Augmenting semi-mechanistic models with deep learning to model the SARS-CoV-2 pandemic

Candidate no. 1060482

Word count: 20895

Submitted in partial completion of the

MSc in Advanced Computer Science

Trinity 2022

Abstract

Inferring key parameters of the SARS-CoV-2 pandemic and predicting its future are important tasks for epidemiologists. In this project we produce three distinct semi-mechanistic COVID models, all of which use the renewal process, with its parameters inferred using Bayesian machine learning.

Our first form allows us to infer how various COVID countermeasures interact with each other to reduce the epidemiological parameter R, something that has not been considered until now. We see that we generally get positive interactions between countermeasures - meaning interventions are more effective in tandem with other interventions - except when closing educational institutions are considered. Our second form, named Epi-ARMA, allows us to predict future COVID data using a statistical framework, in which we model noise in R as following an ARMA time series distribution. Our third form, named Epi-NN, is the namesake for this project. We augment the semi-mechanistic model with a feedforward neural network that aims to infer the value of R from raw data. This neural network allows us to then predict future COVID data. Both Epi-ARMA and Epi-NN are shown to be better than some simple baselines.

While we specifically discuss this augmentation with deep learning for a COVID model, we note that the framework we build should generalise to other semi-mechanistic models. We therefore also introduce a framework that uses deep learning to infer the values of time dependent parameters in semi-mechanistic models.

Contents

1 Introduction

Contents

All code produced in this project is saved in the git repo
https://github.com/theolewy/Epi-Predict

1.1 Motivation

The SARS-CoV-2 pandemic impacted lives across the world, and its consequences have been far-reaching. From being the direct cause of excess deaths to indirectly causing economic damage due to the closure of businesses, its effects have damaged society across the board. Due to this the role of the epidemiologist has become more important now than it has ever been before. Modelling the pandemic allows for crucial inferences to be made, which in turn can be used to help predict potential futures.

Knowing the effect of potential interventions allows governments to choose the best countermeasures to combat the spread of COVID. This means that strong

inference and prediction is key in helping governments make good policies. Further, being able to predict future case numbers allows one to prepare for the weeks ahead, hopefully preventing deaths. The tasks ahead of us are therefore key in helping make policy decisions with regards to pandemics. We hope that the work done in this project may help with the analysis of future epidemics as soon as they begin.

1.2 Objectives

In this project we use a semi-mechanistic Bayesian COVID model based on the renewal process. By semi-mechanistic we mean that there is an underlying mechanistic model that we are using (here this is the renewal process), but that we fit its parameters using Bayesian machine learning. In this dissertation we use this to meet three main objectives.

1. We investigate whether COVID interventions, such as closing small businesses or schools, act independently from each other. Previous models assume that the countermeasures do not interact with each other, forcing their effects to be independent. We investigate to what degree this is true and explain potential causes for these interactions.

2. We aim to predict future trajectories of the pandemic, and we build two methods to do this, named Epi-ARMA and Epi-NN.

3. We extend semi-mechanistic Bayesian models with deep learning, and suggest a general framework with which this can be done.

We now explain the structure of this dissertation so that we can see how we fulfil these objectives.

1.3 Dissertation Structure

Our first job in this dissertation is to provide the necessary background to explain our models. We start in chapter 2 with the necessary technical background that will allow us to understand how the semi-mechanistic models work. This background is in three parts. Firstly we look into epidemiology so that we can understand the mechanisms underpinning the dynamics of a pandemic, and in particular we look at a renewal process model. Secondly we look into Bayesian machine learning, which provides a framework with which we can find reasonable parameters for a model. In

our case we find reasonable parameters for the renewal process. Lastly we explain how a feedforward neural network works, as this is what we will eventually use in our model to infer the value of the COVID reproductive number, R.

Then in chapter 3, we explain how the model that served as our base (from [13]) actually works. We do this so that we can then understand our adaptations to this model later.

In chapter 4 we begin our first section of novel work, and we tick off our first objective for this project. We perform inference, rather than prediction, and we investigate how various countermeasures interact with each other to produce different reductions in R.

We then turn our attention towards prediction. Before we introduce our own work, we look at chapter 5 in which we explain current best practices for COVID prediction. For example, we explain how we can measure the performance of a predictor in a rigorous way, as well as introducing two methods for prediction that are already around. These are EpiNow2 and Prophet. This will allow us to measure the ability of our models against these benchmarks as we introduce them.

In chapter 6 we introduce our first prediction method, which we have named Epi-ARMA. This assumes that noise in R follows an ARMA time series, which then allows us to predict possible trajectories for R in the future. We compare this model to the benchmarks introduced in the previous chapter.

After this we delve into the main chapter of the project, chapter 7. This chapter is the namesake of the project, and we introduce Epi-NN here, where we introduce neural networks into our model. Rather than modelling the noise in R as following an ARMA time series, we instead model R as a learned function of the reported case and death data using a neural network. This allows us to parameterise R using a neural network. Epi-NN therefore uses current data to predict future values of R. At the end of this chapter we explain how our augmentation of the semi-mechanistic model with a neural net can be generalised to other models, allowing us to fulfil our final objective for this project.

2

Background

Contents

In this chapter we will provide the relevant background that is required to understand how our COVID model works. We will begin by learning about epidemiology and the renewal process, which provides a way to model epidemics.

We then learn how Bayesian machine learning works. This requires a probabilistic program which takes a model, samples its parameters, and then uses these to obtain some outputs. The program is then conditioned on observed data to allow us to find a posterior distribution - a distribution of the model parameters given the observed data. We would like to sample from this to obtain reasonable

parameters for our model. We then discuss sampling methods, so that we can actually obtain these samples.

Lastly we look into feedforward neural networks, which will allow us to learn general functions between a set of inputs and outputs. We will eventually use this to obtain epidemiological parameters from epidemic data.

2.1 Epidemiology and the Renewal Process

There are a number of ways to consider the dynamics of an epidemic. The most commonly used is perhaps the Susceptible-Exposed-Infected-Removed model (SEIR), which uses mulitvariate differential equations to consider the trajectory of an epidemic [7], [10]. We use a less commonly used model in this project, known as the Renewal Process model, which uses a number of delay distributions to model how infections pass through a population [8]. Usefully these two models are equivalent, as shown in [5], and so are equally expressive. These models are used to track how an epidemic progresses, and in particular allow one to predict the number of infections that could be expected in an epidemic.

As one might imagine, epidemiology aims to model how disease spreads. It tries to link the number of infected people to the number of cases that are reported, as well as the number of people who die. We will now explain how the renewal process works in light of this. The model considers three time series. We use an explanation from [8].

The first time series is the number of infected people, and we denote the number of infections at time t by i_t. These infections are not observable in and of themselves, but will later become cases and deaths which can be observed. The value of i_t is determined by

$$i_t = R_t \sum_{s<t} i_s g_{t-s} \tag{2.1}$$

where R_t is the *reproductive number* of the disease at time t, and g_t is the *generation interval* distribution (satisfying $g_t > 0$ and $\sum_t g_t = 1$). The value of R_t represents the average number of infections seeded by a previous infection. R_t values greater than 1 cause exponential increases in infections, while values less than 1 cause exponential decreases. The distribution g_t represents the proportion of the infections that are seeded by the infections t days prior to it.

The second and third time series are cases and deaths, and at time t these are denoted by c_t and d_t. They are dealt with in similar ways, and their values are determined by

$$c_t = \alpha_t \sum_{s<t} i_s \pi_{t-s} \tag{2.2}$$

$$d_t = \beta_t \sum_{s<t} i_s \sigma_{t-s} \tag{2.3}$$

We have α_t and β_t as the time dependent *infection ascertainment rate* and the *infection fatality rate* respectively. These represent the rate at which infections become cases and deaths respectively. We also have π_t and σ_t as the appropriate delay distributions, similarly to g_t above.

Now we consider the task of predicting the trajectory of an epidemic until time $t = T$. Suppose the parameters R_t, α_t and β_t are set for all time $0 \leq t \leq T$, and the distributions g_t, π_t and σ_t are fixed. Then, if we know all i_t before time $t = 0$, then equations (2.1) - (2.3) fully determine the system until time $t = T$. We can consider these initial conditions $(i_t)_{t<0}$ to be extra parameters for the renewal process. Hence, to find the epidemic trajectory until time $t = T$ the renewal process has parameters of $(R_t,\ \alpha_t,\ \beta_t)_{0\leq t\leq T}$, $(g_t, \pi_t, \sigma_t)_{0\leq t}$ and $(i_t)_{t<0}$, which fully determine its outputs.

To deal with our infinite collection of parameters we can truncate distributions g_t, π_t and σ_t. This is reasonable, as an infection from 2 months ago has a negligible chance of creating more infections today, and also a negligible chance of becoming a case or death today. If we truncate these distributions after T_0 days, then this also means we only need to know infection seeds $(i_t)_{-T_0\leq t<0}$ for our system to be fully determined.

Importantly we note that we now have a finite collection of parameters, which are $(R_t,\ \alpha_t,\ \beta_t)_{0\leq t\leq T}$, $(g_t, \pi_t, \sigma_t)_{0\leq t\leq T_0}$ and $(i_t)_{-T_0\leq t<0}$. When these have been set we find that the the renewal model deterministically finds the future of the epidemic up until time $t = T$.

The key question now however is how do we find these parameters. There will be uncertainty in how they are set, and so we would like our methodology to reflect this. Hence we turn to Bayesian Machine Learning, a tool which uses data to infer the parameters of a model.

2.2 Bayesian Machine Learning

Standard machine learning aims to learn the single best estimate of a model's parameters, which means it has no built-in sense of uncertainty. Bayesian machine learning however provides a framework that can represent uncertainties. Its key strength is its probabilistic nature.

The core of any Bayesian machine learning algorithm is a probabilistic model. This probabilistic model samples the model parameters (θ) and uses these to (deterministically) produce some outputs ($\tilde{x}$). Generally, observed data (x) is used to fit the model. This means that the sampled values of θ should allow the model's output $\tilde{x}$ to reproduce the observations x.

For example a simple model might try to find the expected income of a shop on a given day, $\tilde{x}$, by sampling parameter θ, where $\theta \sim \mathcal{N}(5, 1)$. It then deterministically sets $\tilde{x} = 1000\theta$. It might also be known that in the last three days, the shop had an actual income, x, of $7000, 4000$ and 8000.

