*0.195* (0.017)	0.253 (0.018)	0.203 (0.012)	0.199 (0.021)
Δ Degree Centrality$_\ell$	0.235 (0.026)	0.247 (0.036)	0.125 (0.065)	*0.107* (0.034)	**0.097** (0.047)
Δ Closeness Centrality$_\ell$	*0.045* (0.009)	**0.037** (0.009)	0.134 (0.022)	0.188 (0.010)	0.183 (0.016)

Table 4.5: Performance metrics and standard deviations (SD) for the **207** dataset. Values were calculated from 100 independent simulations. Superscripts $_h$ and $_\ell$ indicate whether higher or lower values are better for the metric. The best and second-best performances are in **bold** and *italics*, respectively.

Since the model is sampling-based and co-evolves with the latent space, traditional binary classification metrics like precision, recall, F1, and AUC are uncharacteristically low (reaching a max AUC of 0.613, which is marginally better than guessing). These numbers should be used relatively and **not** as absolute quality benchmarks. Our end goal is to understand behaviour from data and not necessarily infer node connections. Therefore, the metrics we are more interested in are the last three, which measure the topological structure of the matrices and general relationships between nodes.

Metric	SGD	NN
Weighted BCE$_\ell$	1.354 (0.029)	**0.750** (0.014)
Precision$_h$	0.327 (0.007)	**0.367** (0.004)
Recall$_h$	0.242 (0.010)	**0.507** (0.016)
F1$_h$	0.278 (0.008)	**0.426** (0.006)
AUC$_h$	0.556 (0.007)	**0.667** (0.007)
Δ Clustering Coeff.$_\ell$	0.315 (0.009)	**0.033** (0.014)
Δ Degree Centrality$_\ell$	0.295 (0.037)	**0.257** (0.036)
Δ Closeness Centrality$_\ell$	**0.016** (0.008)	0.120 (0.010)

Table 4.6: Performance metrics and standard deviations (SD) for the **616** dataset. Values computed from 100 independent simulations. Superscripts $_h$ and $_\ell$ indicate whether higher or lower values are better for the metric. The better metric values are in **bold**.

Neural calibration models perform significantly better on BCE and traditional classification metrics due to the training loss function. For instance, CNN and GNN models exhibit superior performance on these metrics, indicating their effectiveness in capturing the underlying data distribution. Regarding the last three metrics, which measure topological structure, the performance of MCMC, SGD, NN, CNN, and GNN is comparable. For example, the clustering coefficient is marginally

better for SGD (0.195) than GNN (0.199). Under the null hypothesis that these means are equal, the t-statistic p-value is 0.140, thus we don't have sufficient evidence to reject at even the 10%-level. We observe statistically significant improvement in Δ closeness centrality between **MCMC**/**SGD** and neural calibration models.

Table 4.6 presents the same performance metrics for the **616** dataset. The NN neural calibration model shows superior performance across almost all metrics compared to the SGD model, except Δ closeness centrality. Importantly, the NN model significantly outperforms SGD in Δ clustering coefficient, in contrast to the smaller dataset case. This improvement is statistically significant with a t-statistic p-value < 0.001. Highlighting that in the **616** calibration, the NN model predicted a δ point estimate of 2.421, this may indicate a relationship between mirroring ground truth topological structure and having high edge-persistence. This was not learned in the lower-capacity **207** neural architectures.

4.6.4 Latent Space Behaviour

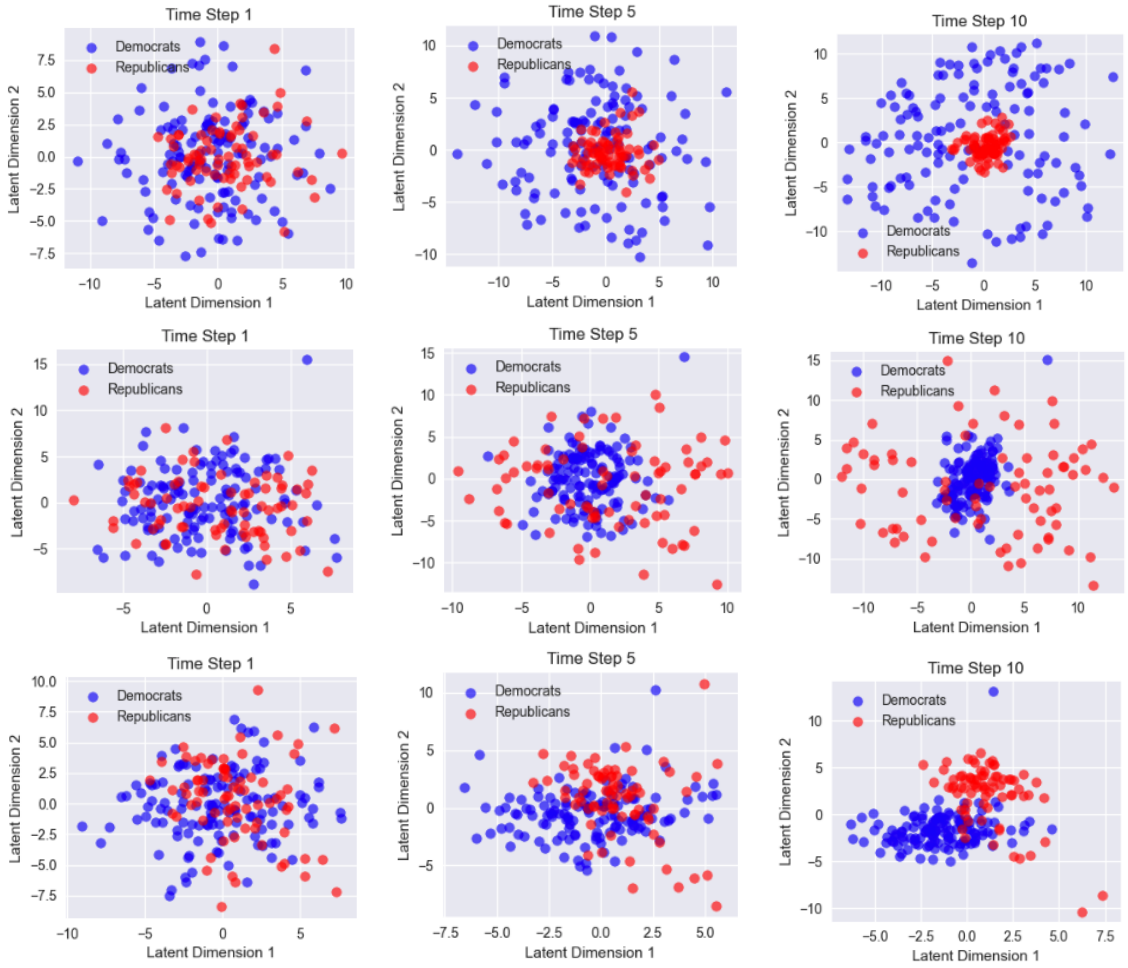


Figure 4.8: Latent space evolution at time steps $T = \{1, 5, 10\}$ for **MCMC** (Row 1), **SGD** (Row 2), and **NN** (Row 3) for **207** dataset. See Appendix B.4 for additional latent space plots.

Figure 4.8 shows the latent space evolution at 3 discrete time steps $T = \{1, 5, 10\}$ for MCMC, SGD, and NN on the **207** dataset. In the neural calibration implementation discussed in Section 4.4, there is no explicit regularisation of the latent space; instead, it is given free rein to evolve. In contrast, the SGD and MCMC models have regularisation terms in their loss functions to guide the latent space evolution. For example, in the SGD approach, the loss function penalises the distance of node latent positions from the mean position of their party members over all time steps $t > 1 \in \mathcal{T}$.

Examining the results, we observe that in the MCMC and SGD models, there is clear consensus formation within a single party, while the other party retains a persistent noisy latent space representation. In contrast, the NN method exhibits clear consensus formation within both parties, with a distinct two-group clustering by time step 10.

4.7 Discussion

Neural calibration proved to be a **significantly** faster calibration method, capable of producing parameter estimates with promising evaluation metric results. It is important to note that the evaluation metrics were not used for training (except for weighted BCE in {NN, CNN, and GNN}) and provide a fair basis for model comparison.

For the **207** dataset, the neural calibration methods, including the GNN, were **not** as effective in capturing topological structure as the MCMC model, although they were similar to the SGD model. There are several explanations for this. First, the architectures may have limited capacity due to lower parameter counts than the **616** NN counterpart. Second, the dataset itself is limited in size, providing fewer data points for the models to learn from. Lastly, the objective function we used was engineered, not theoretically derived. Although we incorporated L2 and persistence regularisation, that weighted BCE loss may have had a disproportionately large influence. It is clear that the neural calibration models are learning what we teach them, and thus tweaking the loss has a high potential to yield the topological behaviour we expect.

The principal reasons topological measures were not directly incorporated into the loss function are non-differentiability and high computational cost that would slow down training substantially. For example, the state-of-the-art average clustering coefficient algorithm has complexity $\mathcal{O}(N \times d_{max}^2) \subseteq \mathcal{O}(N^3)$ where d_{max} is the vertex of the graph with the longest adjacency list [54]. For the **207** dataset, calculating the clustering coefficient takes around 2 seconds for the five last timestamp matrices and shoots up to 35 seconds for the **616** graphs. This adds too much overhead to justify incorporation into the loss function.

In the **616** dataset, the results were promising. The NN model outperformed the SGD model in all metrics and demonstrated significantly better performance on the Δ clustering coefficient metric, all the while calibrating approximately six times faster than the SGD counterpart. Given the observed trend of improvement from NN to GNN in the **207** dataset, developing a sophisticated ML model that incorporates the adjacency matrix to variable-node tensors could yield even better outcomes.

4.7.1 Model Output Evaluation

Dataset	Model	Average Sparsity (%)	Sparsity Last 5 (%)
207	**GT Truth**	78.82	67.15
	SGD	83.38	83.87
	GNN	73.32	67.22
616	**GT**	76.56	66.65
	SGD	80.02	81.46
	NN	62.84	53.97

Table 4.7: Sparsity statistics of predicted adjacency matrices for different models and both datasets.

Figure 4.9 shows the final adjacency matrices at time step $t = 10$ for both datasets. The ground truth matrices display distinct horizontal and vertical patterns, indicating structured relationships between nodes. These patterns are crucial to replicate for accurate modelling. Table 4.7 summarises the sparsity information for the **207** and **616** datasets.

For the **207** dataset, the GNN model output appears noisy with only faint horizontal and vertical striations upon close inspection. In contrast, the SGD matrix, although more sparse, shows patterns similar to the ground truth. However, the SGD matrix is excessively zeroed out with an

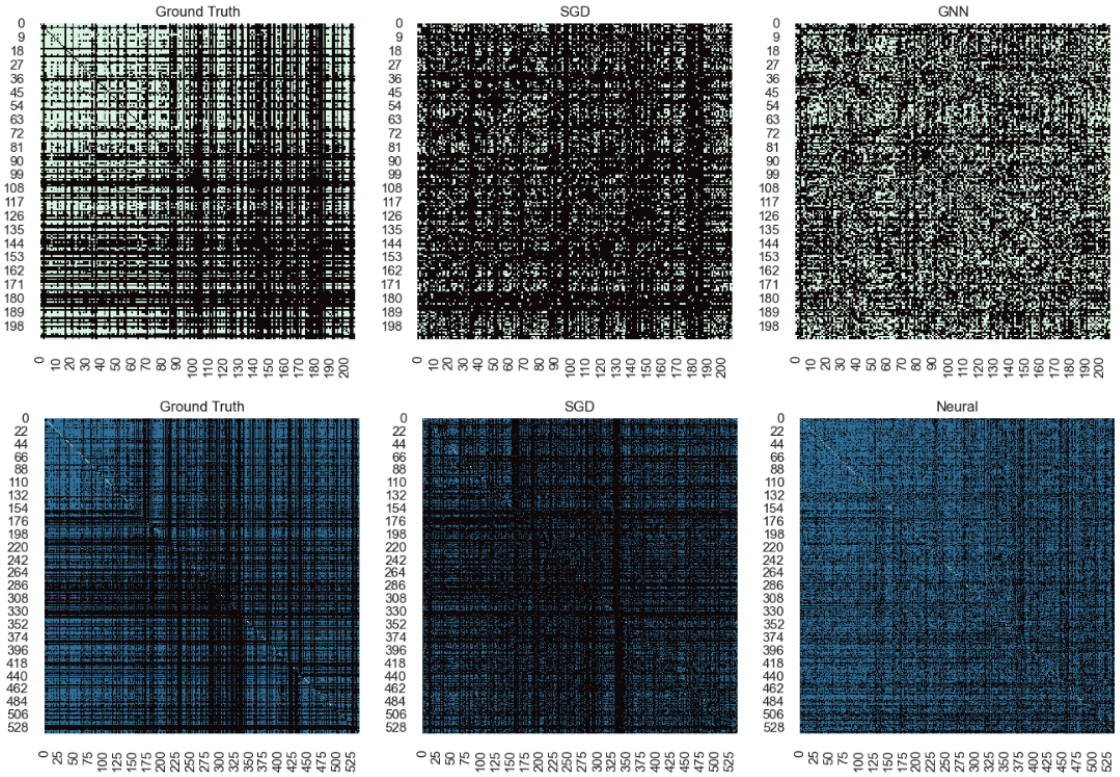


Figure 4.9: **207** (Row 1) and **616** (Row 2) Adjacency Matrix heat-maps for Ground Truth, SGD, and GNN/NN at timestep $t = 10$.

average sparsity of 83.38% over all time steps and 83.87% over the last five time steps, compared to the ground truth's 78.82% and 67.15%, respectively. The GNN model, with 73.32% average sparsity over all time steps and 67.22% over the last 5, aligns more closely with the observed data.

In the **616** dataset, the NN matrix exhibits clear horizontal and vertical lines similar to the ground truth, reflected in the metrics in Table 4.6. However, the sparsity behaviour is less ideal. The ground truth shows an average sparsity of 76.56% over all time steps, dropping to 66.65% for the final five matrices. The SGD model has an overall average sparsity of 80.02%, increasing to 81.46% over the last 5, while the NN model predicts far too many connections between nodes in the final time steps.

4.8 CLSNA Sensitivity Analysis

The objective of this section is to identify the parameters of CLSNA that contribute significantly to simulation metrics performance and relate these insights to what was observed in Section 4.6 (Results). To do so, three methods of SA are employed: Morris method, Sobol's Indices method, and MOO NSGA-II (see Section 3.4 for details). It is important to note that SA is model-agnostic as it only examines CLSNA simulations for a given parameter configuration input, and thus serves as a macro-analysis tool we can use to comment on all the models discussed thus far.

In regards to implementation, we utilised the SALib library [55, 56] and the cuDF RAPIDS Python GPU library [57]. RAPIDS enabled the parallelisation of NetworkX topological metric calculations (i.e. closeness centrality), substantially reducing computation time. For instance, the variance-based method executed in roughly 6 hours for the **616** dataset, compared to over 30 hours using an i5 CPU.

4.8.1 Objective Function

The key steps of performing an SA are (1) formulating an objective function that evaluates the model for each input and (2) defining a parameter search space. As with HEBO hyperparameter sweeps (4.5.2), the objective does not need to be differentiable. The function we used for the SA is defined as follows:

$$O_{\text{composite}} = O_{BCE} + \kappa \times O_{\text{centrality}} + \beta \times O_{\text{row\_sum}} \tag{4.10}$$

where O_{BCE} is the weighted BCE loss, $O_{cenrality}$ the difference in average between centrality between predicted output and ground-truth, and O_{row_sum} formally defined as:

$$O_{\text{row\_sum}} = \left| \frac{1}{5} \sum_{t=T-4}^{T} \left(\mathbf{1}^T \hat{Y}_t - \mathbf{1}^T Y_{\text{GT},t} \right) \right| \tag{4.11}$$

where $\mathbf{1}$ is a column vector of ones, $\hat{Y}_t$ is the predicted adjacency matrix at time t, and $Y_{\text{GT},t}$ is the ground truth adjacency matrix at time t. This was included to capture topological patterns in ground truth (GT) data. The coefficients $\kappa = 5$ and $\beta = 0.03$ were determined by preliminary experiments to ensure equal contribution of each component of 4.10 to $O_{\text{composite}}$.

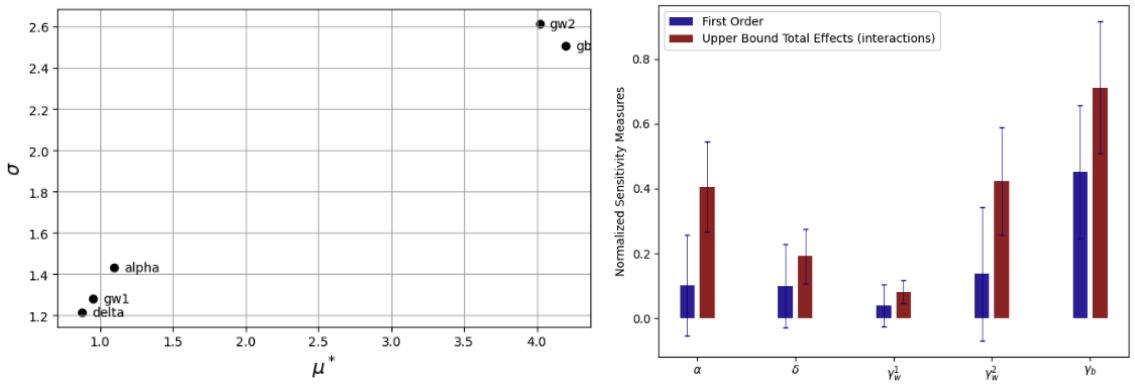
The parameter range for each parameter, λ_i, was defined as $\lambda_i^R = (\mu_{\hat{\lambda}_i} - 5\sigma_{\hat{\lambda}_i}, \mu_{\hat{\lambda}_i} + 5\sigma_{\hat{\lambda}_i})$ where $(\mu_{\hat{\lambda}_i}, \sigma_{\hat{\lambda}_i})$ were parameter and variance estimates from the **207** NN model (see Table 4.2), but this choice was arbitrary. Any reasonable range of parameters suffices.

4.8.2 Results: 207

Morris and Sobol Method

Figure 4.10 shows the results of the sensitivity analysis for the **207** dataset using the Morris and Sobol methods. The graph on the left is a classical Morris method plot, which indicates the mean of the absolute elementary effects (μ^*) on the x-axis and the standard deviation (σ) on the y-axis. The plot clearly highlights γ_b and γ_w^2 as the most influential parameters, given their high μ^* and σ values.

The graph on the right presents the Sobol indices, with the first-order effects shown in blue and the total effects (including interactions) shown in red. The Sobol analysis confirms the findings from the Morris method, showing that γ_b and γ_w^2 are the most influential parameters in terms of first-order effects. Interestingly, the Sobol indices also reveal that α has a significant impact when considering higher-order interactions, suggesting its influence is nuanced.

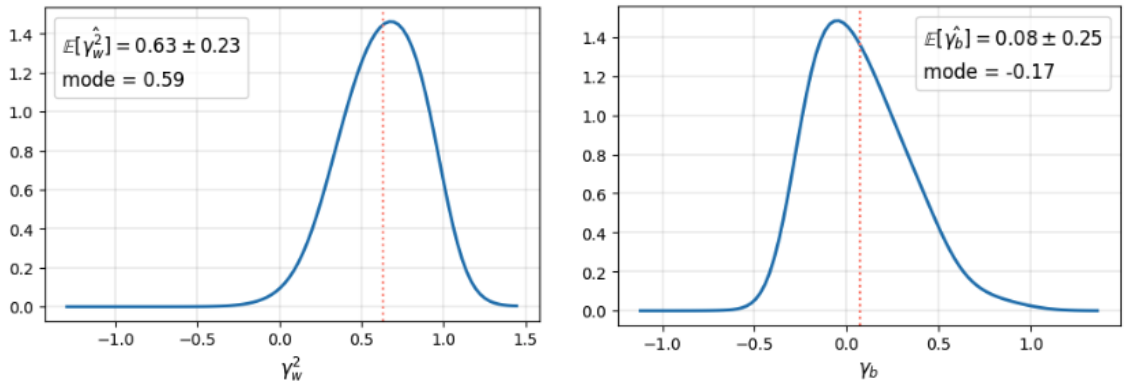


(a) Mean of Absolute EEs

(b) Sobol Indices

Figure 4.10: Morris and Sobol Sensitivity Analysis for **207** dataset

The key insight the figures in 4.10 provide is when they are analysed in conjunction with the PDF parameter estimates the neural calibration method gives us. Figure 4.11 plots the PDFs of γ_w^2 and γ_b for the basic **207** NN model. The shape of these distributions is unimodal, slightly skewed, and tight around the mode (i.e. low SD).



(a) γ_w^2 Estimate

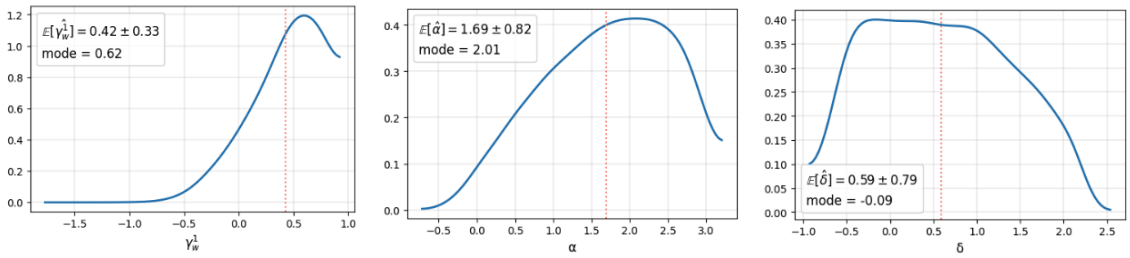
(b) γ_b Estimate

Figure 4.11: Parameter PDF estimates for γ_w^2 and γ_b in 207 NN model.

When contrasted with the PDFs of Figure 4.12 for the remaining three parameters, they have larger variances and do not necessarily exhibit unimodality. In essence, the SA and neural calibration PDFs convey similar information by providing a relationship between low-density variance and high Morris/first-order Sobol significance.

The logical conclusion would be to consider neural calibration as a general framework for sensitivity analysis (SA); in reality, neural calibration follows many of the steps of a traditional SA in that it evaluates a loss function at (ideally) multiple values of the underlying parameter space. The rigorous justification of neural calibration as an SA tool is not explored further in this project and is left for future work.

The PDF shapes we see for the **207** NN model are consistent across all the PDF neural calibrations produced for all the different models. The only exception is the PDF estimates of γ_w^1 in the CNN model, where we obtain a strikingly similar density shape to γ_w^2. Of course, observing these 1D marginals individually does not allow us to verify the higher-order interaction significance of α identified by the Sobol indices in Figure 4.10, so we consider an alternative approach.