Alternatively, as we will discuss later in chapter 3, $\tilde{x}$ could be the expected number of COVID cases and deaths, with θ being the parameters of the renewal process (see section 2.1) that is used to produce this estimate. The observed data x in this case would be the actual COVID case and death data.

There are three main tasks associated with Bayesian Machine Learning.

1. Building a parameterised probabilistic model.
2. Conditioning it on observed data to obtain a probability distribution over its parameters (the *posterior* distribution).
3. Sampling from the posterior to get possible parameterisations of the model.

We discuss these tasks one by one.

2.2.1 Bayesian Probabilistic Models

Building a probabilistic model is surprisingly simple. We can begin with a parameterised deterministic model, like the renewal equation. We define the complete vector of parameters here to be θ. To turn this deterministic model into a probabilistic one we must do two things.

Firstly we place a *prior* on each of its parameters. This represents our beliefs about θ when no data is available. Mathematically this is a distribution $p(\theta)$, and in our shop example above, we used $\theta \sim \mathcal{N}(5, 1)$. The prior is preset and user-defined for any model. There is no single method to choosing a prior, and one could generally set their prior to be from many different distributions. It is good practice to verify that the choice of prior has little effect on the eventual model output after the model has been conditioned on data by performing some robustness analysis. This involves checking that changing the prior distributions does not change the posterior dramatically.

Secondly we must define a *likelihood* probability when data is available. This gives us a probability that the observed data $\mathcal{D}$ would occur, given the model parameters. Mathematically this is a distribution $p(\mathcal{D}|\theta)$. Typically, the likelihood is set so that it is maximised when the parameters θ make the model outputs $\tilde{x}$ as close to the observed data x as possible. We can define the likelihood in terms of an observational model, where we believe the observed data x comes from a defined distribution dependent on $\tilde{x}$. For example, in the shop example we may define the observational model to be $x \sim \mathcal{N}(\tilde{x}, 1)$, or equivalently in terms of the model parameters, $x \sim \mathcal{N}(1000\theta, 1)$.

To compute the likelihood, generally the observed data is assumed to be independent. For example, in our shop example we would have

$$p(\mathcal{D}|\theta) = \prod_{x_i \in \mathcal{D}} p(x_i|\theta) = \prod_{x_i \in \mathcal{D}} \mathcal{N}(x_i; 1000\theta, 1)$$

where $\mathcal{N}(x; \mu, \sigma)$ is the probability density function of the normal distribution with mean μ and standard deviation σ.

To recap, we need to do two things to turn a deterministic model into a probabilistic program. Firstly we attach a *prior* to each of its parameters, and secondly we produce an observational model that defines the *likelihood*. Next we will appeal to Bayes' Law to allow us to use these two distributions to find one more distribution, the *posterior*.

2.2.2 Conditioning on Data: Bayes' Law

Bayes' Rule gives us a way to mathematically incorporate data into our prediction of what θ should be. We have already described the prior $p(\theta)$ and the likelihood $p(\mathcal{D}|\theta)$, but we must now explain one more distribution. The posterior $p(\theta|\mathcal{D})$ represents the distribution of θ given the observed data, and is the most important distribution in Bayesian machine learning.

Bayes' Law gives us a way to calculate it:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \propto p(\mathcal{D}|\theta)p(\theta) \tag{2.4}$$

It turns out that the normalising factor $p(\mathcal{D})$ is not important, as constants of proportionality eventually get ignored when we actually sample from this distribution. Using Bayes' Law has therefore provided us with a way to get access

to an unnormalised posterior distribution of a model's parameters. We would now like to take samples from this posterior, as the parameters that we will sample will parameterise our original model well.

2.2.3 Sampling Methods

Bayesian machine learning allows us compute the unnormalised posterior distribution for our system parameters by using our observed data. We must be able to sample from this posterior if we are to make use of it, and so we will now look into the key methods for sampling. Usefully, there are a number of methods in which normalisation is unnecessary.

MCMC Methods

One of the more widely used class of methods are Markov Chain Monte Carlo (MCMC) algorithms. The term Monte Carlo simply refers to the fact that we are taking repeated random samples, however explaining a Markov Chain is more complicated. Our descriptions come from [1].

A Markov Chain is a series of random variables X_t, each of which satisfies the Markov property. This property states that the probability distribution of X_{t+1} is fully determined by the state of X_t. This means that for a Markov Chain, we have

$$p(X_{t+1}|X_1, ..., X_t) = p(X_{t+1}|X_t) \qquad \forall t$$

In essence, this means that the future of the chain only depends on the present, and not the past. We also will only consider Markov Chains in which $p(X_{t+1}|X_t)$ is independent of t, and hence we can drop any reference to t in the so called transition probabilities, and so we write $p(x|x')$ to denote the (time-independent) probability that the chain moves from a state of x' to a state of x in one step. This time independence means the Markov Chain is said to be time-homogeneous. Time-homogeneous Markov Chains have a rich mathematical structure, and lead to a number of important results. We will discuss here one of the key ones for this project.

Firstly we define an invariant distribution to be a probability distribution π that satisfies

$$\pi(x) = \sum_{x'} \pi(x')p(x|x')$$

This is invariant in the sense that if X_t has probability distribution π, then

X_{t+1} will also have distribution π.

Our key result is that if a Markov Chain has a unique invariant distribution π then

$$\lim_{t\to\infty} p(X_t) = \pi(X_t)$$

We will not prove this here, but this is a hugely significant result. If one can construct a Markov Chain in which the unique invariant distribution is the posterior distribution of a model, then the Markov Chain will eventually follow the posterior distribution. This means that one can sample from a complicated posterior distribution by sampling from a (much simpler) Markov Chain.

There are a number of methods to construct such a Markov Chain with the required invariant distribution, however we will only discuss the one required in this project, called Hamiltonian Monte Carlo.

Hamiltonian Monte Carlo

Hamiltonian Monte Carlo (HMC) simulates physics to produce a Markov Chain. The following analogy comes from [2], and we use it to briefly explain the relation of HMC to the physical world. Imagine a frictionless puck with horizontal position $\mathbf{x}$ moving along a surface with height $U(\mathbf{x})$ with horizontal velocity $\mathbf{v}$. The dynamics of such a system are determined by

$$\frac{d\mathbf{x}}{dt} = \mathbf{v}$$

$$\frac{d\mathbf{v}}{dt} = -g\nabla U(\mathbf{x})$$

where g is the gravitational field strength. In particular this means that the total energy of the system, the Hamiltonian H, is conserved:

$$\begin{aligned}
\frac{d}{dt}H &= \frac{d}{dt}(\text{Kinetic Energy} + \text{Gravitational Potential Energy}) \\
&= \frac{d}{dt}(\frac{1}{2}m|\mathbf{v}|^2 + mgU(\mathbf{x})) \\
&= m\mathbf{v}\cdot\frac{d\mathbf{v}}{dt} + mg\frac{d\mathbf{x}}{dt}\cdot\nabla U(\mathbf{x}) \\
&= -mg\mathbf{v}\cdot\nabla U(\mathbf{x}) + mg\mathbf{v}\cdot\nabla U(\mathbf{x}) \\
&= 0
\end{aligned}$$

where m is the mass of the puck. We now set $m = g = 1$ for simplicity. This

system can be used to form a Markov Chain in the $\mathbf{x}$ variable. Starting with an initial value of $\mathbf{x}$, with $X_0 = \mathbf{x}$, we sample $\mathbf{v}$ from a fixed distribution (typically $\mathbf{v} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$). We then simulate the physical system for a fixed amount of time, allowing the puck to move across the surface. At the end of this time step, the puck has moved to $\mathbf{x'}$, which is the new value of the Markov Chain, and so $X_1 = \mathbf{x'}$. We continue the Markov Chain by resampling $\mathbf{v}$, and continuing the physical simulation. We note that for this Markov Chain we do not care about the values of $\mathbf{v}$, only the values of $\mathbf{x}$. Also in practice we note that this system is discretised, but we will not worry about these details here.

The key property of this Markov Chain is that its invariant distribution $\pi(\mathbf{x})$ is proportional to $\exp(-U(\mathbf{x}))$, which we will not show here but is related to time invariance of the Hamiltonian shown above [2].

We now demonstrate how this is used to sample from a given posterior distribution $p(\theta|\mathcal{D})$, where we recall θ are the model parameters and $\mathcal{D}$ is the observed data. We firstly rewrite our Hamiltonian variable $\mathbf{x}$ as θ. We then consider the Hamiltonian system above with $U(\theta) = -\log p(\theta|\mathcal{D})$. This system has invariant distribution $\exp(-U(\theta)) = p(\theta|\mathcal{D})$. As explained above in section 2.2.3, this means that the HMC system produces a Markov Chain that tends in distribution to the given posterior distribution $p(\theta|\mathcal{D})$. In particular this means that we now have a process that lets us draw samples from the distribution $p(\theta|\mathcal{D})$.

However, we may need to perform a (potentially large) number of 'warmup steps' for the MCMC method so that the Markov Chain has reached this posterior. This is because we only have a guarantee that our Markov Chain tends to the posterior distribution eventually, and so we must run the Markov Chain for some time before we can expect it to be at the posterior. When we talk about *convergence* of a Markov Chain, this is what we are referring to - we need the chain to have converged to the posterior distribution before we start sampling from it.

As simulating Hamiltonian Dynamics is relatively easy, we now have an MCMC method that allows us to draw samples from a posterior distribution.

The No-U-Turn Sampler

The No-U-Turn Sampler (NUTS) is an adaptation of HMC that is widely used as it eliminates the need for the user to set arbitrary hyperparameters. In HMC the length of time in which the physical dynamics are simulated is set by a user. If this length of time is too small, then the Markov Chain has a slow mixing time, and it takes a long time to fully explore a posterior. If this length of time is too

large, then we are wasting computational resources for no added benefit.

Simply put, NUTS sets the length of time at which it simulates the physical system to be the time it takes for the puck to begin to curve back towards its origin. We will not trouble ourselves with the details, but we note that NUTS provides a great boost in efficiency when compared to the standard implementation of HMC [3]. We use NUTS as our sampler in this project.

2.2.4 Summary

Now that we have discussed a number of the tools at our disposal, we will discuss how they fit together. We begin by producing a probabilistic model for our system based on the deterministic renewal process discussed in section 2.1. We will set the priors and the observational model later in section 3. When conditioned on observed data, this probabilistic model allows us to produce a posterior distribution using Bayes' Law (2.2.2). This posterior is then sampled from using an MCMC method, which in our case is the No-U-Turn Sampler (which is based on Hamiltonian Monte Carlo).

In this way Bayesian machine learning provides a pipeline that takes a model and some data and turns it into a series of samples from the posterior of the model. Each sample gives us a different way to parameterise our model, and so we are able to identify uncertainty in the parameters. This uncertainty in the model's parameters can then be used to consider the uncertainty in its inferences and/or predictions. This is the power of Bayesian machine learning, as it gives us a distribution over a model's inferences and predictions.

2.3 Feedforward Neural Networks

The next piece of background that we will discuss here is the feedforward neural network. This powerful component of machine learning allows complex patterns to be identified in data. The aim is that, once trained, the neural net should be able to convert a vector input into some useful output. For example, it may be able to convert data about the characteristics of an iris (a genus of flower), into a classification of the iris' species [15]. Or it may be able to convert some data about a house into a predicted price for what the house is worth [16]. In essence a neural net is just a function that takes some inputs and gives some outputs, but its power comes from its ability to 'learn' exactly what the function is - no user has to predefine the function.

As we will see shortly, a neural net contains a large number of parameters that

control this function. It turns out that provided these parameters are set correctly, any function can be approximated by a neural net. We will explain how these parameters are learned later, but first we begin by defining an 'artificial neuron', the unit that is repeated to form a neural net.

2.3.1 Neurons

A neuron is a function that takes n inputs x_1, x_2, ..., x_n (or equivalently an n dimensional vector $\mathbf{x}$) and converts them into a single output. It has a bias $b \in \mathbb{R}$, weights $\mathbf{w} \in \mathbb{R}^n$, and activation function f associated with it. Typically f is fixed when the neuron is made, while b and $\mathbf{w}$ are parameters that will eventually be learned. The neuron will output a value of

$$a = f(\mathbf{w} \cdot \mathbf{x} + b)$$

where $\mathbf{x}$ and $\mathbf{w}$ are the inputs and weights written as column vectors. We depict this pictorially in figure 2.1.

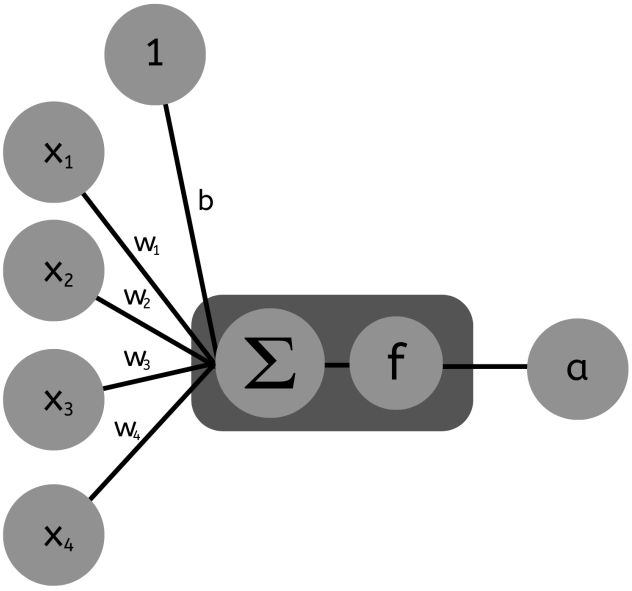

Figure 2.1: The Neuron Architecture

There are a number of choices for the activation function f, the most popular being:

- The sigmoid function $\sigma(z) := \frac{1}{1+\exp(-z)}$
- The hyperbolic tanh function $\tanh(z) := \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$

- The Rectified Linear Unit (ReLU) $\text{ReLU}(z) \coloneqq \max(0, z)$

While we have not yet shown how these neurons produce a complete neural net, we will note that in the context of a neural net each of these activation functions has their own drawbacks and successes.

2.3.2 Neural Network Assembly

A completed neural net consists of multiple layers of neurons. Each layer has an input $\mathbf{x}$ and an output $\mathbf{a}$, and the idea is that the output of each layer is the input of the next. We denote the lth layer's input and output as $\mathbf{x}^l$ and $\mathbf{a}^l$ respectively, and we will say that the neural net has L layers in total. We begin by defining the neural nets input as $\mathbf{x}^1$, and for ease of notation we set $\mathbf{a}^1 = \mathbf{x}^1$.

Each layer in the network consist of a number of neurons, each of which takes the outputs of the previous layer as its inputs. Hence for the lth layer we have $\mathbf{x}^l = \mathbf{a}^{l-1}$. Each neuron has its own weights and bias, and typically all neurons in a layer will use the same activation function. The ith neuron in the lth layer has weights $\mathbf{w}_i^l$, bias b_i^l and activation function f^l. The output of the ith neuron in this layer is therefore

$$a_i^l = f^l(\mathbf{w}_i^l \cdot \mathbf{x}^l + b_i^l)$$

Then the output of this layer is $\mathbf{a}^l = (a_0^l, a_1^l, ...)^T$

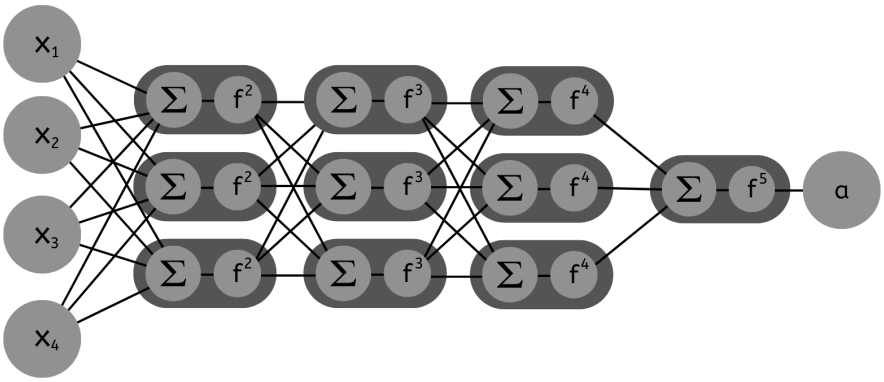

Figure 2.2: The Neural Network Architecture

Now that we know how the output of each layer is formed, we note that this is enough to generate the output of the complete network. Starting from the networks input $\mathbf{x}$, we compute the output of the first layer. We then take this output and feed it in as an input to the second layer. We repeat this until we have fed the

data through the whole network, and we finish with our network output of $\mathbf{a}^L$. This vector is the important output of the network.

Our neural network therefore converts its input $\mathbf{x}^1$ into an output $\mathbf{a}^L$. It is incredible flexible when it has a large number of parameters, and the architecture described is able to approximate many functions. The next question is how are the parameters of the network set so that the neural network is approximating the required function.

2.3.3 Learning

In standard machine learning the parameters are trained via a 'loss function'. This function provides a sense of how close the neural network is to approximating a function. One would take a number of examples where both the input and the output are known, and then predicted outputs would be generated by the neural network. The loss function then gives a metric for how close the predicted outputs were to the actual outputs. While this is a hugely important method for neural networks to learn good parameterisations, we will not worry about the details of this here as it is not what we use in this project. Instead we are able to take advantage of the fact that we are using Bayesian machine learning.

In section 2.2 we saw how to deal with parameters within a model that we do not know the value of. We first attach a prior to them, and then we make use of any data by conditioning the model by observing the data. This gives us a likelihood, and so we are able to find the models posterior by using Bayes' Theorem. We can do this with the parameters of our neural network, and so these parameters are simply extra variables in the posterior. We then saw in section 2.2.3 how we sample from posterior distributions. In this way we are able to treat the parameters of the neural network just like any parameter in our Bayesian probabilistic program, and so we need not worry about providing any new methods for the neural network to learn its parameters, but we use the methods already outlined.

3

Previous COVID Models

Contents

Now that the relevant background has been covered we will turn our attention to modelling the COVID pandemic. We will be using a semi-mechanistic Bayesian framework. 'Mechanistic' refers to the fact that the model makes use of known dynamics of pandemics - in this case we use the renewal model from (2.1). This is only 'semi-mechanistic' however, as we use techniques other than this renewal process mechanism. Specifically, we are using Bayesian machine learning to infer the values of the parameters of the renewal model.

This framework of semi-mechanistic Bayesian models that use the renewal

process is currently popular [8], [9], [13]. The work in this project is mainly built on Sharma et al's model in their paper [13], and in particular the code base used was that from the git repo from this paper [12]. The aim of their model that we use as our starting point is to investigate how various COVID countermeasures impact the epidemiological reproductive number R. In particular it aims to find what percentage reduction in R is obtained when different countermeasures are implemented. These countermeasures range from the introduction of curfews to the closing of universities. They look at COVID data in 114 regions from August 2020 to the end of January 2021 which makes up the 'second wave' of the pandemic.

We use this model instead of other models as it performs a large amount of pooling - in particular it uses data from all regions to help inform its parameters in any one region. This is useful as it means we do not treat each region as completely independent from the rest, something that a number of other models do, including one of the most widely used epidemiology models, EpiNow2 [6]. It also comes with a large amount of data that we are able to make use of. In this project we adapt this model into three distinct forms to perform other kinds of inferences, as well as predicting future rates of COVID. Before we get to our models however, we begin with Sharma et al's original model.

3.1 The Equations

The model presented in [13] turns the renewal process into a relatively simple semi-mechanistic Bayesian model. To do this they begin by taking the renewal process and considering all of its parameters. Recall from (2.1) that the equations are

$$i_t^{(r)} = R_t^{(r)} \sum_{s<t} i_s^{(r)} g_{t-s} \tag{3.1}$$

$$c_t^{(r)} = \alpha_t^{(r)} \sum_{s<t} i_s^{(r)} \pi_{t-s} \tag{3.2}$$

$$d_t^{(r)} = \beta_t^{(r)} \sum_{s<t} i_s^{(r)} \sigma_{t-s} \tag{3.3}$$

where we now use a superscript of (r) to denote a region specific value. They truncate the distributions g_t, π_t and σ_t to be of length T, and note that the parameters of this model are:

Parameter	Times t	Meaning
$R_t^{(r)}$	$0 \leq t$	The Reproductive Rate
$\alpha_t^{(r)}$	$0 \leq t$	Infection Ascertainment Rate
$\beta_t^{(r)}$	$0 \leq t$	Infection Fatality Rate
g_t	$0 \leq t < T$	Infection Delay Distribution
π_t	$0 \leq t < T$	Infection to Case Distribution
σ_t	$0 \leq t < T$	Infection to Death Distribution
$i_t^{(r)}$	$-T \leq t < 0$	Seeded Infections

The aim of the model is to infer the values of these parameters. In particular, the model attempts to find the dependence of R_t on countermeasures that are in place at time t.