(a) γ_w^1 Estimate

(b) α Estimate

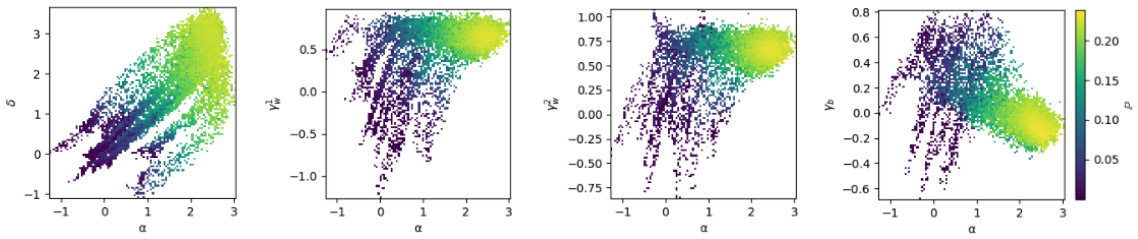
(c) δ Estimate

Figure 4.12: Parameter PDF estimates for γ_w^2 and γ_b in the 207 NN model.

One way to quantify the higher-order interactions of α is to assess 2D parameter joint distributions, plotted in Figure 4.13. These are obtained in a similar way to the 1D marginals. As a 2D analogue of standard deviation, the weighted Standard Distance Deviation (SDD) quantifies the dispersion of these joint distributions by calculating an unbiased SD estimator of the Euclidean distance from each point to the probability-weighted mean centre [58]. Formally, for parameters X and Y:

$$SDD_{XY}(\mathcal{J}) = \sqrt{\frac{\sum \mathcal{J}(X,Y)_i d_{imc}^2}{(\sum \mathcal{J}(X,Y)_i - 2}} \tag{4.12}$$

where $\mathcal{J}$ is the joint distribution of (X, Y) and d_{imc} is the distance from the point to the weighted mean centre. Both summations are over all points of $\mathcal{J}$.



(a) (δ, α) (b) (γ_w^1, α) (c) (γ_w^2, α) (d) (γ_b, α)

Figure 4.13: 2D Joint Density estimates for all combinations of α in the 207 NN model. See Appendix B.5 for plots on the 207 dataset.

For the joint densities in Figure 4.13, the weighted SDDs for $(\alpha, \delta) = 1.153$, $(\alpha, \gamma_w^1) = 0.814$, $(\alpha, \gamma_w^2) = 0.806$, and $(\alpha, \gamma_b) = 0.784$. These values suggest that the interactions between α and the three inter-/intra-party parameters are similarly co-significant. This indicates that both α and γ_w^1 may be more important than the first-order Sobol indices suggest when considered jointly.

MOO and Pareto Front

The motivation for implementing MOO with NSGA-II is the observation that during the training of the neural calibration models, there was constantly a trade-off between competing objectives of minimising BCE and aligning the topological structure of simulation output and ground truth. Constructing a Pareto front of non-dominated solutions was deemed a satisfactory solution to visualise this trade-off and propose a family of model calibration solutions. In effect, this SA framework also provides an alternative approach to the central task at hand: parameter estimation. For this task, the objective function was modified slightly to separate the topological and BCE components of the single-valued evaluation function 4.10. No other modification was made.

Figure 4.14 is a plot of the Pareto front, where the optimal solution is the point with the shortest Euclidean distance to the origin. The total execution time for the NSGA-II MOO was approximately 5 hours on a Tesla T4 GPU with optimised matrix operations. In other words, this is quite a computationally intensive and unrealistic alternative to neural calibration. Nevertheless, we compute similar evaluation metrics to compare with the results in Section 4.6.

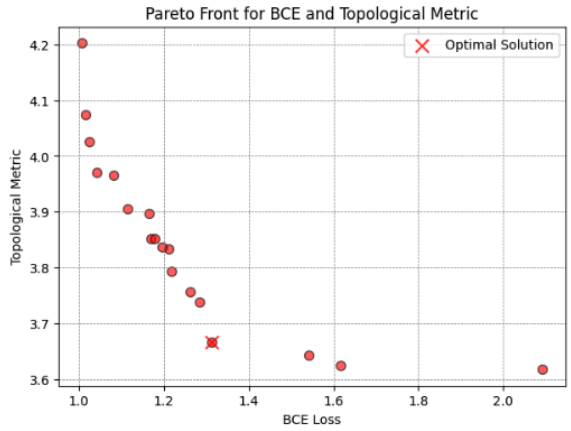


Figure 4.14: MOO Pareto Front using NSGA-II on **207** dataset

Model	$\hat{\alpha}$	$\hat{\delta}$	$\hat{\gamma_w^1}$	$\hat{\gamma_w^2}$	$\hat{\gamma_b}$
NSGA-II[5]	2.492	0.160	0.264	1.253	-0.057

Table 4.8: Parameter Configuration for Optimal Solution in Pareto Front for the **207** dataset.

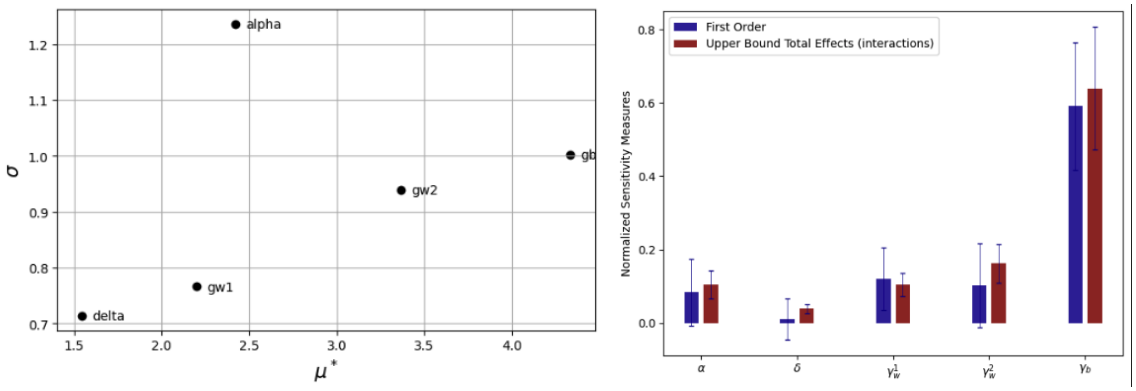
[5]Evolutionary Algorithm ran for 100 iterations with 10 offspring per generation.

Metric	NSGA-II	Best Value
Weighted BCE$_\ell$	0.778 (0.019)	**0.634** (0.020)
Precision$_h$	**0.271** (0.006)	**0.271** (0.015)
Recall$_h$	**0.485** (0.017)	0.359 (0.019)
F1$_h$	**0.347** (0.005)	0.304 (0.007)
AUC$_h$	**0.620** (0.008)	0.613 (0.013)
Δ Clustering Coeff.$_\ell$	**0.085** (0.013)	0.117 (0.013)
Δ Degree Centrality$_\ell$	0.178 (0.055)	**0.097** (0.047)
Δ Closeness Centrality$_\ell$	0.197 (0.018)	**0.037** (0.009)

Table 4.9: Performance metrics and standard deviations (SD) for the NSGA-II model compared to the best values from other models. Superscripts $_h$ and $_\ell$ indicate whether higher or lower values are better for the metric.

From the estimates in Table 4.8, we observe, yet again, polarisation behaviour given by positive inter-group $\{\gamma_w^1, \gamma_w^2\}$ values and negative γ_b. Additionally, we note that as in the SGD model, $\gamma_w^2 > \gamma_w^1$ suggests that both parties have exhibited distinct online behaviour from the data; this contrasts with the estimates from neural calibration. Regarding the evaluation metrics, Table 4.9 demonstrates that the NSGA-II model is effective. A new best is achieved in $\frac{5}{8}$ metrics and importantly reduces the Δ Clustering coefficient. Despite these results, the sheer computational cost of running a MOO NSGA-II outweighs any potential benefit from improved performance.

4.8.3 Results: 616

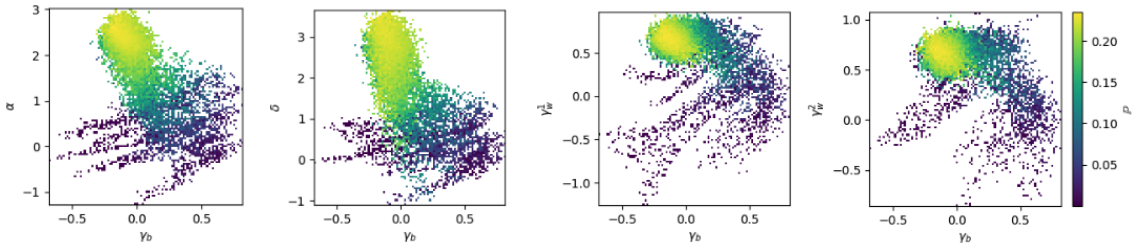


(a) Mean of Absolute EEs

(b) Sobol Indices

Figure 4.15: Morris and Sobol Sensitivity Analysis for **616** dataset

In (a) of Figure 4.15 we see that the same parameters $\{\gamma_w^2, \gamma_b\}$ are deemed significant; however, in contrast to the **207** dataset, we also observe that alpha has high absolute EE variance. Figure (b) tells a slightly different story, where γ_b has a much larger first and higher-order Sobol index value. Curiously, γ_w^2 has a lower first-order Sobol index than γ_w^1, albeit having the second largest upper bound total effects index. Overall, the Morris and Sobol methods inform us that γ_b is significant when modelling the **616** dataset.



(a) (γ_b, α) (b) (γ_b, δ) (c) (γ_b, γ_w^1) (d) (γ_b, γ_w^2)

Figure 4.16: 2D Joint Density estimates for all combinations of γ_b in 616 NN model.

Due to the large total effects index of γ_b, we assess the 2D joint distributions of γ_b to understand how this parameter interacts with other parameters in the dynamical system. Visually, we observe that the joint distributions (γ_b, γ_w^1) and (γ_b, γ_w^2) exhibit a 'tight' dispersion property. Concretely, the weighted SDD for $(\gamma_b, \alpha) = 0.784$, $(\gamma_b, \delta) = 1.010$, $(\gamma_b, \gamma_w^1) = 0.342$, and $(\gamma_b, \gamma_w^2) = 0.342$. From this, we determine that the co-interactions of γ_b with the inter-group parameters induce a higher variance of model output than with α and δ. Moreover, the SDD(γ_b, α) < SDD(γ_b, δ), reflecting what we observe in (b) of Figure 4.15 where α has a non-negligible interaction index. For the **616** dataset, we do **not** conduct a MOO NSGA-II due to computational limitations.

4.9 Concluding Remarks

In this chapter, we have discussed the various components of a successful neural parameter calibration for the CLSNA model. Additionally, we have presented results for many ML architectures trained on the **207** and **616** datasets, concluding that the neural calibration methods are significantly quicker to calibrate than their MCMC and SGD counterparts, all the while providing comparable results when evaluated on various classification and topological metrics. The neural calibration models performed especially well in the variable node (616) case.

We have made the critical observation that despite the number of learnable parameters growing by $input\_size^2$ in the NN case, this did not massively impact training time. The bottleneck in the calibration scheme was predominately in the numerical solver code. With our CNN and GNN implementations, we eliminate the parameter dependency on input size by leveraging the weight-sharing property these architectures have.

In our SA, we have identified γ_w^2 and γ_b as significant parameters to output variance and highlight a neat equivalence observed between Morris/Sobol significance and variance estimates we obtained for parameter PDFs with neural calibration. Computing the Morris, Sobol, and MOO analysis proved a computationally expensive task, so determining parameter significance with a confidence-interval approach we get for free as a by-product of neural calibration could be considered an expedient alternative.

Chapter 5

Neural Calibration as a VAE

In this chapter, we present a reinterpretation of neural calibration as a variational auto-encoder (VAE). Under this Bayesian framework, we consider neural calibration as a tool for updating prior beliefs on system dynamics with observed data. By implementing it in this way, the machine learning model outputs both point and variance estimates, opposed to approximating it empirically with loss values as previously done.

The rest of this section is structured as follows: We start with the motivation for reparameterising neural calibration models, including a detailed discussion on activation function saturation. Next, we provide the mathematical preliminaries on probabilistic models, directed graphical models, and VAEs. Finally, we present a proof-of-concept by applying neural calibration to CLSNA and compare the results with those obtained in Chapter 4.

5.1 Motivation

The initial motivation for reformulating the VAE came from challenges we faced training the neural calibration models. These challenges are common to VAE tuning [59, 60]. Specifically, we observed the following issues:

- **Vanishing Gradient Problem**: While training the neural calibration models, the gradients often diminished, leading to slow convergence and, in some cases, failure to converge. This issue was particularly notorious in the NN trained on the 616 dataset.
- **High Sensitivity to Parameter Initialisation**: The models exhibited significant sensitivity to the initialisation of parameters. When the prior values of the auxiliary neural network had a variance greater than 2, model divergence was more frequent than convergence. This high sensitivity ultimately made the training process unstable and difficult to reproduce.
- **Low Learning Rate Requirement**: Ensuring stable training required a very low learning rate (1e-5 for CNN/GNN) to avoid pathological solutions. Consequently, this made convergence slower as more epochs were necessary.
- **Degenerate Solutions:** Without introducing L_2 regularisation with parameter priors, the ML models often resorted to the degenerate solution of predicting predominately 0's for the adjacency matrices. This minimised the average BCE loss due to the sparsity of the observed $Y_{1:T}$. Incorporating prior beliefs avoided posterior collapse – the phenomenon where uninformative or non-existent priors [61].

These challenges are typical of the complexities involved in VAE implementations. Additionally, we made a key insight that by *uncoiling* the circular calibration method diagram in Figure 2.3, we effectively encode the observed data into a high-dimensional parameter space, which is passed into a non-convex numerical solver. By conceptualising this solver as a decoder with **no** learnable parameters, we essentially construct a VAE.

5.2 Preliminaries

In what follows, we present the mathematical theory for VAEs inspired by Kingma and Wellings's seminal article "An Introduction to Variational Autoencoders" [62]. In parallel to the derivations, we will repeatedly connect the material to the specific implementation of the CLSNA model.

5.2.1 Unsupervised Representation Learning

When performing parameter calibration and engaging in generative machine learning tasks, one is interested in learning probabilistic models from data, a task known as unsupervised representation learning.

Defining $\mathbf{x}$ to be a vector of observed variables, our goal is to learn its joint distribution by obtaining 'disentangled, semantically independent and causal factors of variation in data' [63]. The core assumption is that the dataset $\mathbf{x}$ represents a sample from an underlying process whose distribution, $p^*(\mathbf{x})$, is unknown. Our goal is to approximate the latent process with a model $p_\theta(\mathbf{x}$, parametrised by the set θ, such that $p^*(\mathbf{x}) \approx p_\theta(\mathbf{x})$. For CLSNA, $\mathbf{x}$ corresponds to the observed time-series adjacency matrices $Y_{1:T}$. As the distribution $p^*(\mathbf{x})$ is unknown, approximating it becomes an engineering task that presents a trade-off: maximising the discriminative power of the model $p_\theta(\mathbf{x})$ to match the data and incorporation of prior beliefs on the distribution p^*.

5.2.2 Directed Latent Variable Models (DLVM)

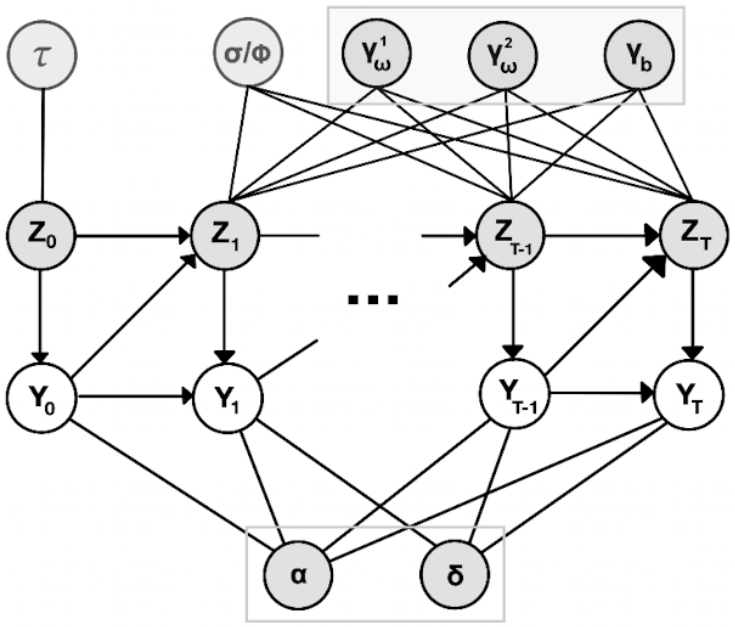


Figure 5.1: Latent directed graphical model (DGM) of CLSNA model. The latent variables learnt by VAE are depicted in the grey rectangles.

Directed Graphical Models (DGMs), also known as Bayesian networks, are probabilistic models that use directed acyclic graphs (DAGs) to represent conditional dependencies between variables [64]. Each node in a DGM represents a random variable, and each directed edge introduces a directed dependency between the nodes. The joint probability distribution of the variables is factorised according to the graph structure.

Specific to the VAE framework, we will work with *directed* latent variable probabilistic models (DLVMs). Defining the set of latent variables to be $\mathbf{z}$, our joint distribution model is of the form $p_\theta(\mathbf{x}, \mathbf{z})$. Conceptually, DLVMs are 'probabilistic models where all the variables are organised into

a directed acyclic graph' [62] whose distributions are parameterised by machine learning architectures (i.e. NNs). More concretely, for $\mathbf{x} := \{x_1, \ldots, x_T\}$ and $\mathbf{z} := \{z_1, \ldots, z_M\}$, $p_\theta(\mathbf{x}, \mathbf{z})$ is factorised as follows:

$$p_\theta(\mathbf{x}, \mathbf{z}) = \prod_{j=1}^{T} p_\theta(x_j | \text{Pa}(x_j)) \times \prod_{k=1}^{M} p_\theta(z_j | \text{Pa}(z_k))$$

where $\text{Pa}(x_j)$ and $\text{Pa}(z_k)$ are the sets of parent variables for variables x_j and z_k. With DLVMs, these conditional distributions are given by:

$$\zeta = \text{MLModel}(\text{Pa}(m))$$

$$p_\theta(m | \text{Pa}(m)) = p_\theta(m | \zeta)$$

where $m \in \{x_1, \ldots, x_T, z_1, \ldots, z_M\}$. In the CLSNA case, $\mathbf{z} = \lambda = \{\alpha, \delta, \gamma_w^1, \gamma_w^2, \gamma_b\}$, i.e. the parameters we aim to estimate. It is **important** to note that the opinion latent space is not considered part of the set $\mathbf{z}$ because we consider it as part of the `simulate_clsna` dynamics. Figure 5.1 is the CLSNA DGM, from which we can immediately see that all the latent variables in $\mathbf{z}$ are parent nodes and hence have empty parent sets.

Remembering our goal of approximating the marginal distribution of $\mathbf{x}$ with $p_\theta(\mathbf{x})$, we note:

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z} \tag{5.1}$$

where we are integrating over the entire latent space. It is in the computation of the integral where issues arise. Given a dataset $\mathcal{D}$, performing maximum likelihood learning is typically intractable. This is because the posterior $p_\theta(\mathbf{z}|\mathbf{x})$ requires computing $p_\theta(\mathbf{x}, \mathbf{z})$ for every single configuration of $\mathbf{z}$ within the support of the priors on the latent space $p_\theta(\mathbf{z})$, which are typically uncountable spaces (i.e. $\mathbf{z} \in \mathbb{R}^M$).

5.2.3 VAE

Variational Autoencoders (VAEs) can be viewed as two coupled models: an **encoder** or recognition model, denoted $q_\phi(\mathbf{z}|\mathbf{x})$, and a **decoder** or generative model, $p_\theta(\mathbf{x}|\mathbf{z})$. The encoder and decoder are independently parameterised and are jointly optimised using gradient-descent optimisation. Under Bayes' rule, these models are interpreted as inverses of one another, where we transform observed data $\mathbf{x}$ to a simple and typically lower-dimensional latent space and back during reconstruction.

The purpose of introducing the coupled model framework is to make the integral of the DLVM posterior in equation 5.1 tractable. The recognition model is introduced as $q_\phi(\mathbf{z}|\mathbf{x})$ where ϕ denotes the set of learnable parameters. Like a DLVM, q_ϕ is parameterised using deep ML networks. Moreover, when defining q_ϕ, we must decide on the form of the posterior distribution, which in most use cases is a multivariate Gaussian. This is a rather strong constraint and will be addressed in the evaluation of the CLSNA VAE.

For the CLSNA model, we use factorised Gaussian posteriors, and our recognition model is therefore $q_\phi(\mathbf{z}|\mathbf{Y_{1:T}}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2))$ with:

$$(\boldsymbol{\mu}, \log \boldsymbol{\sigma}) = \text{EncoderMLModel}_\phi(\mathbf{Y_{1:T}}) \tag{5.2}$$

$$q_\phi(\mathbf{z}|\mathbf{Y_{1:T}}) = \prod_i q_\phi(z_i|\mathbf{Y_{1:T}}) = \prod_i \mathcal{N}(z_i; \mu_i, \sigma_i^2) \tag{5.3}$$

where we explore three different models in our implementation (NN, CNN, GNN). The CLSNA generative model, $p_\Phi(\mathbf{Y_{1:T}}, \mathbf{z})$, is unique in that it has no learnable parameters (i.e., $\Phi = \emptyset$) and is just the numerical solver: `simulate_clsna`. This implies that VAE learning is restricted to updating the encoder parameters.

A crucial aspect of VAEs is the prior distribution $p_\theta(\mathbf{z})$ over the latent variables. In general VAEs, this prior is often chosen to be a simple distribution such as a standard Gaussian, which

acts as a regulariser and ensures tractability of the objective function. For the CLSNA model, we use independent Gaussian priors with means $[2.5, 1.0, 0.5, 0.5, -1.0]$ and standard deviations of approximately $1/4$, inspired by results of Chapter 4.

Figure 5.2 is a diagram of the CLSNA VAE with a fully connected NN encoder. The modular VAE framework allows for swapping the NN architecture for any other ML model. The exact details of the encoder and decoder architectures are specified in Sections 5.4 and 5.5.