The key is that whenever we have an unknown parameter θ_t, we can do one of two things. Firstly we can place a prior on it directly. This would result in us having each θ_t at each time t be independent of each other. Alternatively, and more powerfully, we can use a Bayesian hierarchy. This means that we define a way to recover θ_t using other parameters, each of which have there own prior. This second method is powerful, as it gives us a way to keep the θ_t dependent on each other. It also means we may not need to sample one parameter for each time t, but instead we can produce all of θ_t from a smaller set of inferred parameters.

We will now go through each of the above parameters and explain how the models infers their value, either by using outside knowledge, directly placing a prior on them, or by using a Bayesian hierarchy.

3.2 The Parameters

3.2.1 The Distributions g_t, π_t and σ_t

These distributions are not inferred by the model, but rather taken from previously done work. The g_t distribution was found using a meta-analysis paper [11]. The distributions π_t and σ_t were fit to (truncated) gamma distributions using already obtained data. The shape and scale (the two parameters of a gamma distribution) were set by finding the best fit for patient data containing dates of infections, reported cases and reported deaths. As these were done externally to the model we will be discussing, these three distributions are (effectively) user-inputted known quantities for the model.

We remind ourselves that if one were to pick user-defined values for all of the parameters, we would get a mechanistic model for the dynamics of COVID. Instead however, we will infer the value of the rest of the parameters (R_t, α_t, β_t and i_t) using Bayesian machine learning.

3.2.2 The Reproductive Number $R_t^{(r)}$

As the point of the model is to infer how R_t changes when various countermeasures are in play, we would like R_t to be a function of these countermeasure. Hence we will use a Bayesian hierarchy here. We define $\mathbf{x}_t^{(r)}$ to be the one hot encoding of which countermeasures are activated at time t in region r, which is known data and need not be inferred. This notation means, for example, that if the ith countermeasure is that 'Secondary Schools Close', then $x_{i,t}^{(r)} = 1$ means that at time t, secondary schools were closed in region r, and $x_{i,t}^{(r)} = 0$ otherwise.

The model decomposes $R_t^{(r)}$ into three parts.

$$R_t^{(r)} = R_{\text{basic}}^{(r)} R_{cms,t}^{(r)} R_{noise,t}^{(r)}$$

The first part $R_{\text{basic}}^{(r)}$ is independent of time, and represents the basic value of R in a region. The model sets a prior of $R_{\text{basic}}^{(r)} \sim \text{Truncated Normal}(1.35, 0.3^2)$ where the truncation stops it from being less than 0.1.

The second part $R_{cms,t}^{(r)}$ represents the fractional decrease in $R_t^{(r)}$due to the countermeasures, and it is only a function of the COVID countermeasures active in region r at time t. The model sets

$$R_{cms,t}^{(r)} = \exp(-\gamma \cdot \mathbf{x}_t^{(r)})$$

We can therefore interpret the ith component of γ as determining the multiplicative reduction of $R_t^{(r)}$due to countermeasure i being in place. In particular, countermeasure i being active reduces $R_t^{(r)}$by $\exp(-\gamma_i)$. This parameter is shared across all regions, making it easily interpretable. They then set a prior on γ_i of an Asymmetric Laplace distribution with scale parameter 30, asymmetry parameter 0.5, and location parameter 0. This is all we need to define $R_{cms,t}^{(r)}$ in our model. We note that the form of $R_{cms,t}^{(r)}$ forces us to assume that each countermeasure is completely independent of the other countermeasures - there is no interaction between them.

The third and final part of $R_t^{(r)}$ is $R_{noise,t}^{(r)}$. The rationale behind this term is as follows. As $R_{\text{basic}}^{(r)}$ is independent of time, and $R_{cms,t}^{(r)}$ is only dependent on time via the countermeasures $\mathbf{x}$, setting $R_t^{(r)} = R_{\text{basic}}^{(r)} R_{cms,t}^{(r)}$ would give us that $R_t^{(r)}$ is

constant in region r when a given set of countermeasures are in place. This is not likely to be true, as a population is likely to change its behaviour over time even when there is no change to the active countermeasures. For example, we would expect there to be less compliance with lockdown restrictions the longer these restrictions are in place. To account for this $R^{(r)}_{noise,t}$ is introduced. Sharma et al set this noise term to follow a random walk which can change once a week as follows.

$$R^{(r)}_{noise,t} = \exp(z^{(r)}_t)$$

$$\text{where } z^{(r)}_t = \begin{cases} 0 & t \leq 13 \\ z^{(r)}_{t-1} + \epsilon_{\lfloor (t-14)/7 \rfloor} & \text{if } t \bmod 7 = 0 \\ z^{(r)}_{t-1} & \text{otherwise} \end{cases}$$

From the perspective of a Bayesian probabilistic program, we treat the ϵ_i as parameters to deterministically find $R^{(r)}_{noise,t}$, and we use a prior of $\epsilon_i \sim \mathcal{N}(0, \sigma_R^2)$. Note smaller values of σ_R limit how much the random walk can jump in one step. While a user could set σ_R to be a user-defined value, the model instead sets a prior on it, so that the model can find its own value for it. The model sets this prior as $\sigma_R \sim$ Half Normal(0.15), and so we are using a Bayesian hierarchy here. In particular this prior causes the model to make σ_R as small as possible while also fitting the data accurately.

We note that this noise term is the main improvement when compared to previous models, such as the one in [8]. This model lacks the noise term, and so models $R^{(r)}_t = R^{(r)}_{\text{basic}} R^{(r)}_{cms,t}$, albeit with a slightly different form for $R^{(r)}_{cms,t}$. In particular, this extra term gives the model the benefit of allowing $R^{(r)}_t$ to change on a weekly basis. Further, we no longer need to input a countermeasure for its effect to be seen in the model. Sharma et al mention in [13] that any countermeasures that are not directly considered in their $\mathbf{x}$ vector are still able to be dealt with by their noise term. However, any countermeasure x_i that is included can still have its reduction impact looked at, simply by considering γ_i.

3.2.3 The Rates $\alpha^{(r)}_t$ and $\beta^{(r)}_t$

While the model in [8] mentions that $\alpha^{(r)}_t$ and $\beta^{(r)}_t$ can be modelled in a similar way to how they build $R^{(r)}_t$, the model in [13] uses a simpler method. This simply involves taking $\alpha^{(r)}_t = \alpha^{(r)}$ and $\beta^{(r)}_t = \beta^{(r)}$ as constants over time for each region. This assumption has a useful consequence. By simply changing the scale of our infections $i^{(r)}_t$ we can absorb in the constant $\alpha^{(r)}$ into it. This is equivalent to

just setting $\alpha^{(r)} = 1$ in equations (3.1) - (3.3).

Note this consequence means that we are effectively assuming the infection ascertainment rate to be 1. Hence the infection fatality rate $\beta^{(r)}$ is identical to a case fatality rate ($\text{cfr}^{(r)}$), as

$$\text{cfr}^{(r)} = \frac{\beta^{(r)}}{\alpha^{(r)}} = \beta^{(r)}$$

The model therefore sets $\alpha_t^{(r)} = 1$, and then places a prior on the variable $\beta^{(r)} = \text{cfr}^{(r)}$ of $\beta^{(r)} \sim U(0.001, 1)$.

3.2.4 The Seedings $i_t^{(r)}$

The seedings are modelled in a simple way. A prior is placed on them of $i_t^{(r)} \sim$ Log Normal($\mu = 0, \sigma = 3$) for $-7 \leq t \leq -1$. We also assume $i_t^{(r)} = 0$ for $t < -7$.

3.3 Observation Model

We have now seen how all of the parameters of the renewal equations are modelled, and in particular we have seen how the priors are set. Recalling how Bayesian machine learning works (see 2.2), we note the final remaining piece of our probabilistic program is a definition of the likelihood of the model parameters. This means building in an observational model that takes all of a models parameters θ and the observed data $\mathcal{D}$, and finds a probability $p(\mathcal{D}|\theta)$. This is enough to produce a posterior $p(\theta|\mathcal{D})$ using Bayes' Law, which can then be sampled from using NUTS.

To compute $p(\mathcal{D}|\theta)$ we take all of the parameters θ and feed them into the renewal model. This produces two time series for the expected cases $\tilde{c}_t^{(r)}$ and expected deaths $\tilde{d}_t^{(r)}$. We then compare this to the actual case data $c_t^{(r)}$ and actual death data $d_t^{(r)}$.

The model assumes that $c_t^{(r)}$ is sampled from a negative binomial distribution with mean $\tilde{c}_t^{(r)}$ and dispersal $\psi^{\text{c}}_{\text{country}(r)}$, where $\psi^{\text{c}}_{\text{country}(r)}$is a variable that varies over countries, and not individual regions, and has a prior set of $\psi^{\text{c}}_{\text{country}(r)} \sim$ Half Normal(5). The same is done for $d_t^{(r)}$ with the same priors on parameters $\psi^{\text{d}}_{\text{country}(r)}$.

This observation model allows us to compute $p(\mathcal{D} = (c_t^{(r)}, d_t^{(r)})|\theta)$. This is enough to fully define the model.

3.4 Data

The model we have just described was used primarily to find how the value of $R_t^{(r)}$ changed as various COVID countermeasures were put into place during the second COVID wave from August 2020 to the end of January 2021. Due to this the data both needed to capture the rises and falls of the pandemic, but also needed to track which countermeasures were in place in each region at all times. The data kept track of is found in table 3.1.

There are a large number of countermeasures that are kept track of here. There are two main problems associated with this.

1. Some of the countermeasures are not binary (eg Public Outdoor Gathering Limit), and hence producing $\mathbf{x}_t^{(r)}$ (the one hot encoding of the countermeasures) is not straightforward. Recall this was required for the model described to produce $R_{cms,t}^{(r)}$ in an easily interpretable way.

2. While ideally we would like to learn the relative importance of each of these countermeasures, there is not often enough data that allows us to disentangle the effects of one countermeasure from another. For example, if primary and secondary schools were only ever closed at the same time then it would not be possible for the model to work out which closure produced what reduction in R.

The end result of this is that some preprocessing is required. The aim was to process the countermeasures into a smaller number of binary fields. We also required that for each pair of these fields that there is some data in which one field is active and the other one is not. This ensures we are able to disentangle their effects. Sharma et al convert this data into the binary countermeasures listed in table 3.2.

They preprocess their data in such a way that all of the countermeasures are now binary, and so we can now make use of the data using their model. They use 114 regions across 7 different countries over the course of 175 days starting from 01/08/2020 and ending on 22/01/2021.

We have explained all of the key components of our base model now. All of the parameters of the renewal model have had priors placed on them, and an observational model defines how data is used. We have also seen what data is available to us. We have defined the model completely here.

Column Heading	Example	Notes
Region and Time Data		
Date	27/12/2020	
Region	Lincolnshire	
Country	England	The country that the region is in
Observational Data		
New Cases	321	New Cases reported on this day
New Deaths	17	New Deaths reported on this day
Countermeasures		
Public Outdoor Gathering Limit	6	The maximum number of people that can go to a public outdoor gathering
Public Indoor Gathering Limit	1	
Private Outdoor Gathering Limit	1	
Private Indoor Gathering Limit	1	
Public Outdoor Household Limit	6	The maximum number of different households that people at a public outdoor gathering can come from
Public Indoor Household Limit	1	
Private Outdoor Household Limit	1	
Private Indoor Household Limit	1	
Mandatory Mask wearing	2	How stringent the rules on masks are, with 0 being least stringent and 5 being most.
Gastronomy Closed	1	One hot encoded
Leisure Venues Closed	1	One hot encoded
Retail Closed	0	One hot encoded
Night Clubs Closed	1	One hot encoded
All Face to Face Businesses Closed	0	One hot encoded.
Curfew	0	One hot encoded
Primary Schools Closed	1	One hot encoded
Secondary Schools Closed	1	One hot encoded
Universities Closed	1	One hot encoded

Table 3.1: Table of Data before Processing as per [13]

Processed Countermeasures	Notes
Night Clubs Closed	
Gastronomy Closed	
Leisure Venues Closed	
Retail Closed	
Curfew	
Primary Schools Closed	
Secondary Schools Closed	
Universities Closed	
Public Indoor Gathering Limit - 1	No gatherings allowed
Public Indoor Gathering Limit - 2	Gatherings of max size 2
Public Indoor Gathering Limit - 10	Gathering size limit ≤ 10
Public Indoor Gathering Limit - 30	Gathering size limit ≤ 30
Extra Public Indoor Household limit	Was there a limit of 2 households put on public indoor gatherings
Private Indoor Gathering Limit - 1	No gatherings allowed
Private Indoor Gathering Limit - 2	Gatherings of max size 2
Private Indoor Gathering Limit - 10	Gathering size limit ≤ 10
Private Indoor Gathering Limit - 30	Gathering size limit ≤ 30
Extra Private Indoor Household limit	Was there a limit of 2 households put on private indoor gatherings
Mandatory Mask wearing ≥ 3	Is the mask strigency rating 3 or greater

Table 3.2: Table of Data after Processing as per [13]

3.5 Results

We will briefly discuss the results found by Sharma et al using their model. Recall that the key point of their model was to infer the reduction in $R_t^{(r)} = R_{\text{basic}}^{(r)} R_{cms,t}^{(r)} R_{noise,t}^{(r)}$ when various countermeasures were in play. This is done by considering the vector of parameters γ that we set when defining $R_{cms,t}^{(r)} = \exp(-\gamma \cdot \mathbf{x}_t^{(r)})$.

In particular, activating the ith countermeasure means changing $x_{i,t}^{(r)}$ from 0 to 1. This in turn means $R_{cms,t}^{(r)}$ is scaled down by a factor of $\exp(-\gamma_i)$. Hence, by considering the distribution of $\exp(-\gamma)$, Sharma et al are able to infer the reduction in $R_t^{(r)}$due to each countermeasure considered in table 3.2.

They found that closing educational institutions was not as effective as closing

non-essential businesses, nor banning all gatherings. To see their results in more detail refer to their paper [13].

3.6 Limitations of this Model

The model discussed provided a framework to find the various reductions in R due to different countermeasures, however like any model it makes a number of assumptions. There are two main assumptions that we will investigate in this project.

1. The model assumes that each countermeasure reduces R completely independently of every other countermeasure. For example, if introducing curfew gave a reduction of 13% (meaning $\gamma_{\text{curfew}} = 0.14$) and closing gastronomy reduces R by 12% (meaning $\gamma_{\text{gastronomy}} = 0.13$), then introducing curfew and closing gastronomy must reduce R by 24% (as the reduction due to these two countermeasures is $1 - \exp(-\gamma_{\text{curfew}} - \gamma_{\text{gastronomy}}) = 0.24$).

2. Recall that $R^{(r)}_{noise,t} = \exp(z^{(r)}_t)$. The model makes an assumption that $z^{(r)}_t$ follows a random walk.

In the sections that follow, we will test these two assumptions. We will begin in chapter 4 by investigating whether there are any interactions between the countermeasures. We will demonstrate that the interactions are in fact significant.

We then turn our attention towards prediction. We note that the model described so far is used for inference purposes - specifically to identify the percentage reduction in $R^{(r)}_t$due to various countermeasures. It is not built specifically to predict what COVID rates will look like in the future. We therefore turn our focus towards the task of prediction in chapter 5, where we look at the current models, metrics and best practices to prepare us for the next chapters where we introduce our new work.

In chapters 6 and 7 we investigate whether a random walk is the best way to model $R^{(r)}_{noise,t}$. Chapter 6 will consider statistical time series, while chapter 7, our main novel work, fully augments the semi-mechanistic process with neural networks. This neural network will attempt to spot patterns in the raw data to find R, hopefully increasing the model's predictive power.

We also note here another limitation of this model which we will not look into, but is nonetheless worth mentioning. Epidemiological models typically benefit from having an inbuilt population limit, that prevents the number of cases from growing larger than the population limit of a region. This means that infections saturate over time. We do not investigate this due to a lack of population data, but note

that when this data is available this is relatively easy to implement. It can be done simply by adapting equation 3.1 slightly, as was done by Bhatt et al in [8].

4

Countermeasure Interactions

Contents

We now begin our first chapter in which we introduce new work. Semi-mechanistic models are common place in analysing the effects of interventions during the COVID pandemic [8], [9], [13]. Previous models, like the one just considered, assume that each intervention reduces $R_t^{(r)}$by a set percentage. Up until now, no model allowed the countermeasures to interact with each other. That is to say that the percentage reduction due to a given countermeasure was entirely independent of what other countermeasures were active. We aim to investigate the possible interactions between the interventions in our model.

The model described in the previous chapter aimed to infer the percentage reduction in $R_t^{(r)}$due to COVID countermeasures. They assumed that

$$R_{cms,t}^{(r)} = \exp(-\gamma \cdot \mathbf{x}_t^{(r)})$$

where $\mathbf{x}_t^{(r)}$ is a one hot encoding of the active countermeasures at time t in region r (see 3.2). This makes their model highly interpretable, as the value $\exp(-\gamma_i)$ represents the percentage reduction in $R_t^{(r)}$due to the ith countermeasure being active.

This form for $R_{cms,t}^{(r)}$ however also forces the countermeasures to act completely independently of each other. If the model learns the reduction in $R_t^{(r)}$due to just the ith countermeasure being active (ie $\exp(-\gamma_i)$), and also learns the reduction due to the jth (ie $\exp(-\gamma_j)$), then the reduction due to both countermeasures i and j being active is forced to be $\exp(-\gamma_i - \gamma_j)$. This means there is no possibility for there to be an interaction between countermeasures i and j in their model. For some countermeasures this seems like a reasonable assumption. For example, one might imagine that the effect of closing universities could be independent from the effect of closing secondary schools, as these largely effect different parts of the population. However this assumption does not always seem reasonable.

We aim to investigate these interactions so that we can see whether applying multiple countermeasures at the same time could have counter-intuitive effects. We limit ourselves to pairwise interactions. This chapter doesn't use deep learning and so we keep it relatively short to allow us to move quickly on to the main work done in this project. However the focus for this chapter is about potentially helping to inform healthcare policy, rather than focusing on the simplistic method that we have used, which in the author's opinion is less interesting. We begin by explaining our hypotheses for what these interactions might be, before moving on to the method that we use, and the results that we obtain.

4.1 Hypotheses

To aid in our discussion of what we expect the interactions to look like, it will help to collect the countermeasures used in the model into a number of groups. These groups are shown in table 4.1.

The rationale between these groupings is as follows. While group 1 collects the various face-to-face businesses together, group 2 deals with wide ranging interventions. Group 3 and Group 4 collect the public and private gathering limits together respectively. Lastly group 5 contains the closure of different educational institutions.

When we talk of a 'positive interaction' between two countermeasures we mean that the reduction due to both being active is more than the reduction due to just the first countermeasure, multiplied by the reduction due to just the second countermeasure. This is a good thing - it means that the two interventions are

Group 1: Face-to-Face Businesses	
Night Clubs Closed	Gastronomy Closed
Leisure Venues Closed	Retail Closed

Group 2: Wide Ranging Interventions	
Curfew	Mandatory Mask Wearing ≥ 3

Group 3: Public Gathering Limits	
Public Indoor Gathering Limit - 1	Public Indoor Gathering Limit - 2
Public Indoor Gathering Limit - 10	Public Indoor Gathering Limit - 30
Extra Public Indoor Household Limit	

Group 4: Private Gathering Limits	
Private Indoor Gathering Limit - 1	Private Indoor Gathering Limit - 2
Private Indoor Gathering Limit - 10	Private Indoor Gathering Limit - 30
Extra Private Indoor Household Limit	

Group 5: Educational Institutions Closing	
Primary Schools Closed	Secondary Schools Closed
Universities Closed	

Table 4.1: Groupings for the Countermeasures

better together than they are on their own, and synergistically help prevent the spread of COVID. Similarly, when we speak of 'negative interactions' we mean that the two countermeasures acting independently would have had greater effect than them acting together. We will later quantify the strength of the interaction by finding the extra reduction due to two countermeasures interacting, rather than the reductions due to both acting independently.

Another way to think of these interactions is to consider an ordering to the countermeasures. While the model does not actually take into account any ordering of the countermeasures, it will help to clarify what we mean when we talk about positive and negative interactions. Imagine two worlds A and B, where countermeasure i is active in world A but not in B. If we now activate countermeasure j in both worlds, what do we believe the effect would be? If world A subsequently gets a greater percentage reduction in $R_t^{(r)}$than in world B then we say i and j interact positively, and otherwise we say they interact negatively.