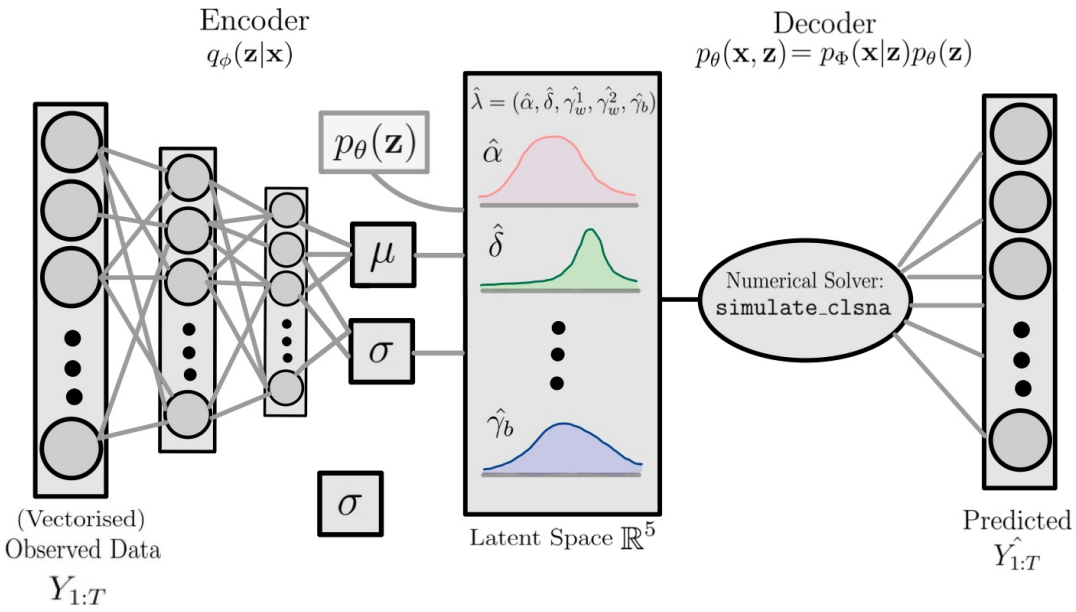


Figure 5.2: CLSNA VAE with vanilla NN encoder and `simulate_clsna` decoder for the **207** dataset. When training on the **616** dataset, the only change is using `simulate_clsna_dynamic` numerical solver.

5.2.4 VGAE

Similarly as the GNN neural calibration implementation of Section 4.3, we aim to leverage the topology induced by the partisan adjacency matrix to guide the training of our VAE. Variational Graph auto-encoders, or VGAEs, allow us to do so. Originally developed by Kipf and Welling, VGAEs are 'capable of learning interpretable latent representations for undirected graphs' [65] by parametrising the encoder model q_ϕ) with a GNN composed of deep GCN layers. For the CLSNA implementation, this implies the recognition model remains of the form $q_\phi(\mathbf{z}|\mathbf{Y_{1:T}}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2))$ with:

$$(\boldsymbol{\mu}, \log \boldsymbol{\sigma}) = \text{GNN}_\phi(\mathbf{Y_{1:T}}, \tilde{\mathbf{A}}) \tag{5.4}$$

$$q_\phi(\mathbf{z}|\mathbf{Y_{1:T}}) = \prod_i q_\phi(z_i|\mathbf{Y_{1:T}}) = \prod_i \mathcal{N}(z_i; \mu_i, \sigma_i^2) \tag{5.5}$$

where $\tilde{\mathbf{A}}$ is a $\{0,1\}^{N\times N}$ unweighted adjacency matrix defined identically as in 4.1.

5.2.5 ELBO and VAE Optimisation

Standard variational inference theory provides an evidence lower-bound, or ELBO, as an approximation to the log-likelihood of $p_\theta(\mathbf{x})$. For any recognition model $q_\phi(\mathbf{z}|\mathbf{x})$ satisfying $q_\phi(\mathbf{z}|\mathbf{x}) > 0$

whenever $p_\theta(\mathbf{z}|\mathbf{x}) > 0$:

$$\log p_\theta(\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x})\right] \tag{5.6}$$

$$= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x})}\right]\right] \tag{5.7}$$

$$= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_\theta(\mathbf{x},\mathbf{z})q_\phi(\mathbf{z}|\mathbf{x})}{q_\phi(\mathbf{z}|\mathbf{x})p_\theta(\mathbf{z}|\mathbf{x})}\right]\right] \tag{5.8}$$

$$= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]\right] + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\right]\right] \tag{5.9}$$

where the first term of (5.9) is the ELBO $\mathcal{L}_{\theta,\phi}(\mathbf{x})$ and the second is the non-negative KL-divergence between $q_\phi(\mathbf{z}|\mathbf{x})$ and the true posterior $p_\theta(\mathbf{z}|\mathbf{x})$, which is zero if and only if the distributions are equal. An immediate consequence of (5.9) is that the KL-divergence determines how tight the lower bound is, as it measures the gap between $\mathcal{L}_{\theta,\phi}(\mathbf{x})$ and $\log p_\theta(\mathbf{x})$ [46]. Another consequence is that optimisation of $\mathcal{L}_{\theta,\phi}(\mathbf{x})$ concerning the parameter sets θ and ϕ will both maximise the log-likelihood $\log p_\theta(\mathbf{x})$ **and** minimise the KL divergence between the recognition model and the true posterior. The latter minimisation is of sole interest in the CLSNA case, as we are performing parameter calibration where the distributions of the latent space will ultimately determine the method's evaluation. As previously mentioned, our decoder is **not** learnable, and we use the reconstruction loss of the ELBO as a proxy for KL minimisation.

To optimise (5.9), we will apply SGD and thus require calculation of $\nabla_{\theta,\phi}\mathcal{L}_{\theta,\phi}(\mathbf{x}) = \nabla_\phi\mathcal{L}_\phi(\mathbf{x})$ (as decoder not learnable). As $\mathcal{L}_{\theta,\phi}(\mathbf{x})$ is an expectation, the exact computation of the gradient is intractable. Instead, we introduced an unbiased estimator based on the reparametrisation of $\mathbf{z}$ [63].

The reparameterisation trick allows us to express the sampling of latent variables $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$. Specifically, we reparameterise $\mathbf{z}$ as:

$$\mathbf{z} = g(\boldsymbol{\epsilon}, \phi, \mathbf{x}) = \boldsymbol{\mu}(\mathbf{x}) + \boldsymbol{\sigma}(\mathbf{x}) \odot \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{5.10}$$

where $g(\boldsymbol{\epsilon}, \phi, \mathbf{x})$ is a differentiable and invertible function, $\boldsymbol{\mu}(\mathbf{x})$ and $\boldsymbol{\sigma}(\mathbf{x})$ are the outputs of the encoder network, and $\boldsymbol{\epsilon}$ is an auxiliary variable sampled from a standard normal distribution. This reparameterisation makes the latent variable $\mathbf{z}$ a differentiable function of $\mathbf{x}$ and $\boldsymbol{\epsilon}$, making expectation and gradient operators commutative from which we can use a simple Monte Carlo estimator:

$$\nabla_\phi\mathcal{L}_\phi(\mathbf{x}) = \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\nabla_\phi \log \frac{p_\theta(\mathbf{x}, \mathbf{z}(\mathbf{x},\boldsymbol{\epsilon}))}{q_\phi(\mathbf{z}(\mathbf{x},\boldsymbol{\epsilon})|\mathbf{x})}\right] \tag{5.11}$$

$$\approx \nabla_\phi \log \frac{p_\theta(\mathbf{x}, \mathbf{z}_\epsilon)}{q_\phi(\mathbf{z}_\epsilon|\mathbf{x})} \tag{5.12}$$

where $\mathbf{z}_\epsilon = g(\boldsymbol{\epsilon}, \phi, \mathbf{x})$ for a noise sample $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The Monte Carlo estimates (5.12) are easily obtained, and auto-differentiation libraries facilitate using backpropagation.

5.3 Implementation Overview

In this section, we outline the key steps for implementing VAEs for CLSNA. As the core principles of neural calibration remain the same, we follow a similar procedure as in Chapter 4:

1. **Define Encoder Architectures:** We define the architectures of the calibration VAE models. In this report, we implement NN-VAE, CNN-VAE, and VGAE for the 207 dataset and NN-VAE for the 616 dataset.
2. **Define `simulate_clsna` Decoder:** We will explore the interpretation of `simulate_clsna` as a non-convex activation function, relating its high saturation property to weight initialisation sensitivity.
3. **Establish a training procedure:** We present the loss function, analytic formulae for f assuming Gaussian priors and posteriors, and adaptation of classical VAE training techniques.

5.4 Encoder Architectures

The first step of neural calibration is defining the VAE architectures that take as input the observed, ground-truth data Y_t and output $(\boldsymbol{\mu}, \log(\boldsymbol{\sigma^2})) \in (\mathbb{R}^5, \mathbb{R}^5)$ corresponding to the means and log-variances of each parameter's Gaussian posterior distribution. For all the encoders, the hyperparameters were drawn from the NN models of Chapter 4 and empirical training trials.

5.4.1 207 Dataset

Neural Network VAE (NN-VAE)

The NN-VAE encoder architecture is a fully connected NN, taking as input the vectorised upper-triangular $Y_{1:T}$ with shape (234531,). The NN is (hyper-) parameterised as follows:

- **Number of Layers:** 5
- **Nodes per Layer:** 96
- **Activation Functions:** Tanh for hidden layers, linear for the last layer
- **Learning Rate:** 5×10^{-4}
- **Optimiser:** Adam with weight decay 0.011258

This model has **22,543,978** parameters and a storage requirement of **21.50MB**.

CNN-VAE

The CNN-VAE encoder architecture incorporates spatial and temporal structure information of the adjacency matrices by applying a series of 3x3 convolutions over input/hidden tensor maps. The CNN takes as input the (11, 207, 207) observed $Y_{1:T}$ and is (hyper-) parametrised as follows:

- **Input Channels:** 11
- **Number of Convolutional Layers:** 5, and one fully-connected linear layer for output
- **Kernel Shape:** (3, 3)
- **Stride:** (2, 2)
- **Padding:** (1, 1)
- **Hidden Dimension$_0$:** 32, which duplicates in each subsequent layer
- **Activation Function:** ReLU for hidden layers, linear for output
- **Learning Rate:** 1e-5
- **Optimiser:** Adam with weight decay 0.011258

This CNN model has **2,436,170** parameters and a storage requirement of **2.32MB**, an 89.19% reduction in both parameters and memory usage compared to NN-VAE.

VGAE

The graph neural network (GNN) encoder architecture is (hyper-) parametrised as follows:

- **Input Channels:** 11×207
- **Hidden Dimension:** 128
- **Number of GCN Layers:** 4
- **Activation Function:** ReLU for hidden layers, linear for output
- **Learning Rate:** 10^{-5}
- **Optimiser:** Adam with weight decay 0.011258

The GNN model has **325,504** parameters and a storage requirement of **0.31MB** (86.64% less than CNN-VAE and negligible compared to NN-VAE).

5.4.2 616 Dataset

NN-VAE

For the 616 models, the only viable implementation was a basic NN because, at each time index, there is a variable number of nodes. The network takes as input the concatenation of Y_t with a shape of (1382495,). The NN encoder is (hyper-) parametrised as follows:

- **Number of Layers:** 5
- **Nodes per Layer:** 48
- **Activation Functions:** Tanh for hidden layers, linear for the last layer
- **Learning Rate:** 5×10^{-4}
- **Optimiser:** Adam with weight decay 0.011258

This model has **66,367,354** parameters and a memory usage of **63.293MB**. When we compare all VAE architectures with their corresponding standard neural calibration models we observe similar parameter and storage requirements, as expected.

5.5 `simulate_clsna` and `simulate_clsna_variable` Decoder

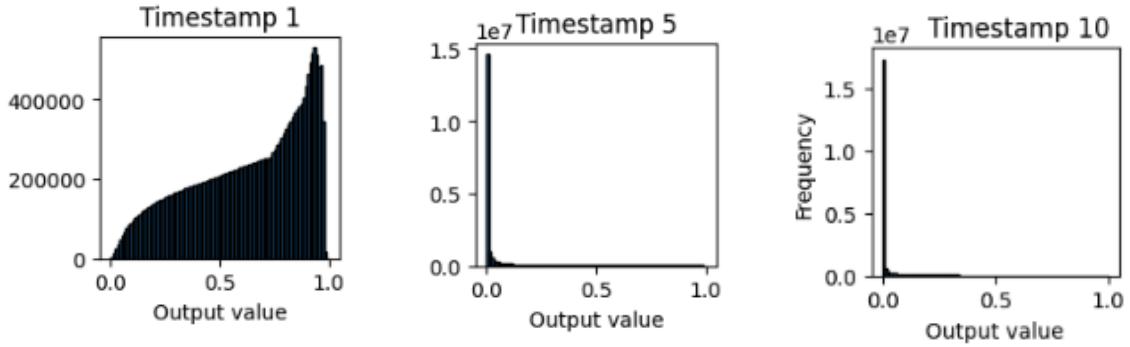


Figure 5.3: `simulate_clsna` output histograms at time steps $T = \{1, 5, 10\}$ showing high activation saturation. Similar plots for `simulate_clsna_dynamic` in Appendix D.1.

As stated in Section 5.2.3, the decoder for CLSNA VAE has no learnable parameters and can be viewed as a neural network with **0** hidden layers, and where the numerical solvers, `simulate_clsna` for 207 and `simulate_clsna_variable` for 616, act as non-convex *activation functions*. When viewed in this light, the properties of the numerical solvers become directly relevant to understanding the behaviour of the VAE during training. We will consider these functions as black-box mappings $f : \mathcal{Z} \to \mathcal{X}$, where $\mathcal{Z}$ and $\mathcal{X}$ are the latent variable and observed data spaces, respectively.

The property of interest is saturation, which will be quantified using the metric introduced in Section 3.2. High saturation levels in a function hinder a network's learning capability due to the vanishing gradient problem. In such cases, only inputs within the function's active range produce gradients with sufficient magnitude to propagate through multi-layer deep networks.

In our evaluations, we observed significant saturation levels in both fixed and variable node `simulate_clsna`. For `simulate_clsna`, the saturation values ranged from 0.506 to 0.933 with a combined average of 0.851, while for `simulate_clsna_dynamic`, the values ranged from 0.507 to 0.906 with a combined average of 0.864. Figure 5.3 shows the output histograms over a range of inputs at discrete timestamps; in particular, we can see low saturation only at timestamp 1. These values are higher than those seen with the Sigmoid and TanH functions in Rakitianskia and Engelbrecht's study [32], where both standard activation functions had values of approximately 0.7. The high saturation we measured indicates a tendency for gradients to diminish, thereby explaining the sensitivity we have observed to initial conditions and parameter initialisation in our training process.

5.6 Training Procedure

The training procedure for all models is standard and identical across each implementation. In each epoch, we perform the following steps:

1. Feed forward through our ML architecture and decoder.
2. Calculate the loss function explicitly.
3. Backpropagate the loss to update the model parameters.

5.6.1 Analytic Form of Loss Function

An analytic form is available because we assumed factored Gaussian priors and posteriors. From Equation 5.11, we see that the Monte Carlo unbiased estimate of the ELBO is given by:

$$\tilde{\mathcal{L}}_{\theta,\phi}(\mathbf{x}) = \log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}) \tag{5.13}$$

$$= \log p_\theta(\mathbf{x}|\mathbf{z}) + \log p_\theta(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x}) \tag{5.14}$$

$$= \log p_\theta(\mathbf{x}|\mathbf{z}) + \log p_\theta(\mathbf{z}) - (\log p_\theta(\boldsymbol{\epsilon}) - \log \left|\frac{\partial \mathbf{z}}{\partial \boldsymbol{\epsilon}}\right|) \tag{5.15}$$

where in (5.14) we have used the law of conditional probability, and in (5.15) we have used the log-determinant of the transformation from $\boldsymbol{\epsilon}$ to $\mathbf{z}$ [46]. We immediately recognise that the first term of (5.15) is the binary cross entropy reconstruction loss, and we can split the loss function into three terms:

$$\mathcal{L} = \mathcal{L}_{\text{recon}} + \mathcal{L}_{\text{prior}} + \mathcal{L}_{\text{post}} \tag{5.16}$$

Given that we assume Gaussian priors for our five parameter estimates,

$$\log p_\theta(\mathbf{z}) = \log \prod_i \mathcal{N}(z_i; \mu_i^{\text{prior}}, \sigma_i^{\text{prior}}) \tag{5.17}$$

This is expanded to:

$$\log p_\theta(\mathbf{z}) = \sum_{i=1}^{5} \log \left[\frac{1}{\sqrt{2\pi}\sigma_i^{\text{prior}}} \exp\left(-\frac{(z_i - \mu_i^{\text{prior}})^2}{2(\sigma_i^{\text{prior}})^2} \right) \right] \tag{5.18}$$

$$= \sum_{i=1}^{5} \left[-\frac{1}{2}\log(2\pi) - \log \sigma_i^{\text{prior}} - \frac{(z_i - \mu_i^{\text{prior}})^2}{2(\sigma_i^{\text{prior}})^2} \right] \tag{5.19}$$

$$= -\frac{5}{2}\log(2\pi) - \sum_{i=1}^{5} \log \sigma_i^{\text{prior}} - \sum_{i=1}^{5} \frac{(z_i - \mu_i^{\text{prior}})^2}{2(\sigma_i^{\text{prior}})^2} \tag{5.20}$$

Similarly, as we have assumed factorised Gaussian priors, calculation of $\log q_\phi(\mathbf{z}|\mathbf{x})$ follows similarly:

$$\log q_\phi(\mathbf{z}|\mathbf{x}) = \log \prod_i \mathcal{N}(\epsilon_i; 0, I) + \sum_i \log \sigma_i \tag{5.21}$$

$$= -\frac{5}{2}\log(2\pi) - \sum_{i=1}^{5} \frac{\epsilon_i^2}{2} + \sum_{i=1}^{5} \log \sigma_i \tag{5.22}$$

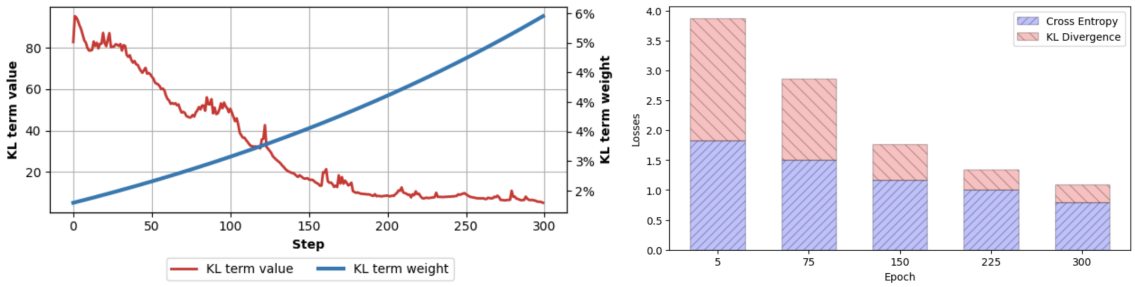
Therefore, the analytic form of the loss function, combining the reconstruction loss and the KL divergence, is derived as follows:

$$\mathcal{L}_{\text{recon}} = \log p_\theta(\mathbf{x}|\mathbf{z}) \tag{5.23}$$

$$\mathcal{L}_{\text{prior}} = -\frac{5}{2}\log(2\pi) - \sum_{i=1}^{5} \log \sigma_i^{\text{prior}} - \sum_{i=1}^{5} \frac{(z_i - \mu_i^{\text{prior}})^2}{2(\sigma_i^{\text{prior}})^2} \tag{5.24}$$

$$\mathcal{L}_{\text{post}} = -\frac{5}{2}\log(2\pi) - \sum_{i=1}^{5} \frac{\epsilon_i^2}{2} + \sum_{i=1}^{5} \log \sigma_i \tag{5.25}$$

5.6.2 KL Annealing and Gradient Clipping



(a) Annealed KL Divergence Loss

(b) Grouped KL and Reconstruction Losses

Figure 5.4: KL Annealing effects on training loss of NN-VAE on 616 datasets. See Appendix D.3 for similar plots on other architectures.

To ensure model convergence, we employed two standard techniques used in VAE training: KL annealing and gradient clipping. KL annealing was introduced by Bowman et al. in the paper "Generating Sentences from a Continuous Space" [66]. The primary purpose of KL annealing is to (1) allow the reconstruction losses to be minimised initially and (2) then adapt the model to the posteriors in the latter epochs. This is accomplished by gradually increasing the weight of the KL divergence factor in the loss function over successive training epochs. By introducing a coefficient of $\mathcal{L}_{post}$ we deviate from the theoretical loss of Equation (5.16), but this has shown to be effective in practice.

In Figure 5.4(a), we observe the controlled decrease of the KL divergence loss as the KL annealing coefficient increases. This coefficient follows the formula:

$$\text{KL\_weight}(\text{epoch}) = \frac{1}{1 + \exp(-k(\text{epoch} - x_0))} \tag{5.26}$$

The hyperparameters $k = 0.003$ and $x_0 = 1250$ were chosen to exhibit this behavior. On the right, Figure 5.4(b) shows the grouped KL and reconstruction losses, which decrease steadily over the training epochs. This demonstrates the effectiveness of KL annealing in balancing the different components of the loss function. Additionally, as done in the training of the standard neural calibration models, gradient clipping was employed to prevent the gradients from exploding, a by-product of highly saturated activation functions like `simulate_clsna`.

Figure 5.5 shows the total, reconstruction, and KL losses over epochs for the VGAE trained on the 207 datasets, where the dark-blue moving averages exhibit a decreasing trend over epochs.

5.7 Results

5.7.1 Parameter Estimates

Table 5.1 shows the parameter estimates and their standard deviations (SD) for NN-VAE, CNN-VAE, and VGAE models trained on the 207 dataset. We include SGD and MCMC for comparison. We observe the consistent negative value for γ_b across all models, indicating a tendency towards network polarisation. Compared to SGD, the magnitude of γ_w^1 is significantly larger in the VAE models, suggesting stronger within-group interactions. However, the relationship $\gamma_w^1 < \gamma_w^2$ is maintained, reflecting that the influence within groups (indexed by γ_w^1) is less than that across different groups (γ_w^2). Another key observation is that the SDs are much larger than MCMC for all VAE models, and comparable to SGD for the inter-/intra- coefficients. We do see lower SDs for γ_w^1, γ_w^2, γ_b than α, δ inline with the SA relationship we identified between low SDs and parameter significance. Table 5.2 shows similar patterns for NN-VAE trained on 616 datasets.