Of course, our model does not take into account any sense of ordering, and

so we must be careful to not read too much into any results by assuming that one countermeasure comes before the other. We now hypothesise what we would expect the interactions to look like.

Hypothesis 1: Face-to-face business x Face-to-face business

We expect that closing face-to-face businesses in group 1 will be largely independent from each other, as each business is independent from each other. One cannot replace going to, say, a night club with going to a retail shop, and so we imagine we will not see any interactions, as closing night clubs should therefore not impact retail shops.

Hypothesis 2: Wide ranging interventions x Everything

We expect when the wide ranging interventions (group 2) interact with all other interventions that we will generally produce negative interactions. This is because if a general countermeasure such as a night time curfew or mandated masks are enforced, then we imagine this makes gatherings less dangerous. Hence, subsequently limiting gatherings will reduce $R_t^{(r)}$by less than it would have done had masks not been worn. This means that we should expect negative interactions.

Hypothesis 3: Gathering Limits x Face-to-face business

We imagine that public and private gathering limits (groups 3 and 4) will interact positively with the closure of face-to-face businesses (group 1). This is because when the various face-to-face businesses are closed, people would stop socialising in restaurants or night clubs, and instead socialise in places where they still can, such as their homes. This mean $R_t^{(r)}$may not be reduced massively, as people are still socialising. However we believe that if social gatherings were also not possible, then we would see a much greater reduction in $R_t^{(r)}$, as there is now nowhere for people to socialise. This means we expect to see a positive interaction, as both interventions together prevent people from socialising, while only doing one of them may just change the place that people socialise.

We expect that the more stringent the limit on gatherings is, the more positive this interaction becomes. This is because the more constraining the limit, the less likely it is that someone could simply shift from socialising at a restaurant to socialising at home. Hence the more constraining the limit, the greater the effect it would have in tandem with business closures.

Hypothesis 4: Closing Education x Closing Education

We expect that the countermeasures regarding closing educational institutions (group 5) are all independent from each other, as they impact different age groups.

Hypothesis 5: Gathering Limits x Gathering Limits

We expect that the various limits on public gatherings (group 3) will have insignificant interactions with each other. This is because it is impossible to have, say, a public gathering limit of 2 without a public gathering limit of 10. This means it is impossible to distinguish between what reduction in $R_t^{(r)}$is due to the limit of 2, and what reduction is due to the limit of 2 interacting with the limit of 10, as there will never be a situation in which we see a limit of 2 without the limit of 10. We expect the same to be true for the limits on private gatherings (group 4).

Hypothesis 6: Gathering Limits x Gathering Limits

We expect limits on public gatherings (group 3) to have positive interactions with limits on private gatherings (group 4). This is because we imagine that if only private gatherings are limited then, in general, people might just attend public gathering instead, and vice versa. This means that limiting just one of these is presumably not that useful, but limiting both would be - due to this, we expect them to have a positive interaction.

4.2 The Method

To investigate a number of pairwise interactions between the countermeasures, we consider the set $\mathcal{I}$, which contains the (unordered) pairs of countermeasure indices that we would like to consider the interactions of. For example, if we would like to investigate the interaction between the ith and jth countermeasures, then we have $(i, j) \in \mathcal{I}$.

Our idea was to change the form of $R_{cms,t}^{(r)}$ by adding in terms representing the pairwise interactions due to the pairs in $\mathcal{I}$. We also remove the $R_{noise,t}^{(r)}$ term that came from [13] to ensure that all changes in $R_t^{(r)}$are due to the countermeasure term.

We then define

$$R_t^{(r)} = R_{\text{basic}}^{(r)} R_{cms,t}^{(r)} \tag{4.1}$$

with $R^{(r)}_{\text{basic}}$ constant in time, and

$$R^{(r)}_{cms,t} = \exp\left(-\gamma \cdot \mathbf{x}^{(r)}_t - \sum_{(i,j)\in\mathcal{I}} \gamma_{i,j} AND(x^{(r)}_{i,t}, x^{(r)}_{j,t})\right) \tag{4.2}$$

The first term in 4.2, as before, encapsulates the individual effects of each countermeasure, as each γ_i reduces $R^{(r)}_{cms,t}$ only when $x^{(r)}_i = 1$. The second term represents the interactions. It is only when both $x^{(r)}_i = 1$ and $x^{(r)}_j = 1$ that $\gamma_{i,j}$ reduces $R^{(r)}_{cms,t}$, and so we can think of $1 - \exp(-\gamma_{i,j})$ as the extra reduction in $R^{(r)}_t$due to countermeasures i and j interacting.

This means that if $1 - \exp(-\gamma_{i,j})$ is positive that the reduction in $R^{(r)}_t$due to countermeasures i and j being active is more than the reduction due to just the ith countermeasure, multiplied by the reduction due to just the jth countermeasure. Similarly if this quantity is negative then it means that the two countermeasures acting independently would have had greater effect than them acting together.

To consider all interactions, we ideally would place all 171 possible pairs in $\mathcal{I}$ and run the model with $R^{(r)}_t$being defined as written in equations 4.1 and 4.2. However, we ran our model using this, and found that this produced too much uncertainty. Instead we split these 171 interactions into 34 sets $\mathcal{I}_1, ..., \mathcal{I}_{34}$, so that every pair of countermeasures was in (at least) one $\mathcal{I}_k$. For each of these 34 sets, we ran our model with $R^{(r)}_t$being defined as written in equations 4.1 and 4.2, with $\mathcal{I} = \mathcal{I}_k, k = 1, ..., 34$. We then take the strength of the pairwise interaction between the ith and jth countermeasures to be $1 - \exp(-\gamma_{i,j})$, where we look at the model in which $(i, j) \in \mathcal{I}_k$. We will later show in section 4.5 that our results are robust to how we split the interactions up, suggesting that this method is valid.

4.3 Results

The results that we discuss are all oriented towards understanding the impact of countermeasures in reducing the spread of COVID. We focus not on the machine learning, but on the health policy consequences for these results in this section.

Within a model that considers the interaction between countermeasures i and j, each sample gives a different value of $1 - \exp(-\gamma_{i,j})$, which represents the reduction in $R^{(r)}_t$due to the interaction. We consider the mean of this across these samples, as well as the standard deviation. We consider the interaction between the ith and jth countermeasures to be significant if the mean of $1 - \exp(-\gamma_{i,j})$ is more than 2 of its standard deviations away from 0. Otherwise, there is enough uncertainty in the

value of the interaction that one could reasonably imagine its effect to be negligible.

We produce a heat map (see fig 4.1) to show the strength of the significant interactions that were inferred by our model. We will now compare this heatmap to our hypotheses. When our results do not align with our hypotheses we will try to explain potential explanations for this if possible.

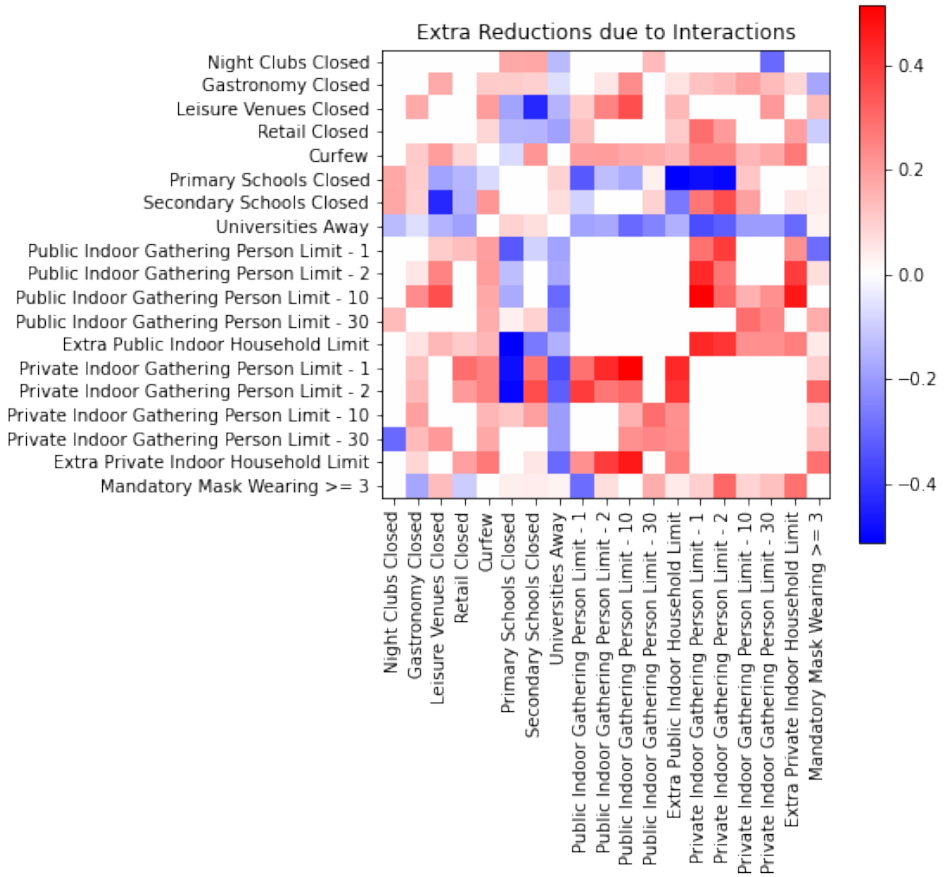

Figure 4.1: Heat map showing the extra reduction due to each pair of countermeasures interacting

Hypothesis 1: Face-to-face business x Face-to-face business

Our hypothesis that the face-to-face businesses would not interact was correct for all pairs of businesses except one. We see that the only significant interaction within this group was seen between closing gastronomy and closing leisure venues,

which had a positive interaction. Perhaps this is because these are the two main face-to-face businesses that allow large numbers of people to socialise. Hence closing only one of these may simply encourage people to do the other. It is only when both are closed together that people are unable to socialise. This explains why these two businesses interact.

Hypothesis 2: Wide ranging interventions x Everything

Our hypothesis that the wide ranging interventions (curfew and mask rules) would generally interact negatively with other interventions was entirely unfounded. In particular, we find that a night time curfew had a positive interaction with almost all other interventions. Perhaps this interaction is down to the fact that face-to-face businesses operate during the day, and closing these might mean that people shift some of their socialising to later times. Similarly, night time curfews would cause people to socialise at earlier times than they might have done otherwise. Hence only implementing one of these interventions may just cause people to change the time at which they socialise. It is only when both interventions are in play that people are not able to socialise as much, and so we get a positive interaction between curfews and closing face-to-face businesses.

Hypothesis 3: Gathering Limits x Face-to-face business

Our hypothesis that introducing gathering limits would interact positively with the closures of face-to-face businesses was accurate. However, we were wrong to suggest that we would get greater interactions when the limits were smaller. It looks like there is a tendency for the opposite to be true, however our robustness analysis which we perform later suggests that this isn't significant.

Hypothesis 4: Closing Education x Closing Education

Our expectation that closing the different educational institutions are independent from each other was accurate. There appears to be some lightly positive interactions between universities closing and the closures of both primary and secondary schools. Perhaps this is because just closing the academic side of a university did not prevent students from socialising and inadvertently spreading COVID. However when more serious interventions were put in place, such as the closures of primary and secondary schools, perhaps students were more likely to take restrictions seriously.

Hypothesis 5: Gathering Limits x Gathering Limits

This hypothesis was shown to be completely accurate. No interactions were seen whatsoever between the public gathering limits, and none were seen between the private gathering limits.

Hypothesis 6: Gathering Limits x Gathering Limits

This hypothesis was also shown to be accurate. We see only positive interactions between all of the public gathering limits and the private gathering limits.

4.4 Discussion

Now that we have gone through each of our hypotheses it is worth looking at the big picture. In general we see that most of the restrictions interact positively. This suggests that we do not get diminishing returns when adding more and more restrictions, which is a good sign for the efficacy of these interventions. This means that governments can expect the interventions that they put in place to behave synergistically together. This may be attributable to the fact that when any one restriction is in place, people generally find a way to change their behaviour to carry on socialising. It is only when multiple restrictions are in place, preventing socialising in more of its forms, that people actually stop. Hence restrictions typically interact to further reduce the spread of COVID. Further, introducing more interventions means people may take the rules more seriously, meaning that they are more likely to stick to them. Hence, typically interactions are positive.

While most of the restrictions interact positively with each other, there is one group of countermeasures that generally produced negative interactions. These were the educational institutions. It appears that we have diminishing returns on closing education on top of the other restrictions. This could be used to further the case that closing education is less effective than the other COVID interventions. Sharma et al's results suggested that closing education did not reduce R as much as the other interventions [13], and our results now suggest that this effect is further weakened when other interventions are in play.

We must caveat these results about educational establishments however with some warnings. The first is that closing schools and universities targets young people, while the case and death data available to us is for a population as a whole. Young people tend to have more asymptomatic cases and so they will not be considered in the reported case data as much as older people. Hence while

closing educational establishments might actually lower R, this would not be seen in the case data. The second caveat deals with how we defined the intervention of 'Universities Away'. When the one hot encoding of the countermeasures $\mathbf{x}_t^{(r)}$ was produced, the value of the 'Universities Away' countermeasure was set in a slightly counter intuitive way. If a region did not contain any universities, then it was assumed that the 'Universities Away' countermeasure was active. This means that our model cannot see the difference between a university closing, and there just being no university. Due to both of these caveats, we should take our results regarding school and university closures with a pinch of salt, and we note further analysis is required to make any strong statements based on our results.

4.5 Robustness

To generate the heatmap 4.1 recall that we split each of the 171 possible interactions into 34 sets $\mathcal{I}_1, ..., \mathcal{I}_{34}$. We then took the reduction in $R_t^{(r)}$due to the interaction between countermeasures i and j to be calculated from the model that had $(i, j) \in \mathcal{I}$.

We hope that the results we obtained are robust towards our split of interactions. We therefore rerun our program using randomised splits of the 171 pairs of countermeasures. The results are seen in figure 4.2.

We have run 3 different splits of the interacting countermeasure pairs, and we find that in all 3 splits the results look very similar. We are therefore pleased that our results were robust, and therefore presumably accurate.

To further verify the robustness of these results, it would be interesting to see whether the heatmaps can be replicated using disjoint subsets of the data, either by splitting the regions into distinct sets, or by looking at shorter periods of time. This would show that our data is robust in more ways than we have already shown. We do not do this, but note that with more time we could do this.

Extra Reductions due to Interactions
Night Clubs Closed
Gastronomy Closed
Leisure Venues Closed
Retail Closed
Curfew
Primary Schools Closed
Secondary Schools Closed
Universities Away
Public Indoor Gathering Person Limit - 1
Public Indoor Gathering Person Limit - 2
Public Indoor Gathering Person Limit - 10
Public Indoor Gathering Person Limit - 30
Extra Public Indoor Household Limit
Private Indoor Gathering Person Limit - 1
Private Indoor Gathering Person Limit - 2
Private Indoor Gathering Person Limit - 10
Private Indoor Gathering Person Limit - 30
Extra Private Indoor Household Limit
Mandatory Mask Wearing >= 3
0.4
0.2
0.0
−0.2
−0.4
−0.6

Figure 4.2: Sensitivity Analysis of our Interaction Results. We see that all of our heat maps look similar, despite using different splits of the pairs of countermeasures.

5

Prediction: Current models, metrics and best practice

Contents

Now that we have ticked off our first objective in this project, of investigating the interactions between COVID interventions, we turn towards our next task of prediction. In this project we produce two methods to predict the future of a pandemic, known as Epi-ARMA and Epi-NN. We would like to frame the task of prediction in a rigorous way that allows us to compare the methods we will introduce later with ones that are already out there. We explain here some background on

prediction, specifically applied to COVID, so that we are able to evaluate our models as we introduce them in chapters 6 and 7.

We also would like to distinguish between 'inference' and 'prediction' here. When we talk about inference, it is to find the value of some parameters and time series during times at which data was seen by the model. This means that the model's outputted expected cases and deaths during the times at which data was seen are inferences. When these time series are found outside of the times where data is observed then we refer to it as a prediction.

In this section we will begin by explaining how we split our data into different datasets for the purpose of choosing hyperparameters, so that we can fairly compare different models on the same data. We then discuss some current methods to predict COVID, and also explain some metrics that we can use to compare different forecasts. Lastly we attempt to use and tune previously produced COVID models so that we are able to compare them to our models later.

5.1 Splitting the Data

Both of the models we produce will have hyperparameters that need tuning. As is always the case in machine learning, we should not test our models on the same dataset that we used to choose our hyperparameters. If we do this, we risk overfitting our model to the noise in the data, and then our model may not generalise to unseen datasets. Hence we explain briefly how we split our dataset.

The data we will have access to runs from 01/08/2020 until 22/01/2021, which is 175 days worth of case and death data. We split this as follows:

1. Data from the 01/08/2020 until 10/12/2020 makes up our training set. This is 132 days of data.

2. Data from 11/12/2020 until 02/01/2021 makes up our validation set. This is 23 days of data.

3. Data from 03/01/2021 until 22/01/2021 makes up our test set. This is 20 days of data.

To find the best value of the hyperparameters for a given type of model, we train a number of versions of the model with different hyperparameters on the training set. We then compare each of these using the NMSE and NCRPS metrics (which we will explain soon in section 5.3) on the validation set, choosing the hyperparameters that

perform the best. After this validation process, we rerun our model, training it on both the training set and validation set, and find out how it performs on the test set.

In this way, we ensure that when we finally test our model, it has not been overfit to the data that we are testing it on.

5.2 Previous Prediction Methods

We consider two previously built methods to predict future COVID rates. The first, EpiNow2 is a semi-mechanistic model that finds the parameters of its epidemic model using Bayesian machine learning [6]. The second, Prophet, is a pure machine learning model that has no inbuilt understanding of epidemics [4].

5.2.1 EpiNow2

EpiNow2 has an inbuilt understanding of epidemics and is implemented in R. On its default setting, it assumes that R_t follows a Gaussian Process, and then uses this to find the number of infections. These infections are used to determine the number of reported cases, which is the observed data in this Bayesian model. There are a number of other settings one could use, specifically it allows the R_t Gaussian Process to be considered to be other processes, such as a random walk. The main problem with EpiNow2 is it fits each region completely separately, meaning it cannot use data from one region to help fit others.

5.2.2 Prophet

Prophet is a pure machine learning model that attempts to fit data using an additive model [4]. To infer the value of $y(t)$, Prophet assumes that y is of the form

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t$$

where

- $g(t)$ represents the growth. This is piecewise and can be set to be either linear, or logistic in its form. The *changepoints* - the places where the different piece of $g(t)$ meet - can be inferred automatically, or inputted by the user.
- $s(t)$ represents the seasonality. This is periodic in form and is found by fitting Fourier Series.

- $h(t)$ represents the effects of holidays. This is non-periodic, and is dealt with by considering shocks in the data due to a given holiday. We do not use this term in our use of Prophet in this project.
- ϵ_t is the added noise.

Prophet quickly fits these terms, and extrapolates into the future, where the growth is of the form learned by the final 'piece'.

5.3 Metrics

To compare two models we need some metrics to compare their forecasts. We used two metrics in this project: the normalised mean square error and the normalised continuous ranked probability score. Both of these require comparing a model's inferred or predicted time series with the true values of the time series. For example, a model will output an 'expected cases' distribution over all time. We need metrics that will allow us to compare this to the observed cases on each day.

Our models each output a number of samples of what it believes to be the expected cases and deaths at each time and region. The sth sample for the expected cases and deaths on day t and in region r are denoted by $\tilde{c}_{t,s}^{(r)}$ and $\tilde{d}_{t,s}^{(r)}$ respectively. We would like to compare these to the actual observed cases and deaths on day t and in region r, $c_t^{(r)}$ and $d_t^{(r)}$.

5.3.1 Normalised Mean Square Error

The normalised mean square error (NMSE) is a simple way to calculate how good a model is. To find the NMSE for the expected number of cases we use the following formula:

$$NMSE(t) = \frac{1}{nRs} \sum_r \left(\frac{median_s(\tilde{c}_{t,s}^{(r)}) - c_t^{(r)}}{K^{(r)}}\right)^2$$

where nRs is the number of regions, $K^{(r)} = \frac{1}{nDs} \sum_t median_s(\tilde{c}_{t,s}^{(r)})$ is the normalising constant, and nDs is the number of days of data. Note $K^{(r)}$ is the time-averaged median expected cases in a region. The reason we normalise this metric, rather than simply setting $K^{(r)} = 1$, is that some of the regions have hugely different scales. This means that a metric that wasn't normalised is very heavily weighted towards the regions with larger scales, something we do not want to happen.

This finds the normalised squared error between the models median expected cases and the true time series, and then averages this over all of the regions. We track how the NMSE changes over time. The NMSE for the expected number of deaths is found in an identical fashion.