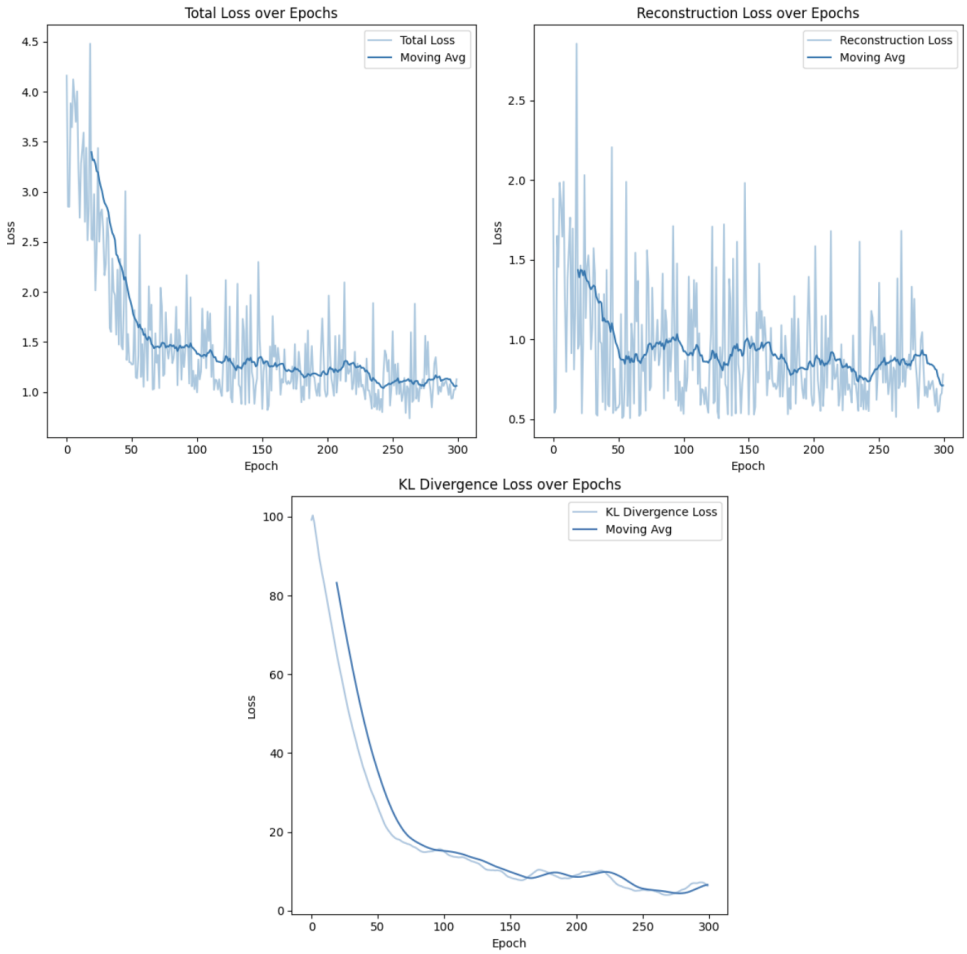


Figure 5.5: Total, Reconstruction, and KL Loss Curves for VGAE trained on 207 datasets. See Appendix D.2 for a full set of loss curves across architectures.

Model	$\hat{\alpha}$	$\hat{\delta}$	$\hat{\gamma}_w^1$	$\hat{\gamma}_w^2$	$\hat{\gamma}_b$
SGD[5]	2.241 (0.158)	1.464 (0.156)	0.075 (0.174)	0.406 (0.206)	-0.182 (0.127)
MCMC[4]	2.809 (0.022)	1.500 (0.018)	0.493 (0.026)	0.105 (0.025)	-0.155 (0.014)
NN-VAE	2.287 (0.378)	0.857 (0.400)	0.476 (0.360)	0.613 (0.299)	-0.211 (0.268)
CNN-VAE	2.223 (0.348)	0.834 (0.369)	0.478 (0.314)	0.631 (0.266)	-0.199 (0.224)
VGAE	1.789 (0.379)	0.983 (0.392)	0.604 (0.360)	0.688 (0.305)	-0.163 (0.331)

Table 5.1: Parameter estimates and standard deviations (SD) for NN-VAE, CNN-VAE, and VGAE models trained on the **207** dataset. MCMC and SGD are included for comparison.

Model	$\hat{\alpha}$	$\hat{\delta}$	$\hat{\gamma}_w^1$	$\hat{\gamma}_w^2$	$\hat{\gamma}_b$
SGD[5]	3.223 (0.104)	1.085 (0.101)	-0.113 (0.115)	0.328 (0.138)	-0.214 (0.091)
NN-VAE	1.789 (0.380)	0.983 (0.387)	0.604 (0.389)	0.688 (0.363)	-0.163 (0.345)

Table 5.2: Parameter estimates and standard deviations (SD) for NN-VAE trained on the **616** dataset. SGD is included for comparison.

Dataset	Model	GPU	CPU
207	**SGD**	<10	42.92 (1.208)
	MCMC	N/A	≈ 240
	NN-VAE	**1.759** (0.033)	6.067 (0.301)
	CNN-VAE	1.991 (0.024)	3.74 (0.107)
	VGAE	2.095 (0.031)	3.298 (0.156)
616	**SGD**	<20	78.89 (1.452)
	NN-VAE	**3.460** (0.194)	11.82 (0.421)

Table 5.3: Execution times reported in **minutes**. Standard deviations (SD), when available, are listed in parentheses. Blue highlights the best GPU (**Tesla T4**) training time in each dataset, respectively.

5.7.2 Execution Time

Table 5.3 presents the execution times for training the VAE models on the 207 and 616 datasets, reporting GPU and CPU times in minutes. As expected, we see **significant** improvements from baseline SGD and MCMC. Comparing the VAE models (NN-VAE, CNN-VAE, VGAE) to standard neural calibration models (NN, CNN, GNN), we note the following key insights:

- The analysis shows that neural calibration models generally perform better. For example, the standard CNN model is approximately 55.00% faster than the CNN-VAE model on GPU (0.896 minutes vs. 1.991 minutes).
- A counterintuitive result is observed with NN-VAE on the 616 dataset, where the NN-VAE GPU training time is around 5.5% faster than the standard NN model (3.46 minutes vs. 3.67 minutes). This could be because with larger node systems, gradient calculation becomes more of a bottleneck than `simulate_clsna`, explaining this behaviour.

The last bullet point highlights a major pro of neural calibration: scalability. Although the number of parameters in NN-VAE grows proportionally to input_size^2, the architecture is still capable of learning quickly.

5.7.3 Evaluation Metrics

Table 5.4 presents the performance metrics and standard deviations (SD) for various models on the **207** dataset.

We note that VAE implementations exhibit better performance on traditional classification metrics (Precision, Recall, F1, and AUC) compared to standard neural calibration. For instance, on the 207 dataset, the CNN-VAE and VGAE models achieve superior precision and recall values. The AUC for VGAE (0.614) is marginally higher than other models, indicating better discriminative power.

There is significantly improved performance in terms of the (Δ Clustering Coefficient for the VAEs. The VAE values lie between those of MCMC and SGD, demonstrating the VAE models' ability to capture the topological structure of the graph more effectively. This is crucial for our use case where understanding behavioural trends is the main focus. Figure 5.6 illustrates this improvement, where the horizontal and vertical patterns present in the ground truth are more evident in the VAE reconstructions.

For the **616** dataset, the NN-VAE model is promising, outperforming SGD in all metrics except for Δ closeness centrality. The improvement in Δ clustering coefficient is notable, with the NN-VAE

Metric	MCMC	SGD	$\mathbf{NN_{VAE}}$	$\mathbf{CNN_{VAE}}$	VGAE
Weighted BCE$_\ell$	1.441 (0.058)	1.017 (0.043)	0.795 (0.040)	*0.772* (0.039)	**0.722** (0.033)
Precision$_h$	0.190 (0.006)	0.271 (0.015)	*0.279* (0.019)	**0.283** (0.017)	*0.279* (0.015)
Recall$_h$	0.209 (0.013)	0.256 (0.028)	*0.358* (0.036)	**0.360** (0.034)	0.347 (0.029)
F1$_h$	0.199 (0.007)	0.263 (0.021)	*0.313* (0.023)	**0.316** (0.020)	0.309 (0.018)
AUC$_h$	0.433 (0.010)	0.509 (0.018)	0.604 (0.025)	*0.611* (0.021)	**0.614** (0.018)
Δ Clustering Coeff$_\ell$	**0.117** (0.013)	0.195 (0.017)	0.146 (0.022)	*0.145* (0.019)	0.178 (0.020)
Δ Degree Centrality$_\ell$	0.235 (0.026)	0.247 (0.036)	*0.074* (0.056)	**0.063** (0.035)	0.078 (0.040)
Δ Closeness Centrality$_\ell$	*0.045* (0.009)	**0.037** (0.009)	0.131 (0.031)	0.140 (0.026)	0.156 (0.025)

Table 5.4: Performance metrics and standard deviations (SD) for the **207** dataset. MCMC and SGD are included for comparison. Values computed from 100 independent simulations. Superscripts $_h$ and $_\ell$ indicate whether higher or lower values are better for the metric. The best and second-best performances are in **bold** and *italics*, respectively.

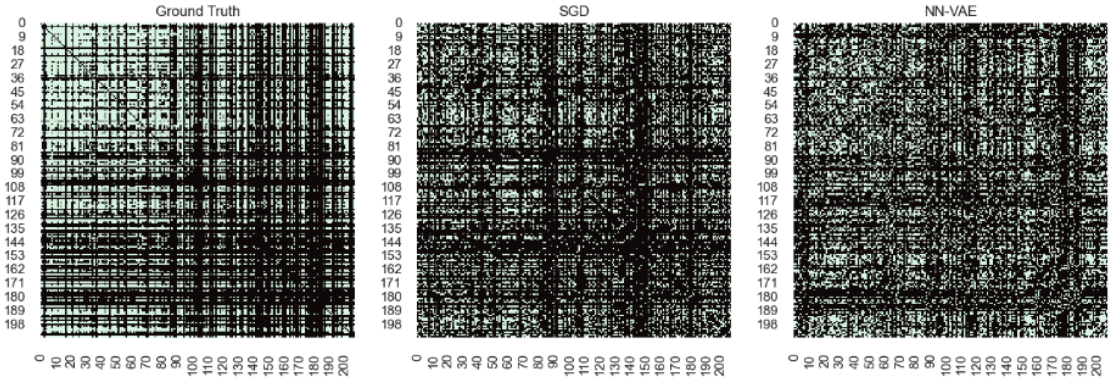


Figure 5.6: **207** Adjacency Matrix heat-maps for Ground Truth, SGD, and NN-VAE at timestep $t = 10$.

showing a significant reduction from SGD.

Comparing the performance of the VAE models with the standard neural calibration models on both datasets, we generally observe **decreased** classification metric performance but **improved** topological metrics performance, highly influenced by our priors. This observation motivates the potential use of both methods in conjunction - first applying standard neural calibration with non-informative priors and subsequently using VAEs to update the posteriors of the parameter distributions. Although this would explicitly constrain the parameter PDFs to factorised Gaussians, we may obtain better results. The latency of a combined approach is not a concern as it would still be substantially lower than SGD.

5.8 Discussion

The performance of the NN-VAEs is at the very least competitive with standard neural calibration, if not better, particularly in capturing the topological structure of adjacency matrices. This discussion focuses on the benefits and drawbacks of the VAE neural calibration interpretation.
VAEs have successfully served to update prior beliefs to posteriors. In Figure 5.7, we present the evolution of parameter posteriors over training epochs. In the early epochs, such as epoch 150 (blue

Metric	SGD	$\mathbf{NN_{VAE}}$
Weighted BCE$_\ell$	1.354 (0.029)	**0.760** (0.020)
Precision$_h$	0.327 (0.007)	**0.337** (0.005)
Recall$_h$	0.242 (0.010)	**0.339** (0.014)
F1$_h$	0.278 (0.008)	**0.338** (0.007)
AUC$_h$	0.556 (0.007)	**0.607** (0.006)
Δ Clustering Coeff.$_\ell$	0.315 (0.009)	**0.158** (0.018)
Δ Degree Centrality$_\ell$	0.295 (0.037)	**0.045** (0.048)
Δ Closeness Centrality$_\ell$	**0.016** (0.008)	0.072 (0.009)

Table 5.5: Performance metrics and standard deviations (SD) for the **616** dataset. SGD is included for comparison. Values computed from 100 independent simulations. Superscripts $_h$ and $_\ell$ indicate whether higher or lower values are better for the metric. The better metric values are in **bold**.

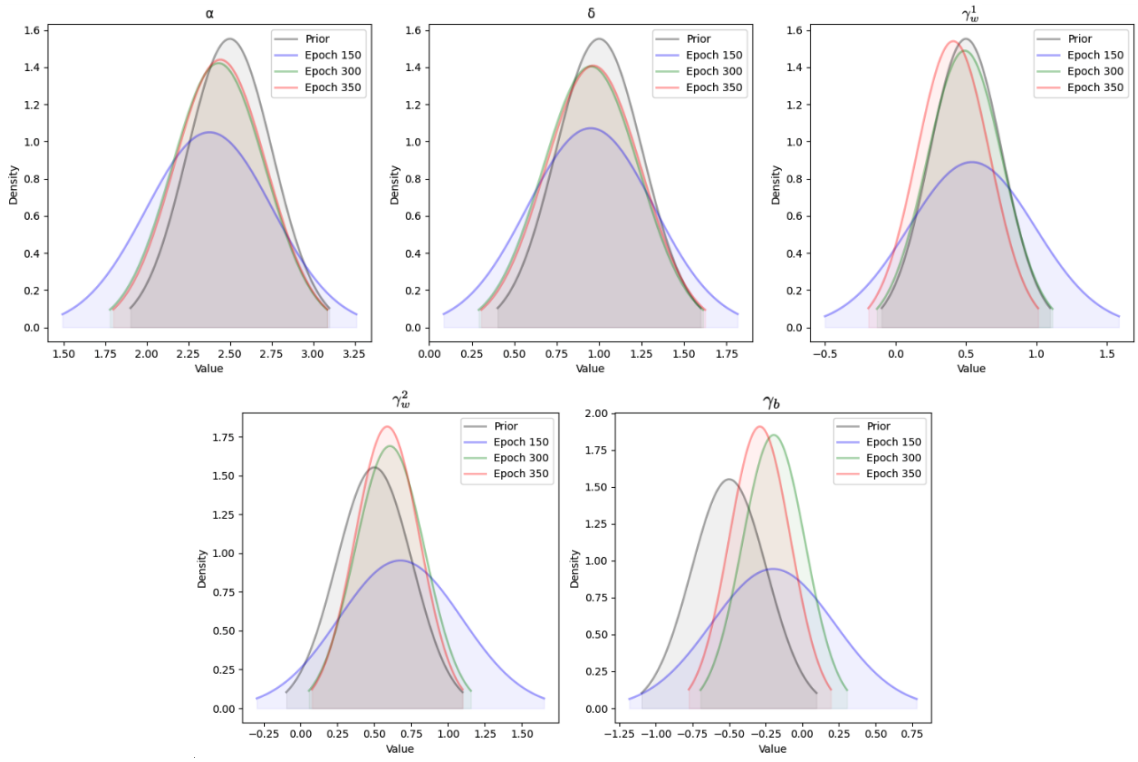


Figure 5.7: Posterior updates of $\{\gamma_w^1, \gamma_w^2, \gamma_b\, \alpha, \delta\}$ parameters in CNN-VAE trained on 207 dataset. See Appendix D.4 for additional posterior plots.

curve), the distributions flatten out as we are still exploring the hyperparameter space, potentially aided by KL annealing. In later epochs (green and red curves), the distributions become more defined, indicating exploitation and concretisation of the distributions.

However, achieving the best results with VAE neural calibration requires thorough prior knowledge of the system dynamics, which was only available to us from MCMC, SGD, and NN/CNN/GNN neural calibration. This is problematic for general purposes, but in the specific context of opinion

dynamics, it is not unreasonable to assume some prior beliefs or intuitions. Opinion dynamics models are intentionally developed to capture macrosocial behaviour trends from different interdisciplinary fields (i.e., economics, history, and marketing).

Additionally, VAEs require much more tuning than standard neural calibration, with additional considerations such as KL annealing and harsh constraints on priors/posteriors. It is noted in the literature that generative models tend to make stronger assumptions on the data than discriminator ML architectures, often leading to higher asymptotic bias [67]. This is the case for the VAE neural calibration interpretation.

Lastly, we can gain intuition on differences between standard neural calibration and the VAE models by visually comparing the density estimates, as shown in Figure 5.8. While the mean and mode estimates are relatively consistent across $\gamma_w^1, \gamma_w^2, \gamma_b$, the variances do not align. This is because the VAE SD estimates are heavily influenced by the priors. Unlike point estimates for the parameters, having prior intuition for SDs is not reasonable. A potential solution could be concatenating standard neural calibration and VAE, as suggested at the end of 5.7.3. This way, we can establish priors with neural calibration and update them with VAEs.

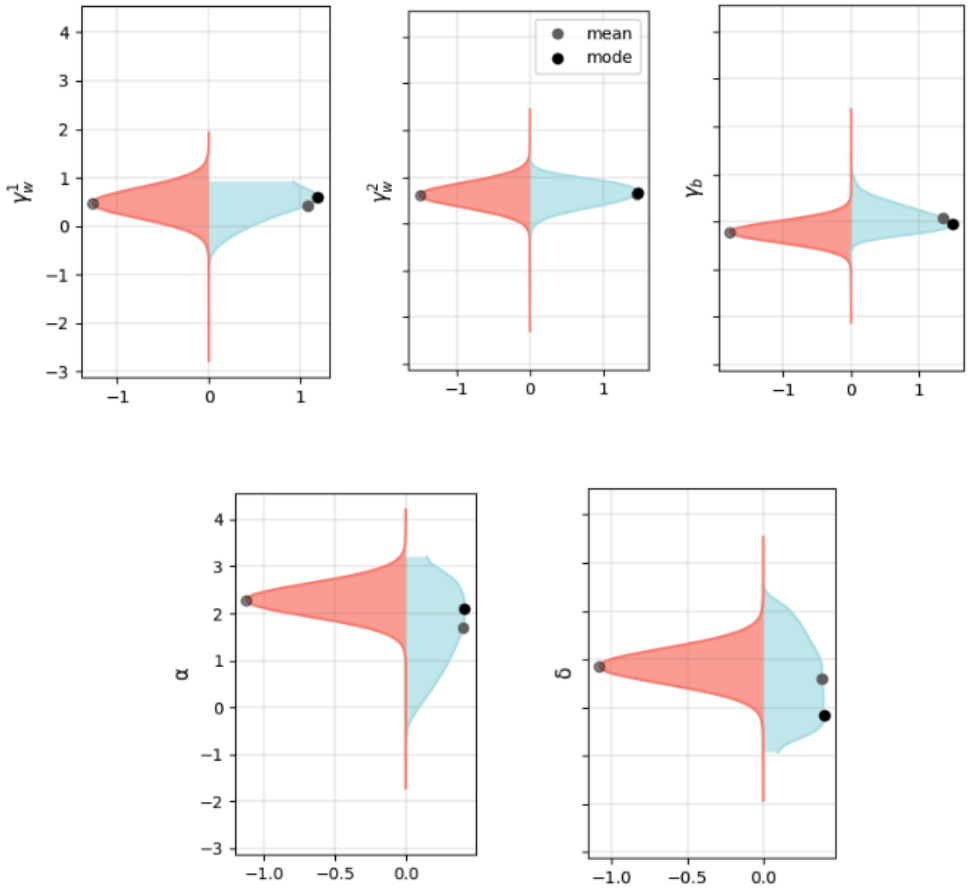


Figure 5.8: Marginals for parameters $\lambda = \{\gamma_w^1, \gamma_w^2, \gamma_b, \alpha, \delta\}$ for both CNN-VAE calibration with factorised Gaussian posteriors (red, left side) and NN neural calibration (blue, right side) on 207 datasets. Black dots correspond to distribution modes and grey dots to means. See Appendix D.5 for a complete set of marginal density plots.

Chapter 6

Conclusion

In this work, we successfully implemented neural calibration for the CLSNA opinion dynamics model. In doing so, we extended Gaskin et al.'s scheme beyond fully connected neural networks to convolutional and graph architectures. We demonstrated that neural calibration is significantly faster than the baseline MCMC and SGD algorithms, converging in approximately **80%** less time for both fixed and variable node CLSNA while producing promising evaluation results. Regarding the parameter estimates, we observed polarisation in the hashtag networks between the Democrat and Republican political parties, in line with results published by Zhu et al. [4] and Pan et al. [5]. Among the NN, CNN, and GNN models on the 207 dataset, we highlighted that the **GNN** was superior, performing the best on average across five classifications and three topological evaluation metrics.

We then conducted a sensitivity analysis using the neural calibration parameter probability density functions to quantitatively determine which parameters in $\lambda := \{\alpha, \delta, \gamma_w^1, \gamma_w^2, \gamma_b\}$ contribute the most to CLSNA output variance. The SA included the Morris and Sobol methods, as well as constructing a Pareto front with the NSGA-II evolutionary algorithm. We identified a **clear relationship** between low neural calibration standard deviation estimates and parameter significance, suggesting the viable use of neural calibration as an SA tool.

Building on these findings and addressing the challenges in training NN/CNN/GNNs, we explored the Bayesian interpretation of neural calibration as a VAE. To this end, we used CLSNA calibration to update prior beliefs using SGD and obtain robust parameter posteriors. We implemented three encoder architectures, and presented the calibration results for NN-VAE, CNN-VAE, and GNN-VAE, demonstrating their enhanced ability to identify topological structures in the observed data. Notably, the CNN-VAE obtained a Δ Clustering Coefficient value of 0.145, marginally worse than MCMC (0.117) but a marked improvement over the GNN trained with standard neural calibration (0.199). Despite the refined pattern recognition, the VAEs also had various drawbacks, including the necessity of a thorough understanding of CLSNA dynamics a priori and the constraint of Gaussian priors and posteriors.

6.1 Future Work

To further advance this research, future work should focus on two main avenues: (1) enhancing the neural calibration scheme itself and (2) improving the performance of neural calibration for the CLSNA application. The former is geared towards generalising many of the key insights made in the execution of this report while the latter addresses the limitations of the solutions presented in Chapters 4 and 5.

Concerning the standard and VAE neural calibration implementation for CLSNA, we hereby suggest:

- **Addressing Data Limitations.**
  The hashtag datasets used for both the fixed and dynamic node CLSNA variants represent a single time-series realisation of behaviour from 2010-2020; that is to say, we have limited data that the ML models can learn from. This is a common problem in time-series analysis,

and a few exciting solutions have been proposed for similar use cases.

For example, Gao et al., in the paper titled "Stochastic Graph Neural Networks", introduce controlled perturbations of input graph data to simulate topological randomness [68]. At its core, the stochastic GNN generates a family of random graphs $\mathcal{G}_k = (\mathcal{V}, \mathcal{E}_k)$ from $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ by. Additionally, MarkovGNNs use a dynamic adjacency matrix that updates at each convolutional layer via a predefined Markov process [69]. This would promote a stochastic diffusion of information among graph nodes that may model social media behaviour more realistically. We believe incorporating principles from these recent developments could have a notable impact on the quality of the parameter estimations.

- **Employing Sophisticated Architectures to 616 Dataset.**
  We have only used basic NN models for the vectorised 616 dataset due to the presence of nodes entering and exiting the system. Ideally, we would use a convolutional architecture (CNN/GNN) that allows for spatial and temporal recognition.

  A potential solution would be transforming the 616 dataset to define a larger $\mathcal{G}_{tot} = (\mathcal{V}_{tot}, \mathcal{E}_{tot})$ that has a fixed $\mathcal{V}_{tot}$ containing the union of nodes observed at all time steps. Upon doing this, implementing either CNNs or GNNs would follow the same procedure of Sections 4.3 and 5.4. This data transformation is non-trivial, as it is equivalent to a feature-embedding imputation task. To this end, we propose integrating You et al.'s GRAPE framework [70] or Isallari and Rekik's super-resolution methodology [48].

Concerning the robustness and performance of the neural calibration scheme, we put forth the following ideas:

- **Formalising Neural Calibration for SA.**
  In Section 4.8 we provided a relationship between estimated parameter variance and significance. We believe exploring the viability of neural calibration as an SA methodology, either through theoretical derivations or by showing similar patterns empirically in other applications (i.e. the SIR model), is a worthwhile venture.

- **Going Beyond Non-Gaussian VAE Priors/Posteriors.**
  We used Gaussian priors and posteriors in the VAE implementation due to the convenience of closed-form loss function derivation and the simplicity of distribution interpretation. However, this constraint might be too restrictive. Considering non-Gaussian posteriors could enhance the flexibility of the inference model $q_\theta(\mathbf{z}|\mathbf{x})$, thereby tightening the ELBO bound. Two general techniques, as presented by Kingma and Welling [62], are worth considering.

  The first technique involves incorporating auxiliary latent variables $\mathbf{u}$ to the q_θ, i.e., $q_\theta(\mathbf{z}|\mathbf{x}) = \int q_\theta(\mathbf{u}, \mathbf{z}|\mathbf{x})\, d\mathbf{u}$. This approach may initially worsen the bound due to errors in the integral approximation, but the increased flexibility could potentially outweigh this cost. The second technique is the Inverse Autoregressive Flow (IAF). This method involves a chain of transformations from a basic initial distribution using an Autoregressive Neural Network (ANN). This way, we do not have to modify the encoder architectures and instead train the ANN to transform Gaussian posteriors into a complex set of distributions.

- **2-Step Neural Calibration.**
  In the concluding remarks of Chapter 5 we proposed a 2-Step calibration scheme that composes a standard neural calibration and a VAE implementation. In this way, we address the limitation of VAEs requiring prior information by first generating empirical PDFs for the parameters and using these as the VAE's latent variable priors. This methodology aligns well with the previous suggestion of transitioning to a more generic Bayesian framework. A thorough analysis of this 2-step calibration would be a valuable extension of the current work.

We leave these extensions as opportunities to catalyse the usage of neural calibration as a **fast** and **accurate** tool across various fields of mathematics, computer science, and interdisciplinary research. By significantly reducing operational costs and risks — key inhibitors in the uptake of new technologies — we believe neural calibration can become widely adopted.

Appendix A

CLSNA Numerical Solver

A core component of neural parameter calibration is the numerical solver that abstractly takes parameter inputs and outputs simulation results. Efficiently implementing the `simulate_clsna` and `simulate_clsna_dynamic` numerical solvers that correspond to the 207 and 616 datasets proved to be both a conceptual and code-bug prone challenge – especially the latter. We will now present the data processing and pseudo code[1] for both solvers, commenting on operations that were parallelised to minimise latency and prevent the solver being a calibration bottleneck. The methodology for this section involves a series of incremental modifications (specifically for `simulate_clsna_dynamic`) to the CLSNA solver implementations originally developed by Zhu et al.[2]. Full credit for the foundational work is attributed to their research team.

In A.1 and A.2, $\odot$ represents element-wise operation application and s the L2 euclidean norm.

A.1 `simulate_clsna`

The implementation of `simulate_clsna` is relatively straightforward. The key insights made by Zhu et al. were in how to parallelise the operations, specifically by using `np.outer` and `torch.nn.functional.laplacian` to calculate matrices where each entry corresponds to the inter-party and intra-party attractor values, $A_{w/b}(z_t, Y_{t-1})$.

Algorithm 2 `simulate_clsna`

Assumes: $\sigma^2 = 1, \tau^2 = 10$ fixed constants.
Input: N: Fixed number of nodes. M: Binary list mapping node index to political party.
D: Dim of latent space. T: Number of time steps. $\lambda = (\alpha, \delta, \gamma_w^1, \gamma_w^2, \gamma_b)$ model params.
Output: $(\hat{\mathbf{z}_{1:T}}, \hat{Y_{1:T}}, \hat{P_{1:T}})$: Latent position, adjacency matrices, probability matrices estimates respectively.

$\mathbf{z}_0 \leftarrow Z_0 \in \mathbb{R}^{N \times D} \sim \mathcal{N}(0, \tau^2)$
$\text{logit}(p_{0,ij}) \leftarrow \alpha - s(\mathbf{z}_{0,i}, \mathbf{z}_{0,j})$ ▷ Optimised with torch's `pdist` pairwise distance.
$Y_0 \leftarrow \text{Binomial} \odot (P_0)$
for $t = 1$ to T **do**
  $\bar{z}^1_{t,i} \leftarrow \frac{1}{|S_1^i|} \sum_{j \in S_1^i} z_{t-1,j}$ ▷ Vectorized using `scipy` and `sklearn`'s normalised Laplacian.
  $\bar{z}^2_{t,i} \leftarrow \frac{1}{|S_2^i|} \sum_{j \in S_2^i} z_{t-1,j}$
  $A_i^w(z_{t-1}, Y_{t-1}) \leftarrow \bar{z}^1_{t,i} - z_{t-1,i}$
  $A_i^b(z_{t-1}, Y_{t-1}) \leftarrow \bar{z}^2_{t,i} - z_{t-1,i}$
  $\mu_t \leftarrow z_t + \gamma_w^1 \times A_i^w(z_{t-1}, Y_{t-1}) + \gamma_w^2 \times A_i^b(z_{t-1}, Y_{t-1}) + \gamma_b \times (A_i^b(z_{t-1}, Y_{t-1}))$
  $z_t \leftarrow \mu_t + \mathcal{N}(0, \sigma^2)$
end for
Return: $\mathbf{z}_{1:T}, Y_{1:T}, P_{1:T}$

[1] All code can be found in `/clsna_model/clsna_model.py` of the NeuralCLSNA GitLab Repository: https://gitlab.doc.ic.ac.uk/og519/nerualclsna
[2] https://github.com/KolaczykResearch/SGD4CLSNA

A.2 `simulate_clsna_dynamic`

The key challenge in implementing the variable-node CLSNA solver is keeping track of node indices; more specifically, mapping persistent node indices at time t to $t+1$, discarding nodes that exist the system, and adding in new agents. To achieve this, using the approach of Zhu et al., we constructed four key data structures during data prepocessing: `prev_at_t`, `new_at_t`, `ar_pairs` and `persist`. `prev_at_t` and `new_at_t` are both time-indexed lists where each entry at index t is a subset of nodes $N' \subseteq N_{total}$ that are either present or absent in system. N_{total} represents the union of all participating agents in the dataset. `ar_pairs` is also a time-indexed list, containing tuples of where persistent node indices are in subsequent times, i.e. pairs of the form ($\text{ix}_{n,t}$,$\text{ix}_{n,t+1}$). Lastly, `persist` contains sub-adjacency matrices $Y_t^{'} \subseteq Y_t$ for edge-persistence.

Algorithm 3 `simulate_clsna_variable`

Assumes: $\sigma^2 = 1, \tau^2 = 10, \phi^2 = 10$ fixed constants.

Input: n_{nodes}: Array of node counts at each time step.
M: Binary list mapping node index to political party.
D: Dim of latent space. T: Number of time steps.
$\lambda = (\alpha, \delta, \gamma_w^1, \gamma_w^2, \gamma_b)$: Model parameters.
`ar_pairs`: Array of pairs of nodes that persist.
`prev_at_t`: Array of dictionaries of persistent nodes per party.
`new_at_t`: Array of dictionaries of new nodes per party.
`persist`: Array of persistent adjacency matrices.

Output: ($\hat{\mathbf{z}_{1:T}}$, $\hat{Y_{1:T}}$, $\hat{P_{1:T}}$): Latent position, adjacency matrices, probability matrices estimates respectively.

$\mathbf{z}_0 \leftarrow Z_0 \in \mathbb{R}^{N_0 \times D} \sim \mathcal{N}(0, \tau^2)$
$\text{logit}(p_{0,ij}) \leftarrow \alpha - s(\mathbf{z}_{0,i}, \mathbf{z}_{0,j})$ ▷ Optimised with torch's `pdist` pairwise distance.
$Y_0 \leftarrow \text{Binomial} \odot (P_0)$
$Z \leftarrow [\mathbf{z}_0]$, $P \leftarrow [p_0]$, $Y \leftarrow [Y_0]$
for $t = 1$ to T **do**
    `prev_t` $\leftarrow$ `prev_at_t`$[t]$
    `new_t` $\leftarrow$ `new_at_t`$[t]$
    `persist_t` $\leftarrow$ `persist`$[t]$
    $\bar{z}_{t,i}^1 \leftarrow \frac{1}{|S_1^i|} \sum_{j \in S_1^i} z_{t-1,j}$ ▷ Vectorized using `scipy` and `sklearn`'s normalised Laplacian.
    $\bar{z}_{t,i}^2 \leftarrow \frac{1}{|S_2^i|} \sum_{j \in S_2^i} z_{t-1,j}$
    $A_i^w(z_{t-1}, Y_{t-1}) \leftarrow \bar{z}_{t,i}^1 - z_{t-1,i}$
    $A_i^b(z_{t-1}, Y_{t-1}) \leftarrow \bar{z}_{t,i}^2 - z_{t-1,i}$
    $\mu_t \leftarrow z_t + \gamma_w^1 \times A_i^w(z_{t-1}, Y_{t-1}) + \gamma_w^2 \times A_i^b(z_{t-1}, Y_{t-1}) + \gamma_b \times (A_i^b(z_{t-1}, Y_{t-1}))$
    $z_t \leftarrow \mu_t + \mathcal{N}(0, \sigma^2)$
    ▷ Latent positions for incoming nodes
    $new\_s1\_zt \leftarrow \bar{z}_{t,i}^1 + \mathcal{N}(0, \phi^2)$
    $new\_s2\_zt \leftarrow \bar{z}_{t,i}^2 + \mathcal{N}(0, \phi^2)$
    ▷ Map z_t to correct indices from previous and new nodes
    $z_t \leftarrow \text{zeros}([n_{nodes}[t], D])$
    for $(i, j) \in$ `ar_pairs`$[t-1]$ **do**
        $z_t[j] \leftarrow z_t[i]$
    end for
    z_t[`new_t`$[0]$] $\leftarrow new\_s1\_zt$
    z_t[`new_t`$[1]$] $\leftarrow new\_s2\_zt$
    ▷ Calculate $p_{t,ij}$ probabilities for new and previous nodes
    $\text{logit}(p_{t,ij}) \leftarrow \alpha - s(\mathbf{z}_{t,i}, \mathbf{z}_{t,j}) + \delta \times$ `persist_t`
    $Y_t \leftarrow \text{Binomial} \odot (P_t)$
    $Z \cup [\mathbf{z}_t]$, $P \cup [p_t]$, $Y \cup [Y_t]$
end for
Return: $\mathbf{z}_{1:T}$, $Y_{1:T}$, $P_{1:T}$

Appendix B

CLSNA Neural Calibration: Auxiliary Plots

The main report only includes representative figures (i.e. for only one type of architecture) sufficient to guide the reader through the analysis and discussion. In this appendix we present the full set of relevant plots.

B.1 Full X (Twitter) Dataset Adjacency Matrices

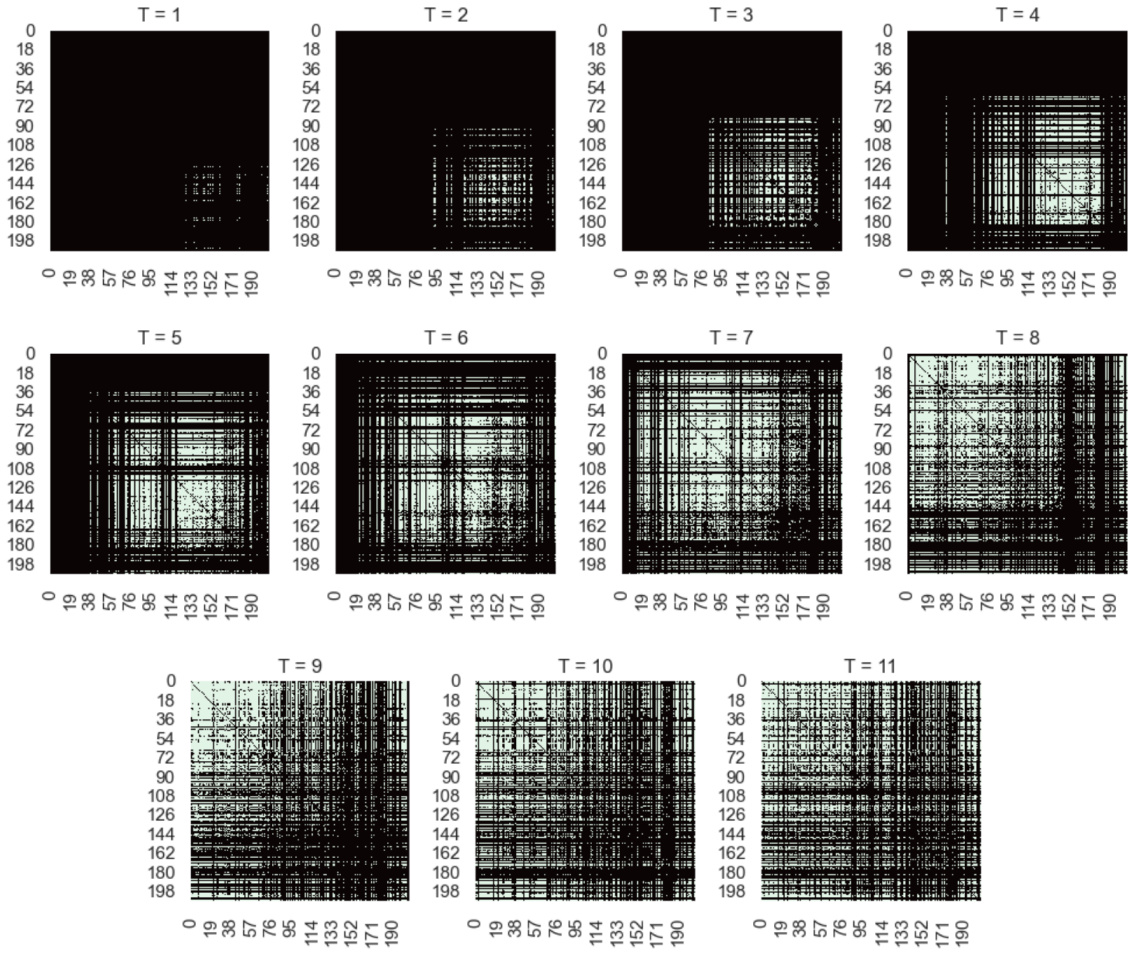


Figure B.1: 207 Dataset

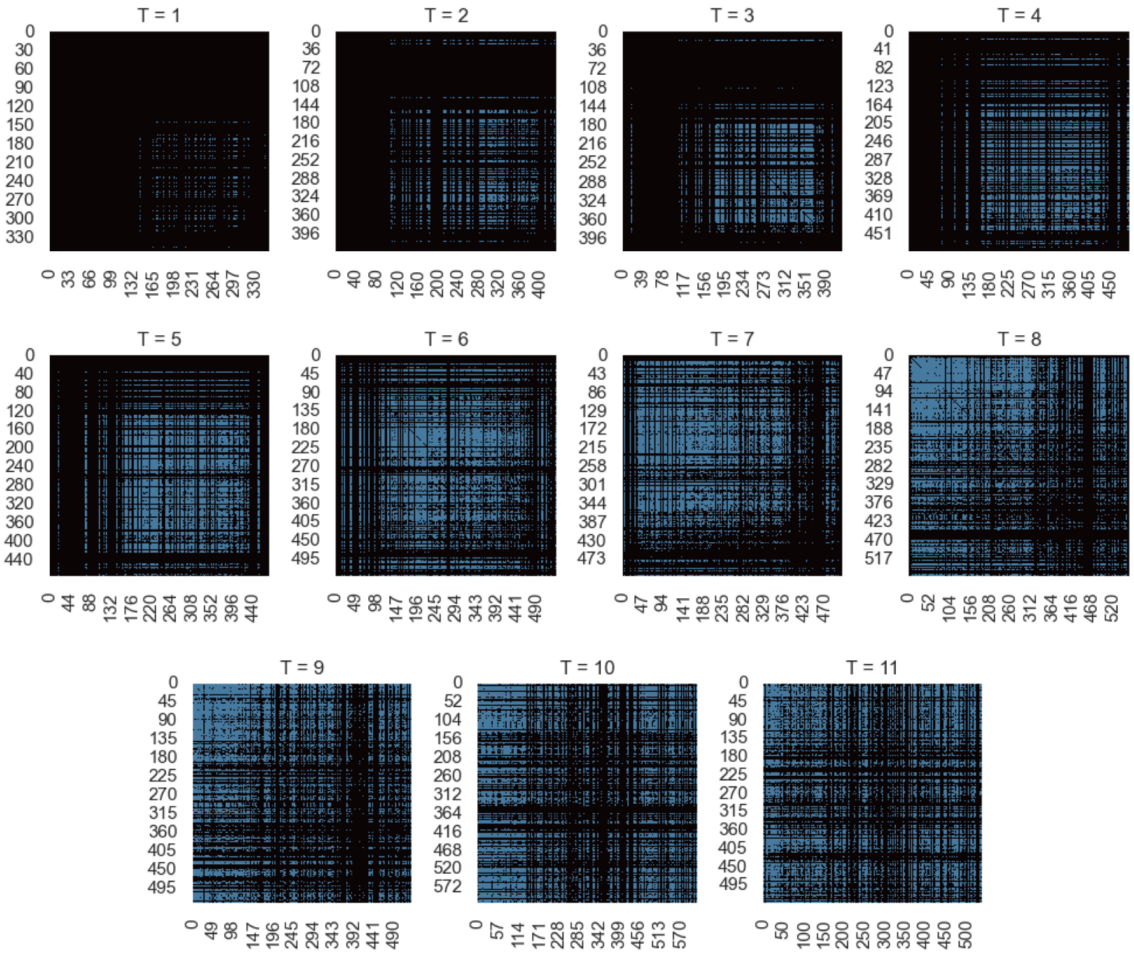
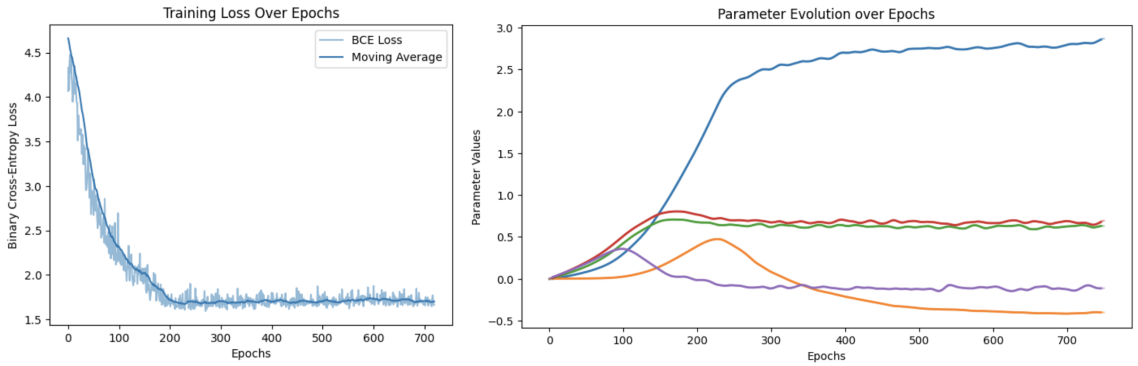


Figure B.2: 616 Dataset

B.2 Neural Calibration Training Loss Curves

Figure 4.5 contains the loss curves and parameter evolutions for the NN model trained on both the 207 and 616 datasets. Here we provide the loss curves for the CNN and GNN models trained on the 207 dataset.

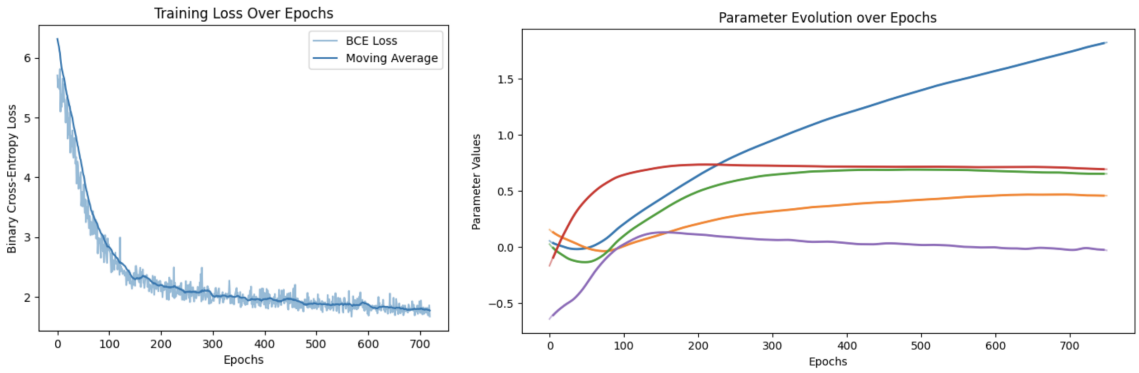
Note: The parameter evolution plots for both CNN and GNN are smoother than for NN because of the use of an Adam optimiser for the former and RMSProp for the latter.



(a) Training Loss Curve for 207 Model

(b) Parameter Evolution in 207 training instance

Figure B.3: Single Training Instances of CNN for the 207 Model. Darker lines corresponding to moving averages.



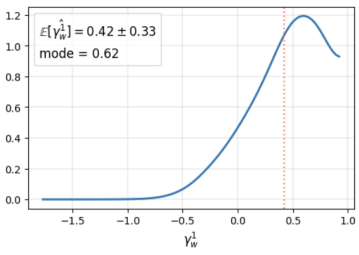
(a) Training Loss Curve for 207 Model

(b) Parameter Evolution in 207 training instance

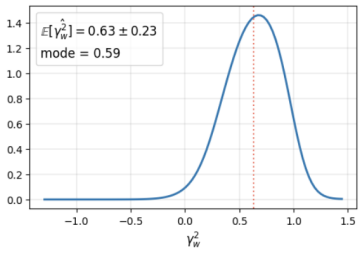
Figure B.4: Single Training Instances of GNN for the 207 Model. Darker lines corresponding to moving averages.

B.3 Density Estimations

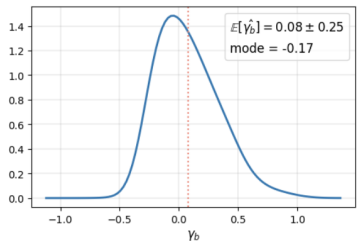
In Section 4.5.1 we plot the 5 PDF estimates for the 207 GNN model over 30 training instances. We now provide the remaining PDF estimates for the 207 NN and CNN models as well as the 616 NN model.



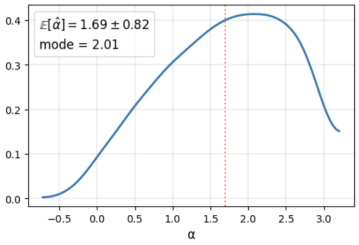
(a) γ_w^1 Estimate



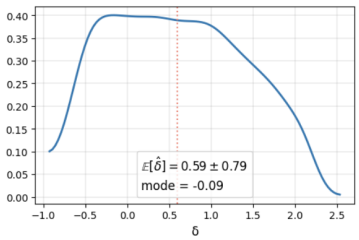
(b) γ_w^2 Estimate



(c) γ_b Estimate

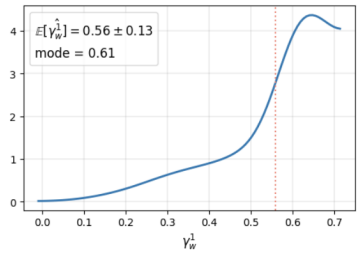


(d) α Estimate

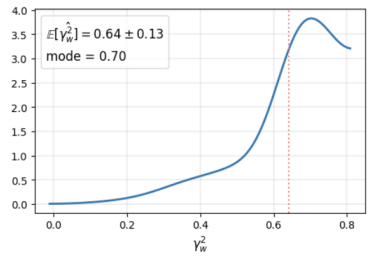


(e) δ Estimate

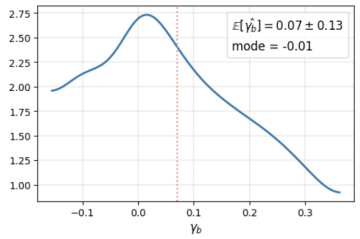
Figure B.5: Parameter PDF estimates for 207 NN model over 30 training instances.



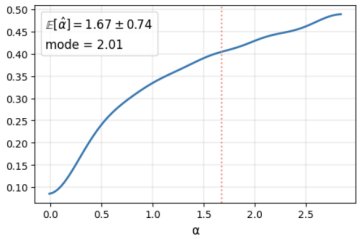
(a) γ_w^1 Estimate



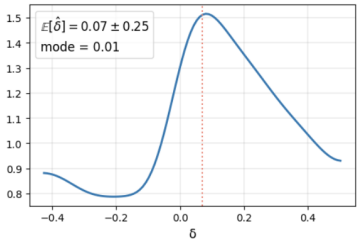
(b) γ_w^2 Estimate



(c) γ_b Estimate

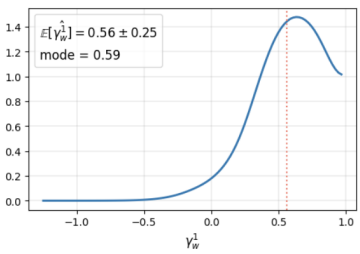


(d) α Estimate

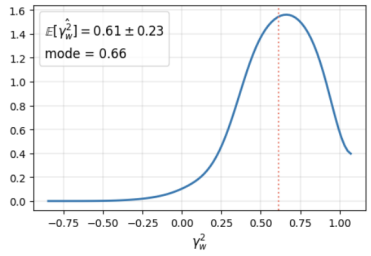


(e) δ Estimate

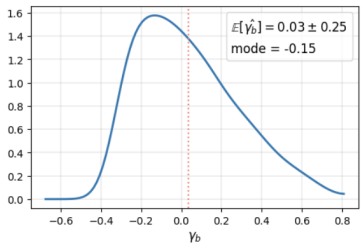
Figure B.6: Parameter PDF estimates for 207 CNN model over 30 training instances.



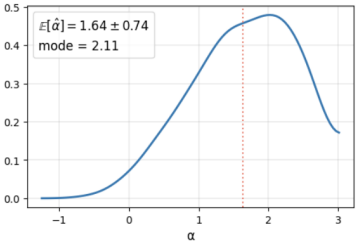
(a) γ_w^1 Estimate



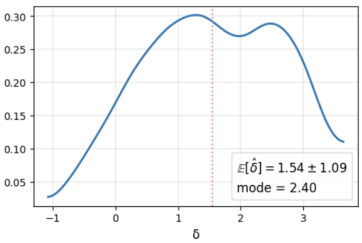
(b) γ_w^2 Estimate



(c) γ_b Estimate



(d) α Estimate



(e) δ Estimate

Figure B.7: Parameter PDF estimates for 616 NN model over 30 training instances.

B.4 Latent Position Evolution

Figure 4.8 presents the latent space at discrete time steps for the MCMC, SGD, and NN models on the 207 dataset. In this section we provide the remaining 207 plots for CNN and GNN as well as SGD/NN plots for the 616 dataset.

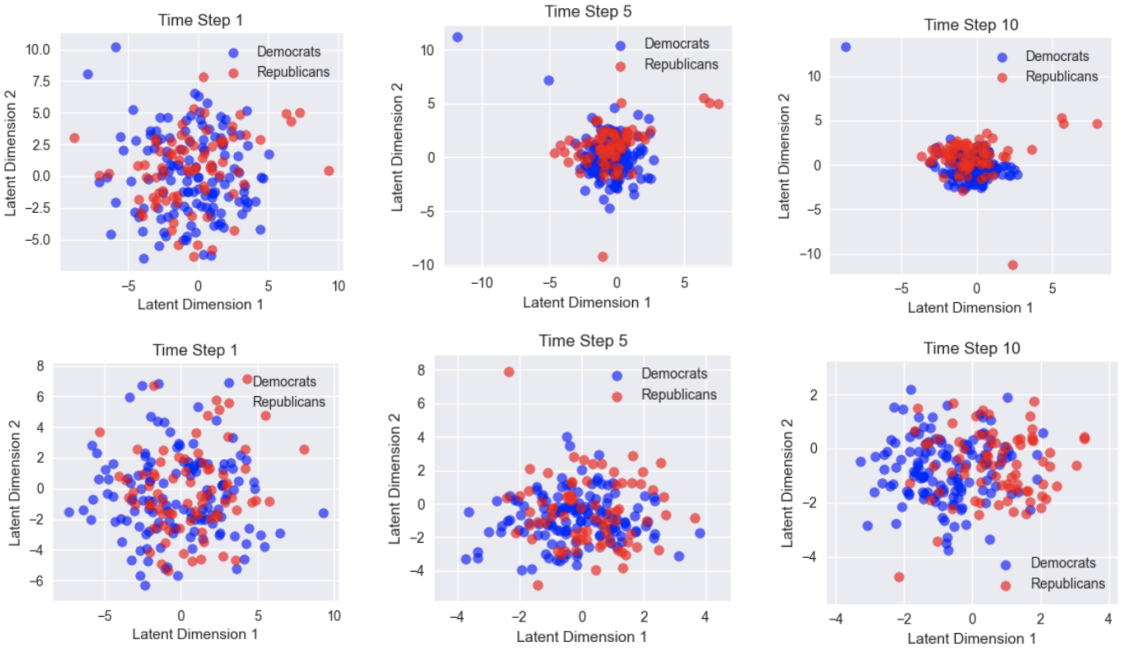


Figure B.8: Latent space evolution at time steps $T = \{1, 5, 10\}$ for **CNN** (Row 1) and **GNN** (Row 2) for **207** dataset.

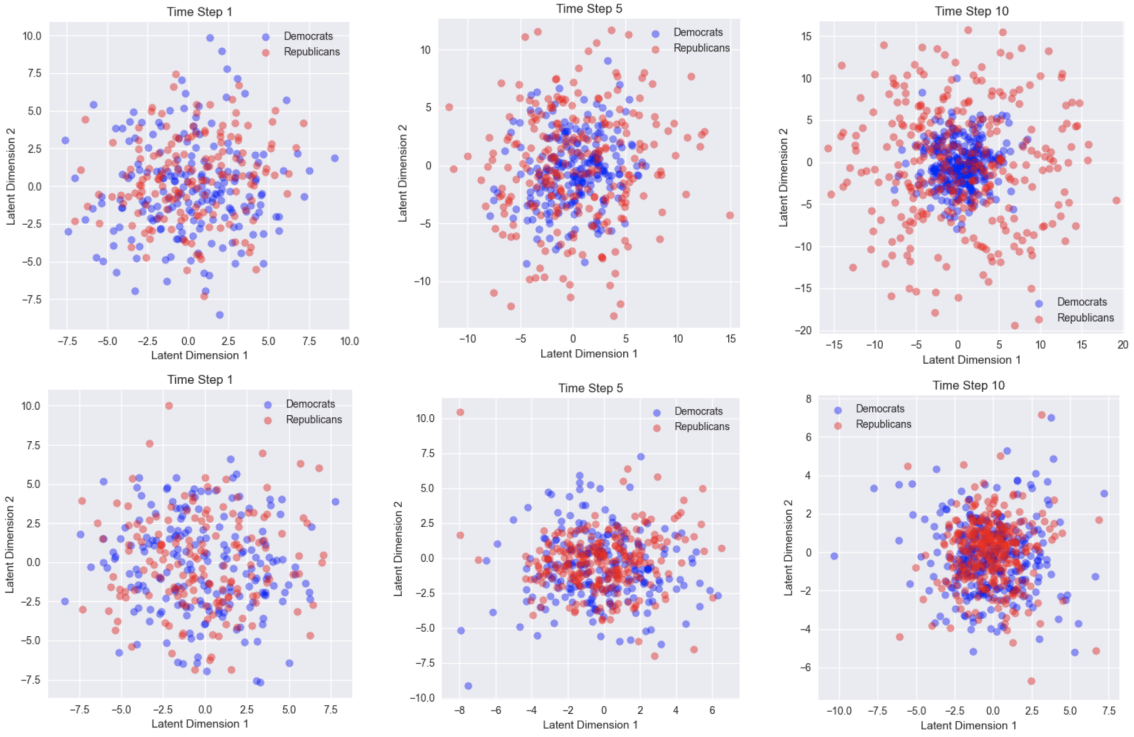
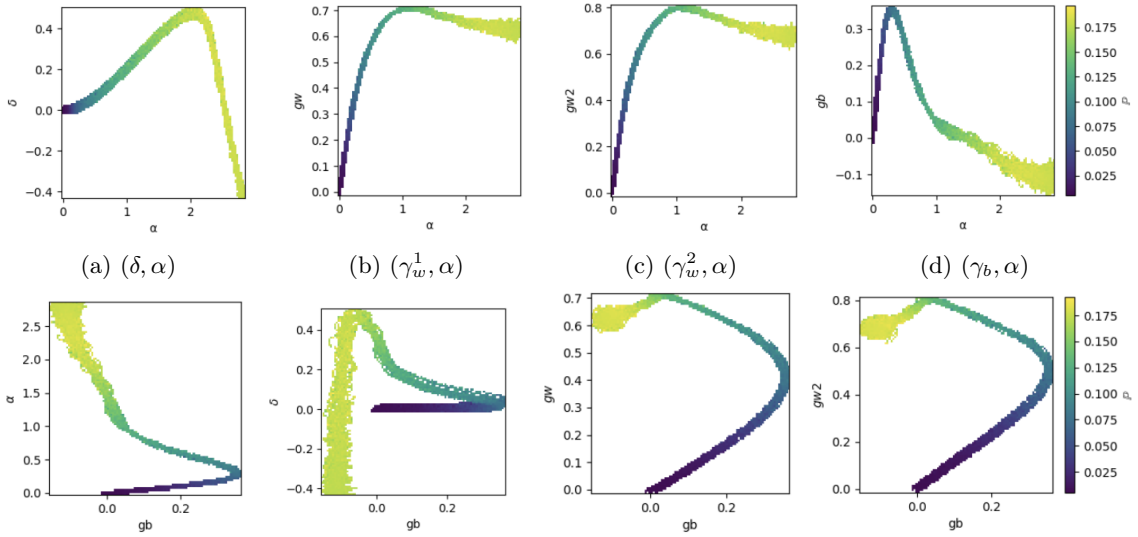


Figure B.9: Latent space evolution at time steps $T = \{1, 5, 10\}$ for **SGD** (Row 1) and **NN** (Row 2) for **616** dataset.

B.5 Joint 2D Densities

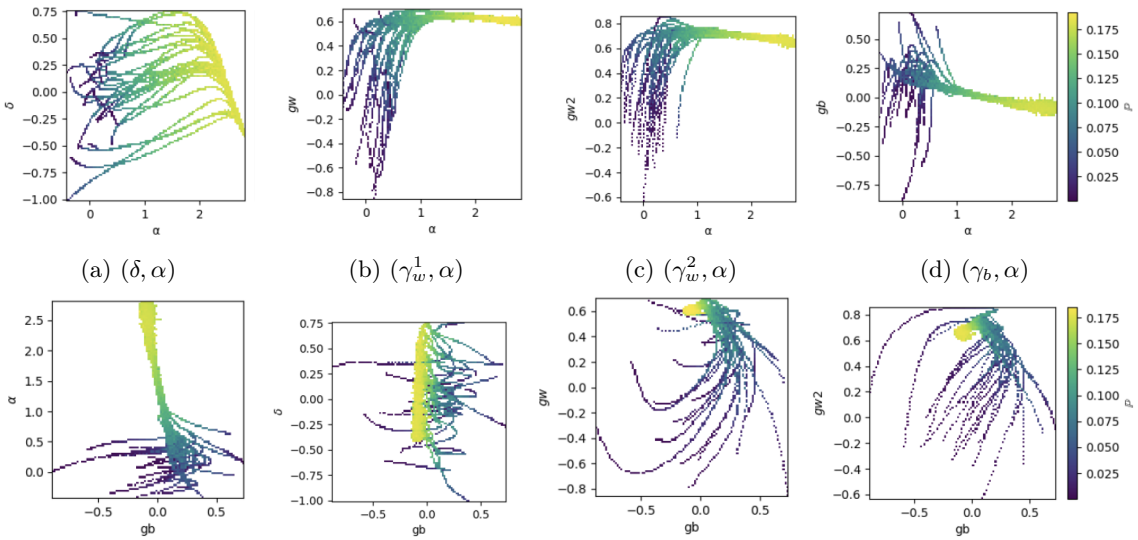
To finish this chapter, we provide a full set of 2D joint density estimates for the CNN and GNN models trained on the 207 dataset.



(a) (δ, α) (b) (γ_w^1, α) (c) (γ_w^2, α) (d) (γ_b, α)

(e) (α, γ_b) (f) (δ, γ_b) (g) (γ_w^1, γ_b) (h) (γ_w^2, γ_b)

Figure B.10: 2D Joint Density estimates for all combinations of α and γ_b in 207 CNN model.



(a) (δ, α) (b) (γ_w^1, α) (c) (γ_w^2, α) (d) (γ_b, α)

(e) (α, γ_b) (f) (δ, γ_b) (g) (γ_w^1, γ_b) (h) (γ_w^2, γ_b)

Figure B.11: 2D Joint Density estimates for all combinations of α and γ_b in 207 GNN model.

Appendix C

HEBO: Heteroscedastic Evolutionary Bayesian Optimisation

A key component in any data science project is identifying a configuration of case-specific hyperparameters that produce the best results. Approaches to do this are widely studied in literature, ranging from simple hyperspace grid searches to sophisticated Bayesian optimisation solutions. In this section, we present in detail HEBO or Heteroscedastic Evolutionary Bayesian Optimisation [53], which we used to perform the hyperparameter search for the neural calibration models trained on both the 207 and 616 datasets.

In Bayesian Optimisation (BO), the goal is to find $\mathbf{x}^* = \arg\max f(\mathbf{x})$ without any prior knowledge of both the smooth function $f(\mathbf{x})$ and the optimal $\mathbf{x}^*$. Algorithms must balance the trade-off between exploiting regions of the hyperparameter space that have been previously evaluated and exploring new, unevaluated regions. Typically, BO methods comprise two main elements: a probabilistic surrogate model, which incorporates priors derived from existing observations, and an acquisition function, which directs the selection of new points for evaluation.

Because of idealised assumptions about the nature of f, many algorithms struggle to handle noisy surrogates and may become trapped in local minima due to a singular acquisition objective. To overcome these challenges, HEBO introduced a surrogate model that employs warping and transformation in the input space, enhancing the model's adaptability and resistance to noise. HEBO's surrogate model is a Gaussian process with input warping and transformations applied post hoc, represented by $g(\mathbf{x}) = h(f(\mathbf{x}))$, where h is a transformation applied to the original function f. This transformation enables the surrogate model to capture heteroscedasticity (situations where the variance of the residuals varies) present in the data. The transformed surrogate model is subsequently used to predict the mean and variance of the objective function at new points, which are crucial for the acquisition function.

For exploring new solutions, HEBO conducts multi-objective optimisation over the Pareto front of solutions, integrating three standard acquisition functions: Expected Improvement (EI), Probability of Improvement (PI), and Upper Confidence Bound (UCB). This multi-objective strategy uses the NSGA-II algorithm (the same algorithm applied for the MOO in the SA of neural calibration models).

Appendix D

Neural Calibration as a VAE: Auxiliary Plots

D.1 `simulate_clsna_dynamic` Saturation

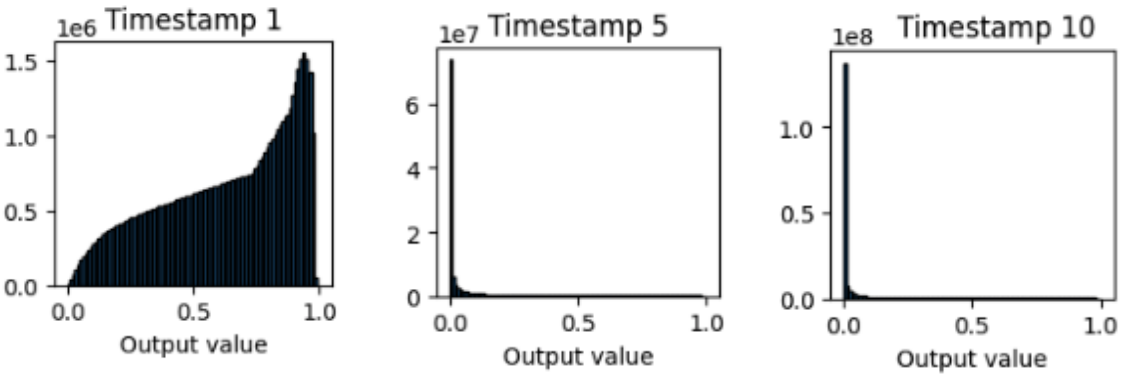


Figure D.1: `simulate_clsna_dynamic` output histograms at time steps $T = \{1, 5, 10\}$

D.2 VAE Loss Curves

Figure 5.5 presents the total, reconstruction, and KL losses of the VGAE trained on the 207 dataset. Here we provide the same plots for $(\text{NN-VAE/CNN-VAE})_{207}$ and NN-VAE_{616}.

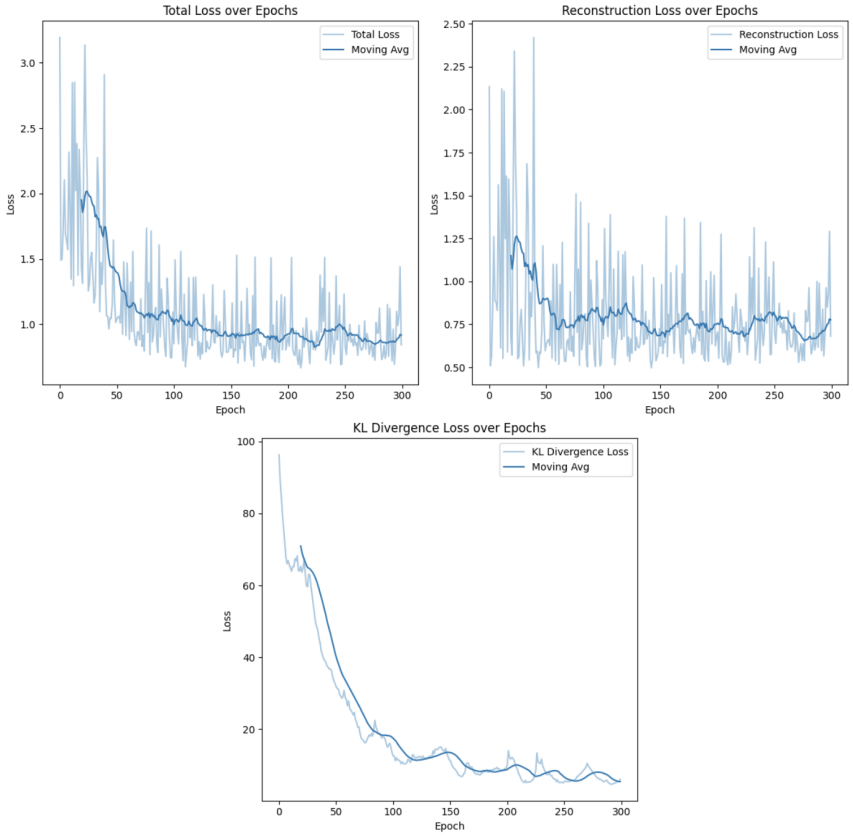


Figure D.2: Total, Reconstruction, and KL Loss Curves for NN-VAE on 207 dataset.

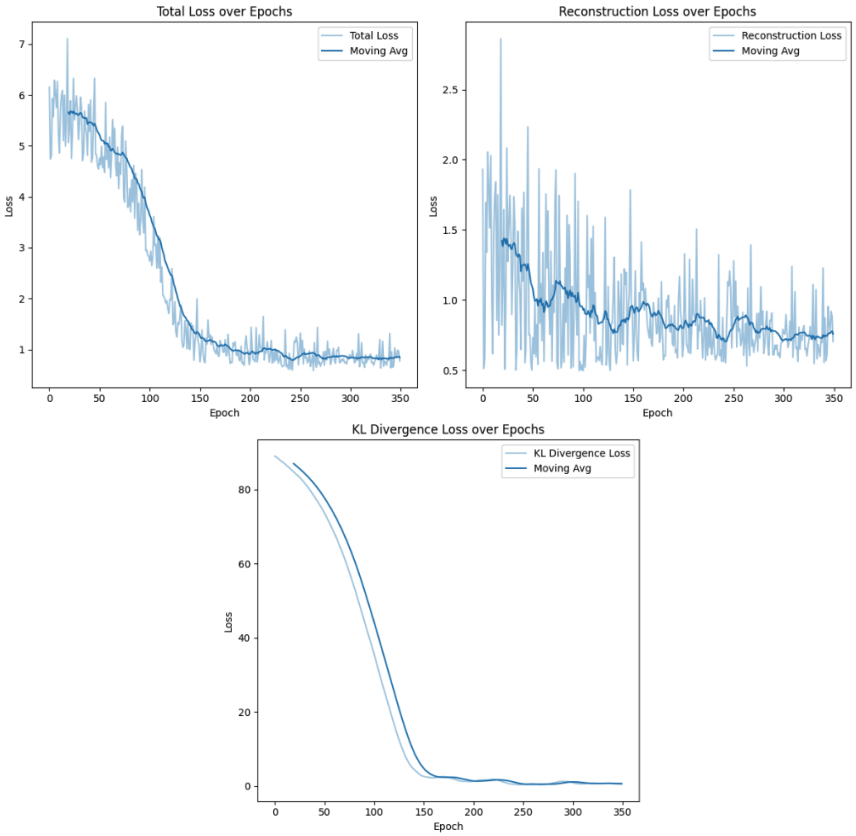


Figure D.3: Total, Reconstruction, and KL Loss Curves for CNN-VAE on 207 dataset.

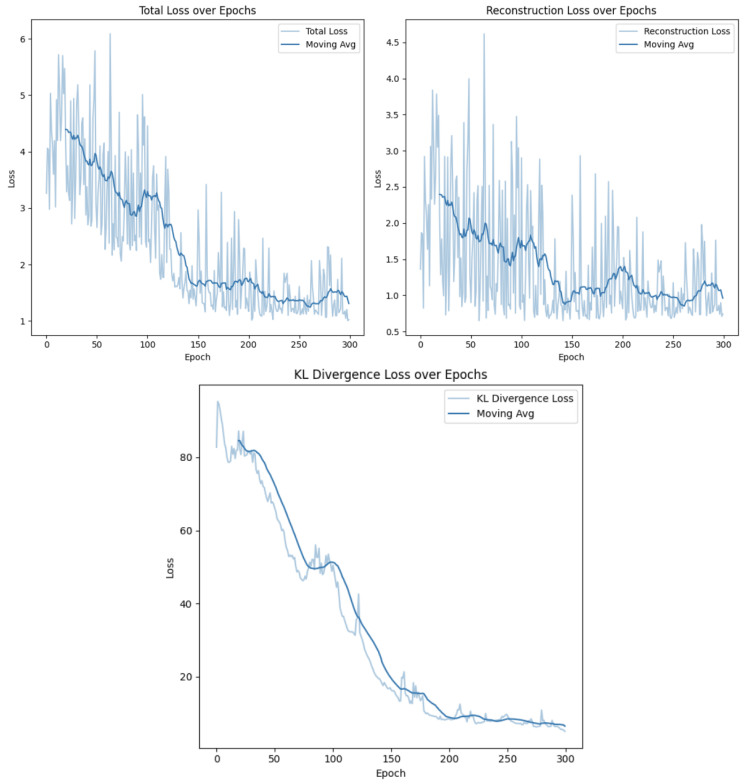
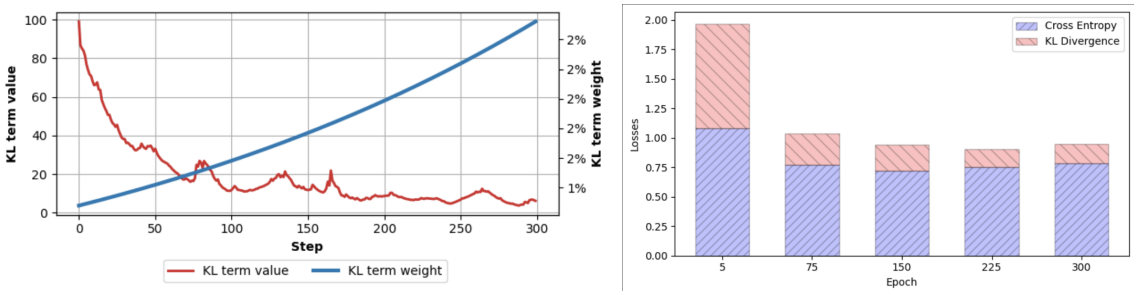


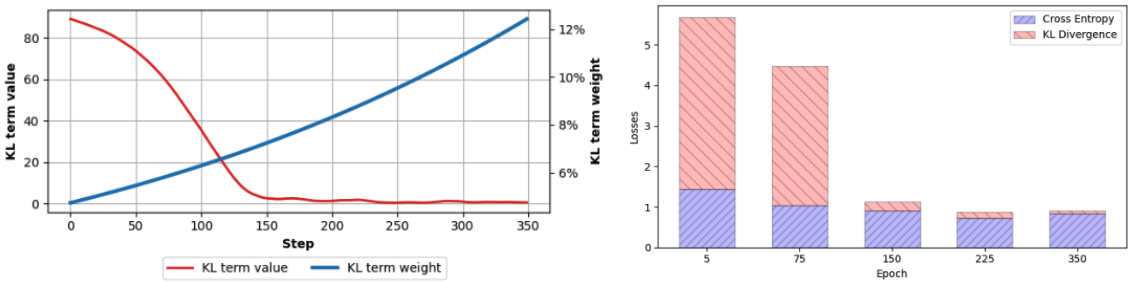
Figure D.4: Total, Reconstruction, and KL Loss Curves for NN-VAE on 616 dataset.

D.3 KL Annealing and Deconstructed VAE Loss



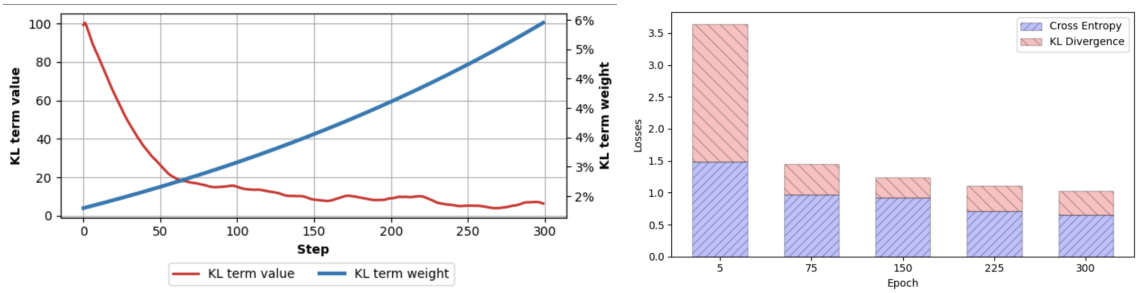
(a) Annealed KL Divergence Loss (b) Grouped KL and Reconstruction Losses

Figure D.5: KL Annealing effects on training loss of NN-VAE on 207 dataset.



(a) Annealed KL Divergence Loss (b) Grouped KL and Reconstruction Losses

Figure D.6: KL Annealing effects on training loss of CNN-VAE on 207 dataset.



(a) Annealed KL Divergence Loss

(b) Grouped KL and Reconstruction Losses

Figure D.7: KL Annealing effects on training loss of VGAE on 207 dataset.

D.4 Posterior Updates

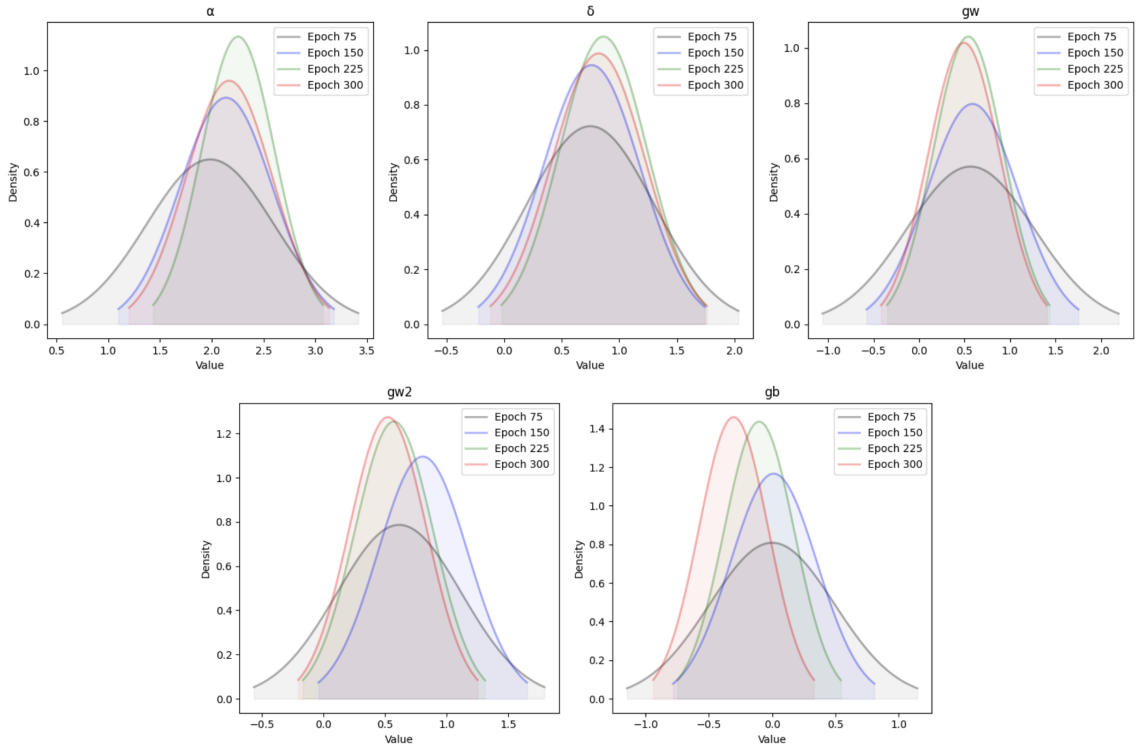


Figure D.8: Posterior updates of parameters in NN-VAE trained on 207 dataset.

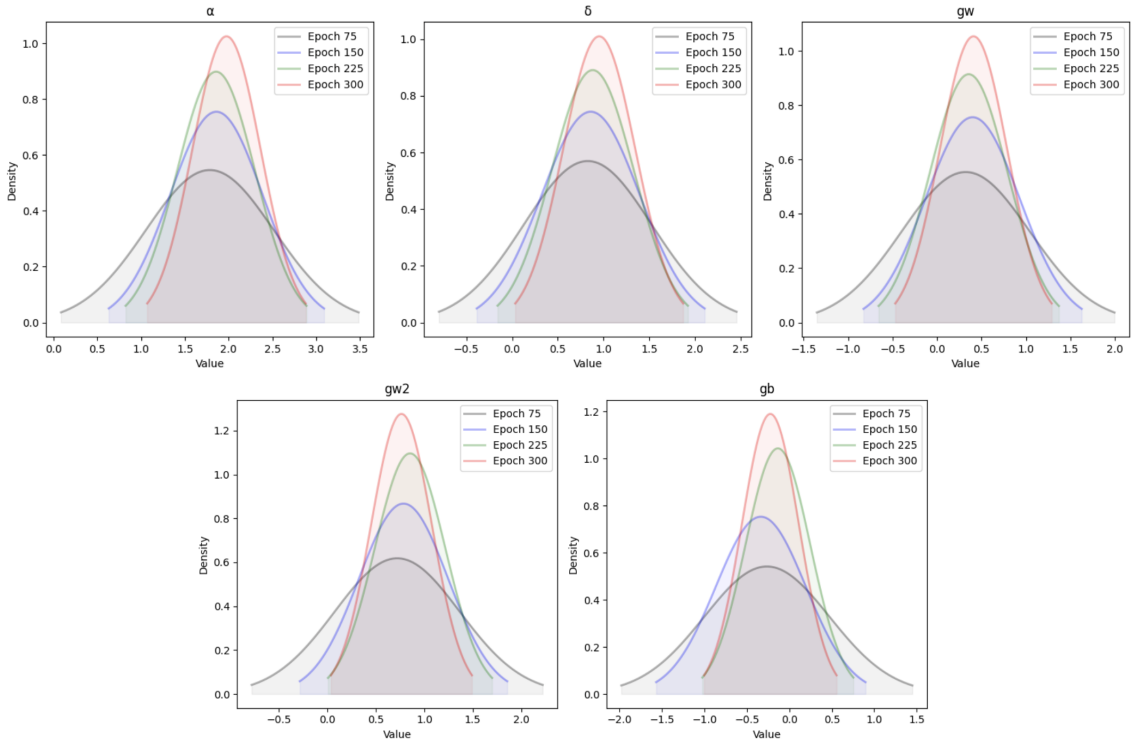


Figure D.9: Posterior updates of parameters in VGAE trained on 207 dataset.

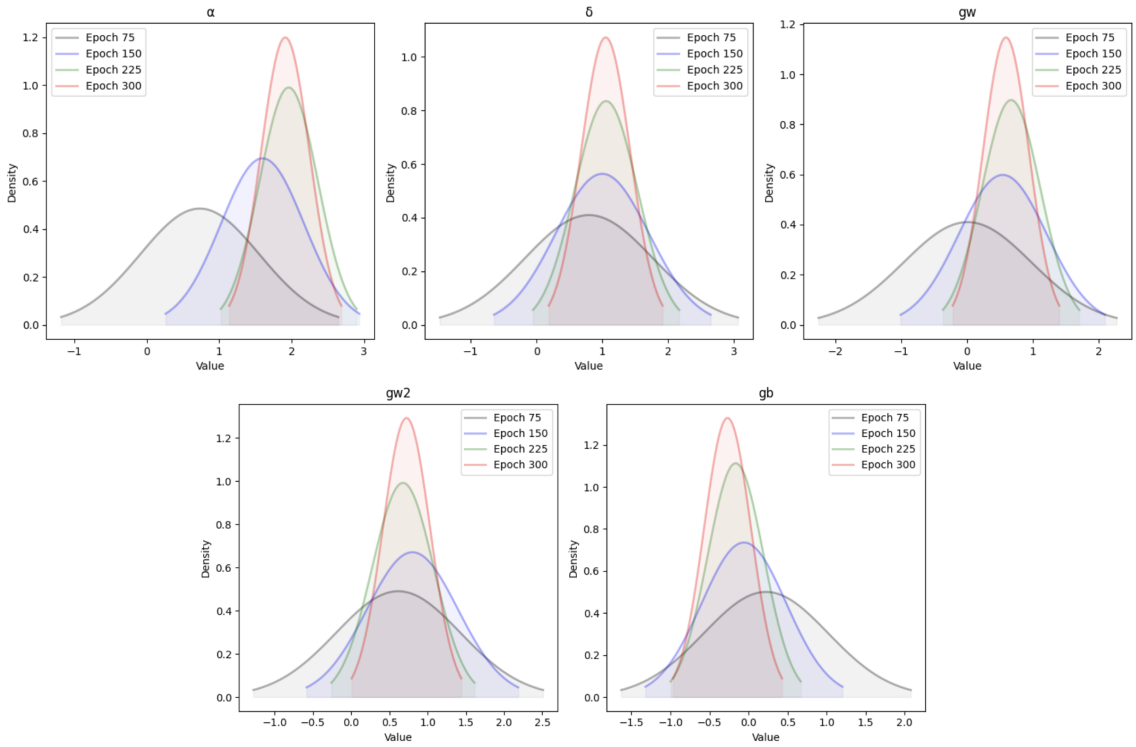


Figure D.10: Posterior updates of parameters in NN-VAE trained on 616 dataset.

D.5 VAE and Neural Calibration Distribution Comparisons

The full-set of figures comparing NN/CNN VAE and VGAE to all neural calibration models would consist of 10 pairs. Figure 5.8 plots CNN-VAE and basic NN neural calibration. In this section we will plot only three more cases of interest: $(\text{VGAE, GNN})_{207}$, $(\text{NN-VAE, NN})_{207}$, and lastly

$(\text{VAE, NN})_{616}$.

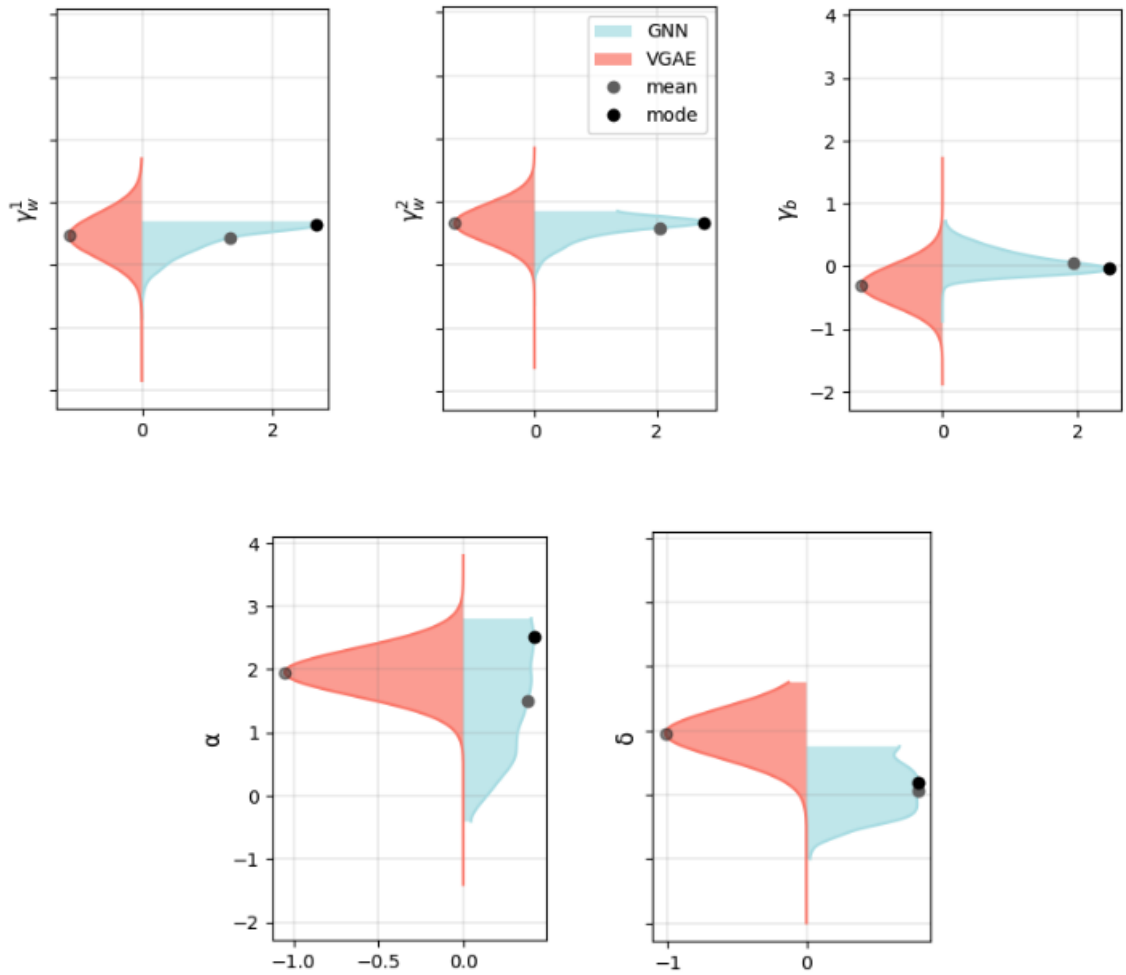


Figure D.11: Marginals for parameters $\lambda = \{\gamma_w^1, \gamma_w^2, \gamma_b, \alpha, \delta\}$ for both VGAE calibration with factorised Gaussian posteriors (red, left side) and GNN neural calibration (blue, right side) on 207 dataset. Black dots correspond to distribution modes and grey dots to means.

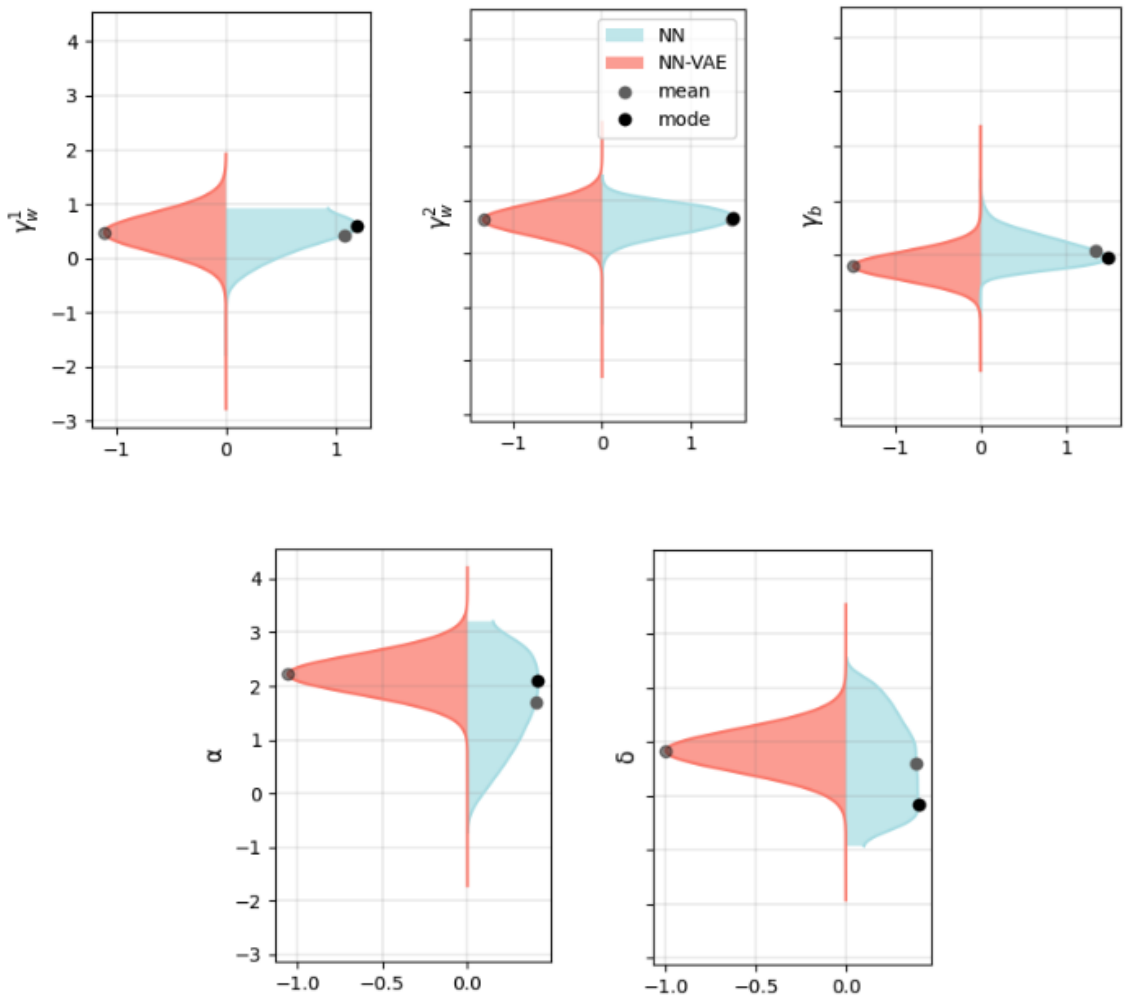


Figure D.12: Marginals for parameters $\lambda = \{\gamma_w^1, \gamma_w^2, \gamma_b, \alpha, \delta\}$ for both NN-VAE calibration with factorised Gaussian posteriors (red, left side) and NN neural calibration (blue, right side) on 207 dataset. Black dots correspond to distribution modes and grey dots to means.

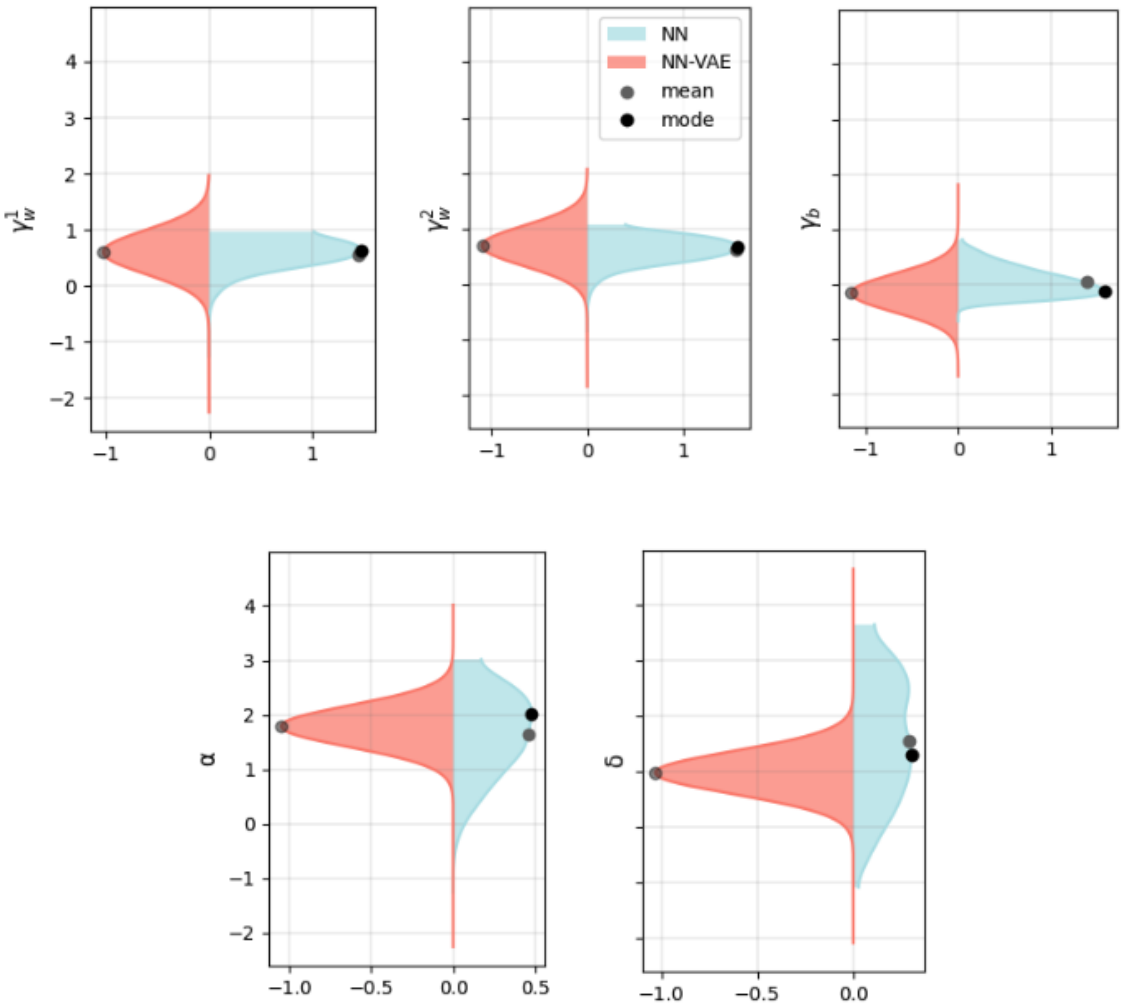


Figure D.13: Marginals for parameters $\lambda = \{\gamma_w^1, \gamma_w^2, \gamma_b, \alpha, \delta\}$ for both NN-VAE calibration with factorised Gaussian posteriors (red, left side) and NN neural calibration (blue, right side) on 616 dataset. Black dots correspond to distribution modes and grey dots to means.

Bibliography

[1] Antonio F. Peralta, János Kertész, and Gerardo Iñiguez. Opinion dynamics in social networks: From models to data. *arXiv preprint arXiv:2201.01322*, December 2022. doi: 10.48550/arXiv.2201.01322. URL http://arxiv.org/abs/2201.01322.

[2] Eytan Bakshy, Jake M. Hofman, Winter A. Mason, and Duncan J. Watts. Everyone's an influencer: quantifying influence on twitter. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 65–74, Hong Kong China, February 2011. ACM. ISBN 978-1-4503-0493-1. doi: 10.1145/1935826.1935845. URL https://dl.acm.org/doi/10.1145/1935826.1935845.

[3] Sandra González-Bailón. Social science in the era of big data. *Policy & Internet*, 5(2): 147–160, June 2013. ISSN 1944-2866, 1944-2866. doi: 10.1002/1944-2866.POI328. URL https://onlinelibrary.wiley.com/doi/10.1002/1944-2866.POI328.

[4] Xiaojing Zhu, Cantay Caliskan, Dino P. Christenson, Konstantinos Spiliopoulos, Dylan Walker, and Eric D. Kolaczyk. Disentangling positive and negative partisanship in social media interactions using a coevolving latent space network with attractors model. August 2022. doi: 10.1093/jrsssa/qnad008. URL http://arxiv.org/abs/2109.13129. arXiv:2109.13129 [stat].

[5] Hancong Pan, Xiaojing Zhu, Cantay Caliskan, Dino P. Christenson, Konstantinos Spiliopoulos, Dylan Walker, and Eric D. Kolaczyk. Stochastic gradient descent-based inference for dynamic network models with attractors. March 2024. doi: 2403.07124/arXiv:2403.07124v2. URL http://arxiv.org/abs/2403.07124. arXiv:2403.07124 [cs, stat].

[6] Grigorios A. Pavliotis and Andrea Zanoni. Eigenfunction martingale estimators for interacting particle systems and their mean field limit. *arXiv preprint arXiv:2112.04870*, January 2024. doi: 10.48550/arXiv.2112.04870. URL http://arxiv.org/abs/2112.04870.

[7] Thomas Gaskin, Grigorios A. Pavliotis, and Mark Girolami. Neural parameter calibration for large-scale multiagent models. *Proceedings of the National Academy of Sciences*, 120(7): e2216415120, February 2023. ISSN 0027-8424, 1091-6490. doi: 10.1073/pnas.2216415120. URL https://pnas.org/doi/10.1073/pnas.2216415120.

[8] Thomas Gaskin, Tim Conrad, Grigorios A. Pavliotis, and Christof Schütte. Neural parameter calibration and uncertainty quantification for epidemic forecasting. December 2023. URL http://arxiv.org/abs/2312.03147. arXiv:2312.03147 [physics].

[9] Thomas Gaskin, Grigorios A Pavliotis, and Mark Girolami. Inferring networks from time series: A neural approach. *PNAS Nexus*, 3(4):pgae063, March 2024. ISSN 2752-6542. doi: 10.1093/pnasnexus/pgae063. URL https://academic.oup.com/pnasnexus/article/doi/10.1093/pnasnexus/pgae063/7604085.

[10] Twitter developer agreement and policy. https://developer.x.com/en/developer-terms/agreement-and-policy. Accessed: 2024-05-30, Last updated: 2023-11-14.

[11] European Union. General Data Protection Regulation (GDPR) - Chapter 1, Article 4. https://gdpr-info.eu/, 2016. Accessed: 2024-06-04. Official Journal of the European Union, L 119, 04.05.2016; corrigendum L 127, 23.5.2018.

[12] Eleonora Bertoni, Matteo Fontana, Lorenzo Gabrielli, Serena Signorelli, and Michele Vespe, editors. *Handbook of Computational Social Science for Policy*. Springer International Publishing, Cham, 2023. ISBN 978-3-031-16623-5 978-3-031-16624-2. doi: 10.1007/978-3-031-16624-2. URL https://link.springer.com/10.1007/978-3-031-16624-2.

[13] Corrado Monti, Marco Pangallo, Gianmarco De Francisci Morales, and Francesco Bonchi. On learning agent-based models from data. *Scientific Reports*, 13(1):9268, June 2023. ISSN 2045-2322. doi: 10.1038/s41598-023-35536-3. URL https://www.nature.com/articles/s41598-023-35536-3.

[14] Natasa Djurdjevac Conrad, Jonas Köppl, and Ana Djurdjevac. Feedback Loops in Opinion Dynamics of Agent-Based Models with Multiplicative Noise. *Entropy*, 24(10):1352, September 2022. ISSN 1099-4300. doi: 10.3390/e24101352. URL http://arxiv.org/abs/2209.07151. arXiv:2209.07151 [cs, math].

[15] Rainer Hegselmann and Ulrich Krause. Opinion dynamics and bounded confidence models, analysis and simulation. *Journal of Artificial Societies and Social Simulation*, 5(3), 07 2002. URL https://www.jasss.org/5/3/2/2.pdf.

[16] Qinyue Zhou, Zhibin Wu, Abdulrahman H. Altalhi, and Francisco Herrera. A two-step communication opinion dynamics model with self-persistence and influence index for social networks based on the DeGroot model. *Information Sciences*, 519(519):363–381, May 2020. ISSN 00200255. doi: 10.1016/j.ins.2020.01.052. URL https://linkinghub.elsevier.com/retrieve/pii/S0020025520300633.

[17] Zhaogang Ding, Xia Chen, Yucheng Dong, and Francisco Herrera. Consensus reaching in social network DeGroot Model: The roles of the Self-confidence and node degree, June 2019. ISSN 00200255. URL https://linkinghub.elsevier.com/retrieve/pii/S0020025519301380.

[18] Alan I. Abramowitz and Steven W. Webster. Negative Partisanship: Why Americans Dislike Parties But Behave Like Rabid Partisans. *Political Psychology*, 39(S1):119–135, February 2018. ISSN 0162-895X, 1467-9221. doi: 10.1111/pops.12479. URL https://onlinelibrary.wiley.com/doi/10.1111/pops.12479.

[19] D.J. Bartholomew. Factor analysis and latent structure: Overview. In Neil J. Smelser and Paul B. Baltes, editors, *International Encyclopedia of the Social and Behavioral Sciences*, pages 5249–5254. Pergamon, Oxford, 2001. ISBN 978-0-08-043076-8. doi: https://doi.org/10.1016/B0-08-043076-7/00425-3. URL https://www.sciencedirect.com/science/article/pii/B0080430767004253.

[20] Christian P. Robert. The metropolis-hastings algorithm. *arXiv preprint arXiv:1504.01896*, Jan 2016. doi: 10.48550/arXiv.1504.01896. URL http://arxiv.org/abs/1504.01896.

[21] L. F. Burgette and J. P. Reiter. Multiple Imputation for Missing Data via Sequential Regression Trees. *American Journal of Epidemiology*, 172(9):1070–1076, November 2010. ISSN 0002-9262, 1476-6256. doi: 10.1093/aje/kwq260. URL https://academic.oup.com/aje/article-lookup/doi/10.1093/aje/kwq260.

[22] M. Alan Brookhart, Sebastian Schneeweiss, Kenneth J. Rothman, Robert J. Glynn, Jerry Avorn, and Til Stürmer. Variable Selection for Propensity Score Models. *American Journal of Epidemiology*, 163(12):1149–1156, June 2006. ISSN 1476-6256, 0002-9262. doi: 10.1093/aje/kwj149. URL http://academic.oup.com/aje/article/163/12/1149/97130/Variable-Selection-for-Propensity-Score-Models.

[23] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, 3(5):551–560, January 1990. ISSN 08936080. doi: 10.1016/0893-6080(90)90005-6. URL https://linkinghub.elsevier.com/retrieve/pii/0893608090900056.

[24] M. Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge, MA, 1969. URL https://mitpress.mit.edu/9780262630221/perceptrons/.

[25] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986. URL https://www.nature.com/articles/323533a0.

[26] Automatic differentiation with torch.autograd - pytorch tutorials 2.3.0+cu121 documentation. URL https://pytorch.org/tutorials/beginner/basics/autogradqs_tutorial.html. Version 2.3.0+cu121.

[27] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, R. Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989. URL https://proceedings.neurips.cc/paper_files/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf.

[28] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9(4):611–629, August 2018. ISSN 1869-4101. doi: 10.1007/s13244-018-0639-9. URL https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9.

[29] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. Graph convolutional networks: a comprehensive review. *Computational Social Networks*, 6(1):11, December 2019. ISSN 2197-4314. doi: 10.1186/s40649-019-0069-y. URL https://computationalsocialnetworks.springeropen.com/articles/10.1186/s40649-019-0069-y.

[30] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, February 2017. doi: 10.48550/arXiv.1609.02907. URL http://arxiv.org/abs/1609.02907. Published as a conference paper at ICLR 2017.

[31] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark. June 2022. URL http://arxiv.org/abs/2109.14545. arXiv:2109.14545 [cs].

[32] Anna Rakitianskaia and Andries Engelbrecht. Measuring Saturation in Neural Networks. In *2015 IEEE Symposium Series on Computational Intelligence*, pages 1423–1430, Cape Town, South Africa, December 2015. IEEE. ISBN 978-1-4799-7560-0. doi: 10.1109/SSCI.2015.202. URL http://ieeexplore.ieee.org/document/7376778/.

[33] PyTorch. torch.nn.bceloss, 2023. URL https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html. Accessed: 2024-06-18, Version 2.3.0+cu121.

[34] Mateusz Buda, Atsuto Maki, and Maciej A. Mazurowski. A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks*, 106:249–259, October 2018. ISSN 08936080. doi: 10.1016/j.neunet.2018.07.011. URL http://arxiv.org/abs/1710.05381. arXiv:1710.05381 [cs, stat].

[35] Jari Saramäki, Mikko Kivelä, Jukka-Pekka Onnela, Kimmo Kaski, and János Kertész. Generalizations of the clustering coefficient to weighted complex networks. *Physical Review E*, 75 (2):027105, February 2007. ISSN 1539-3755, 1550-2376. doi: 10.1103/PhysRevE.75.027105. URL https://link.aps.org/doi/10.1103/PhysRevE.75.027105.

[36] Linton C. Freeman. Centrality in social networks conceptual clarification. *Social Networks*, 1 (3):215–239, January 1978. ISSN 03788733. doi: 10.1016/0378-8733(78)90021-7. URL https://linkinghub.elsevier.com/retrieve/pii/0378873378900217.

[37] A. Saltelli, editor. *Sensitivity analysis in practice: a guide to assessing scientific models*. Wiley, Hoboken, NJ, 2004. ISBN 978-0-470-87093-8. URL http://www.andreasaltelli.eu/file/repository/SALTELLI_2004_Sensitivity_Analysis_in_Practice.pdf.

[38] Livia Paleari, Ermes Movedi, Michele Zoli, Andrea Burato, Irene Cecconi, Jabir Errahouly, Eleonora Pecollo, Carla Sorvillo, and Roberto Confalonieri. Sensitivity analysis using Morris: Just screening or an effective ranking method? *Ecological Modelling*, 455:109648, September 2021. ISSN 03043800. doi: 10.1016/j.ecolmodel.2021.109648. URL https://linkinghub.elsevier.com/retrieve/pii/S0304380021002088.

[39] I.M Sobol. Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates. *Mathematics and Computers in Simulation*, 55(1-3):271–280, February 2001. ISSN 03784754. doi: 10.1016/S0378-4754(00)00270-6. URL https://linkinghub.elsevier.com/retrieve/pii/S0378475400002706.

[40] Matthias Ehrgott. *Multicriteria optimization*. Springer, Berlin ; New York, 2nd ed edition, 2005. ISBN 978-3-540-21398-7. URL https://link.springer.com/book/10.1007/3-540-27659-9.

[41] N. Srinivas and Kalyanmoy Deb. Muiltiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, 2(3):221–248, September 1994. ISSN 1063-6560, 1530-9304. doi: 10.1162/evco.1994.2.3.221. URL https://direct.mit.edu/evco/article/2/3/221-248/1396.

[42] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002. ISSN 1089778X. doi: 10.1109/4235.996017. URL http://ieeexplore.ieee.org/document/996017/.

[43] Albert-László Barabási and Reka Albert. Emergence of scaling in random networks. *Science*, 286:509, Oct 1999. doi: 10.1126/science.286.5439.509. URL https://arxiv.org/abs/cond-mat/9910332. arXiv:cond-mat/9910332 [cond-mat.dis-nn].

[44] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012. URL https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[45] Thomas Gaskin and Joerg Wangemann. NeuralABM github repository. https://github.com/ThGaskin/NeuralABM, 2024. Last Accessed: 26-05-2024.

[46] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, Jan 2017. doi: 10.48550/arXiv.1412.6980. URL http://arxiv.org/abs/1412.6980. Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[47] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *arXiv preprint arXiv:1505.04597*, May 2015. doi: 10.48550/arXiv.1505.04597. URL http://arxiv.org/abs/1505.04597. Conditionally accepted at MICCAI 2015.

[48] Megi Isallari and Islem Rekik. Brain Graph Super-Resolution Using Adversarial Graph Neural Network with Application to Functional Brain Connectivity. May 2021. URL http://arxiv.org/abs/2105.00425. arXiv:2105.00425 [cs, eess, q-bio].

[49] Mahdi Hashemi. Enlarging smaller images before inputting into convolutional neural network: zero-padding vs. interpolation. *Journal of Big Data*, 6(1):98, December 2019. ISSN 2196-1115. doi: 10.1186/s40537-019-0263-7. URL https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0263-7.

[50] Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In *Neural Information Processing Systems*, 1991. URL https://api.semanticscholar.org/CorpusID:10137788.

[51] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[52] Anne Randi Syversveen. Noninformative bayesian priors: Interpretation and problems with construction and applications. pages 1–3,5, 1998. URL https://www.ime.unicamp.br/~veronica/MI402/Randi21998.pdf.

[53] Alexander I. Cowen-Rivers, Wenlong Lyu, Zhi Wang, Rasul Tutunov, Hao Jianye, Jun Wang, and Haitham Bou Ammar. HEBO: Heteroscedastic Evolutionary Bayesian Optimisation. *ArXiv*, abs/2012.03826, 12 2020. URL https://api.semanticscholar.org/CorpusID:227338279.

[54] Oded Green and David A. Bader. Faster Clustering Coefficient Using Vertex Covers. In *2013 International Conference on Social Computing*, pages 321–330, Alexandria, VA, USA, September 2013. IEEE. ISBN 978-0-7695-5137-1. doi: 10.1109/SocialCom.2013.51. URL http://ieeexplore.ieee.org/document/6693348/.

[55] Jon Herman and Will Usher. Salib: An open-source python library for sensitivity analysis. *Journal of Open Source Software*, 2(9):97, 2017. doi: 10.21105/joss.00097. URL https://doi.org/10.21105/joss.00097.

[56] Takuya Iwanaga, William Usher, and Jonathan Herman. Toward salib 2.0: Advancing the accessibility and interpretability of global sensitivity analyses. *Socio-Environmental Systems Modelling*, 4:18155, May 2022. doi: 10.18174/sesmo.18155. URL https://sesmo.org/article/view/18155.

[57] NVIDIA. Rapids cudf: Gpu dataframe library. https://github.com/rapidsai/cudf, 2024. Version 24.06 (stable).

[58] National Institute of Justice. Crimestat: Spatial statistics program for the analysis of crime incident locations, Dec 2019. URL https://nij.ojp.gov/topics/articles/crimestat-spatial-statistics-program-analysis-crime-incident-locations. Accessed: 2024-06-19.

[59] Danilo Jimenez Rezende and Fabio Viola. Taming vaes. *arXiv preprint arXiv:1810.00597*, Oct 2018. doi: 10.48550/arXiv.1810.00597. URL http://arxiv.org/abs/1810.00597. NeurIPS 2018.

[60] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models, May 2014. URL http://arxiv.org/abs/1401.4082. arXiv:1401.4082 [cs, stat].

[61] Yixin Wang, David M. Blei, and John P. Cunningham. Posterior collapse and latent variable non-identifiability. *arXiv preprint arXiv:2301.00537*, January 2023. doi: 10.48550/arXiv.2301.00537. URL https://doi.org/10.48550/arXiv.2301.00537. NeurIPS 2021.

[62] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691*, Dec 2019. doi: 10.1561/2200000056. URL http://arxiv.org/abs/1906.02691. Foundations and Trends in Machine Learning: Vol. 12 (2019): No. 4, pp 307-392.

[63] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, Dec 2022. doi: 10.48550/arXiv.1312.6114. URL http://arxiv.org/abs/1312.6114. Fixes a typo in the abstract, no other changes.

[64] M. Sarrafzadeh. Department of electrical engineering and computer science. *ACM SIGDA Newsletter*, 20(1):91, June 1990. ISSN 0163-5743. doi: 10.1145/378886.380416. URL https://dl.acm.org/doi/10.1145/378886.380416.

[65] Thomas N. Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, Nov 2016. doi: 10.48550/arXiv.1611.07308. URL http://arxiv.org/abs/1611.07308. Bayesian Deep Learning Workshop (NIPS 2016).

[66] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, May 2016. doi: 10.48550/arXiv.1511.06349. URL http://arxiv.org/abs/1511.06349. First two authors contributed equally. Work was done when all authors were at Google, Inc.

[67] Arindam Banerjee. An Analysis of Logistic Models: Exponential Family Connections and Online Performance. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 204–215. Society for Industrial and Applied Mathematics, April 2007. ISBN 978-0-89871-630-6 978-1-61197-277-1. doi: 10.1137/1.9781611972771.19. URL https://epubs.siam.org/doi/10.1137/1.9781611972771.19.

[68] Zhan Gao, Elvin Isufi, and Alejandro Ribeiro. Stochastic Graph Neural Networks. *IEEE Transactions on Signal Processing*, 69:4428–4443, 2021. ISSN 1053-587X, 1941-0476. doi: 10.1109/TSP.2021.3092336. URL http://arxiv.org/abs/2006.02684. arXiv:2006.02684 [cs, eess].

[69] Md Khaledur Rahman, Abhigya Agrawal, and Ariful Azad. MarkovGNN: Graph Neural Networks on Markov Diffusion. *arXiv preprint arXiv:2202.02470*, Apr 2022. doi: 10.48550/arXiv.2202.02470. URL http://arxiv.org/abs/2202.02470. Graph Learning at the ACM Web Conference 2022, total 11 pages.

[70] Jiaxuan You, Xiaobai Ma, Daisy Yi Ding, Mykel Kochenderfer, and Jure Leskovec. Handling missing data with graph representation learning. *arXiv preprint arXiv:2010.16418*, Oct 2020. doi: 10.48550/arXiv.2010.16418. URL http://arxiv.org/abs/2010.16418. NeurIPS 2020.