We will also consider the value

$$\widehat{NMSE} = \sum_{t \in future} NMSE(t)$$

which allows us to consider a single number, rather than the NMSE time series. We will only sum times in the future as these are the times which we are validating and testing our model on. Otherwise we are partly evaluating our predictive model based on how it behaves on data that was used to train it, which is not appropriate.

While this metric is good at giving us a general sense of if our model is predicting reasonable values for the future cases and deaths, it doesn't take into account any uncertainty in our predictions. For example, if two models have the same median expected cases, but one of them is (unreasonably) certain about this while the other is (more reasonably) uncertain, then the NMSE metric cannot distinguish between these two models. Hence NMSE is a good indicator of if our median prediction is good, but it tells us nothing more than this.

5.3.2 Normalised Continuous Ranked Probability Score

The normalised continuous ranked probability score (NCRPS) is a more appropriate metric for probabilistic forecasts as it takes into account uncertainty in the model. We define $F_t(x)$ to be the cumulative distribution function of the sampled output at time t, and $c_t^{(r)}$ to be the true value of the data. The NCRPS for the expected cases takes the form:

$$NCRPS(t) = \frac{1}{nRs} \sum_r \frac{1}{K^{(r)}} \left(\int_{-\infty}^{c_t^{(r)}} F_t(y)^2 dy + \int_{c_t^{(r)}}^{\infty} (1 - F_t(y))^2 dy \right)$$

where, as before, nRs is the number of regions, $K^{(r)} = \frac{1}{nDs} \sum_t median_s(\tilde{c}_{t,s}^{(r)})$ is the normalising constant, and nDs is the number of days of data. We reiterate that we normalise this, rather than setting $K^{(r)} = 1$, so that we don't weight this metric towards the regions with much larger populations. This is a more complicated looking metric than the NMSE, but it allows us to compare the model's outputted distributions, rather than just the medians. We also consider

$$\widehat{NCRPS} = \sum_{t \in future} NCRPS(t)$$

in analogy to $\widehat{NMSE}$. We will use both NMSE and NCRPS when considering how good a model is, both on the predicted case data and on the predicted death data. In this way, we have 4 metrics to track how good a model may be.

5.4 Predicting with EpiNow2 and Prophet

We have already explained how EpiNow2 and Prophet work, and now we will try to tune them using the validation set and then test them on the test set using the metrics just described.

5.4.1 EpiNow2

EpiNow2 has a number of different latent processes that can inform its inference of $R_t^{(r)}$. However, we found that running EpiNow2 with a sufficient number of warmup steps took a long time (9 days), making it unwieldy to validate. This speed problem came from the fact that EpiNow2 predicts the future in each region separately, and we have 114 regions to predict. Due to this time problem, we ran it on its default settings. This means we cannot expect our metrics for EpiNow2 to be perfectly accurate. We only use EpiNow2 to give a very rudimentary baseline for the predictive power of our model.

Further, we have only used EpiNow2 to predict future cases, and not future deaths, as this was also its default setting. It is able to handle reporting delays in the case data, meaning that its outputs have a strong periodic component. We plot in figure 5.1 an example of the epinow predictive output.

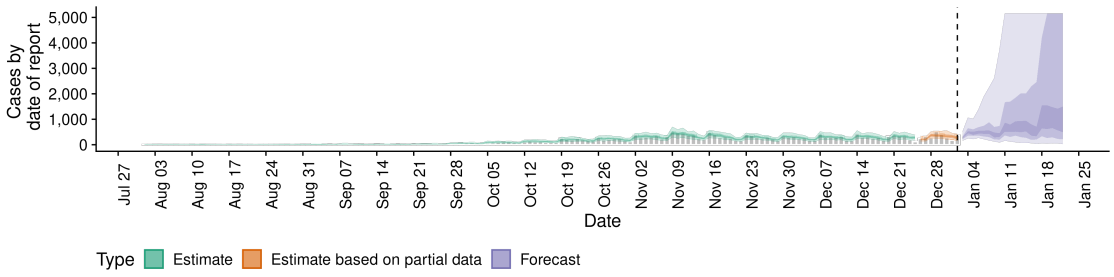

Figure 5.1: Plots of Predicted Cases over time in Lincolnshire using EpiNow2.

5.4.2 Prophet

Prophet is much faster to train, owing to it not modelling the infection process itself. We have the option to use either piecewise linear fits or piecewise logistic. Clearly for an epidemic where trajectories are exponential in nature we should be considering the logistic growth fit.

Using the logistic growth setting requires some hyperparameters that we will need to tune. These are the 'floor' and 'cap', representing the lower and upper saturation points of the logistic growth. We will need to find the floor and cap for both predicting cases and deaths, giving us 4 hyperparameters to find. Clearly the 'floor' for both case and death data should be 0 - otherwise our model cannot infer any 0 data, which is something we would like it to do. Similarly we want the 'cap' for the case data to be greater than the maximum case load in any one region (11489 cases in Lombardy on 07/11/2020), and similarly for the 'cap' on death data (347 deaths in Lombardy on 03/12/2020).

By running Prophet on the training set with various 'caps', we check its NMSE and NCRPS on the validation set. We try the caps listed in table 5.1, and we show our results in this table and in figure 5.2.

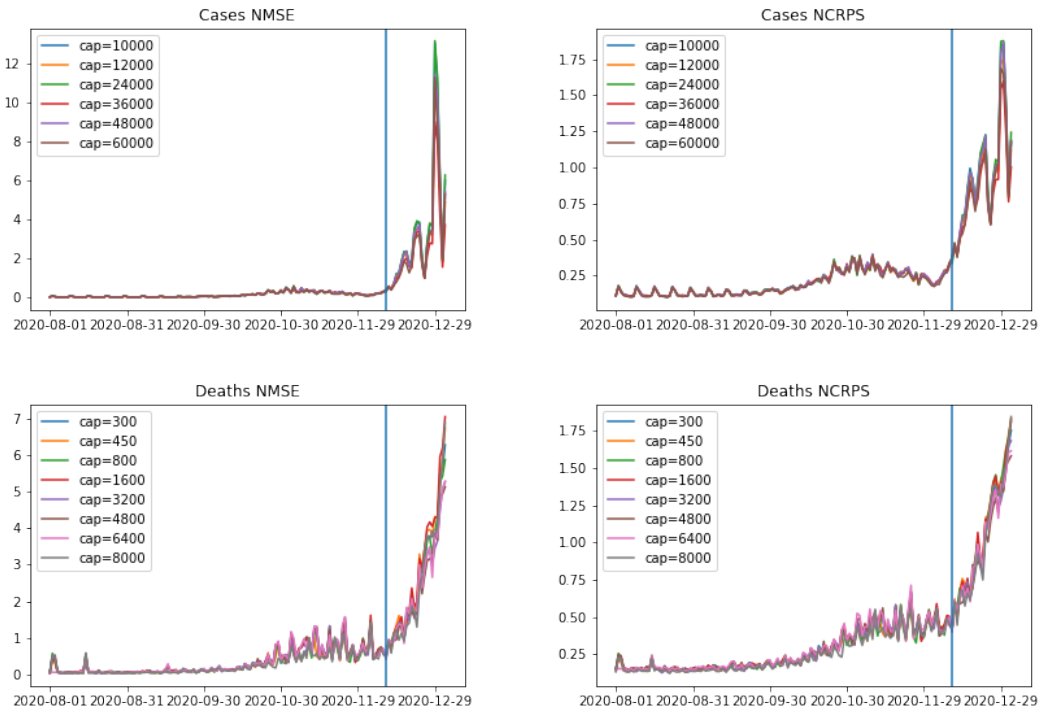

Figure 5.2: Plots of NMSE and NCRPS over time for the predicted number of cases and deaths with various caps. The vertical line marks the point at which the data ends, and prediction starts.

Case Prediction			Death Prediction		
Cap	$\widehat{NMSE}$	$\widehat{NCRPS}$	Cap	$\widehat{NMSE}$	$\widehat{NCRPS}$
10000	76.7	22.0	300	61.7	23.9
12000	68.9	21.3	450	64.3	24.7
24000	78.0	22.2	800	60.8	24.9
36000	59.1	19.7	1600	66.9	24.6
48000	71.8	21.9	3200	55.6	23.4
60000	66.0	20.5	4800	55.1	22.9
			6400	56.8	23.5
			8000	59.9	23.6

Table 5.1: Validation of Prophet Hyperparameters. We use this table to select the best cap hyperparameter for case prediction and for death prediction. Smaller NMSE and NCRPS is better. Rounded to nearest whole number

We can see that the best results here come from using a case cap of 36000, and a death cap of 4800. These will be the hyperparameters we use when we test Prophet on the test set later.

We note that like EpiNow2, Prophet has a periodic component that gives it the flexibility to model the weekly reporting cycle seen in case data. We see this in figure 5.3.

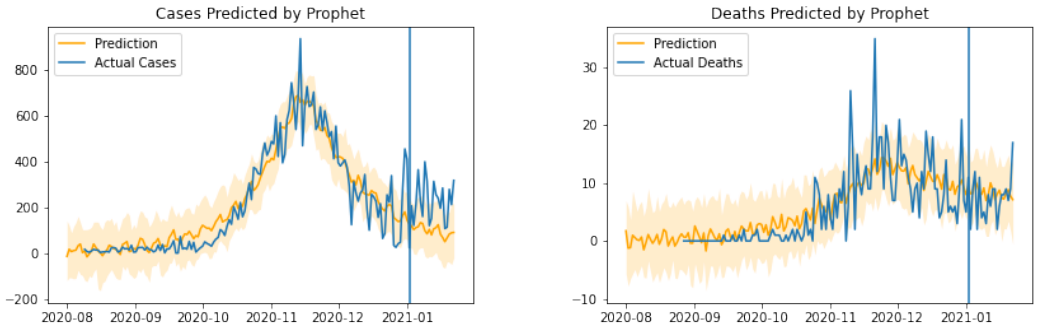

Figure 5.3: Plots of Predicted Cases and Deaths over time in Abruzzo using Prophet. Shaded areas represent the 90% confidence interval, and the vertical line represents the point at which the data inputted into Prophet ends.

Now that we have discussed two prediction methods that will act as a baseline for us, we will begin the task of explaining the first of the predictive models that have been built as part of this project. We aim to try and beat these baselines.

6

The ARMA Predictor: Epi-ARMA

Contents

We will now begin our description of our first predictor, Epi-ARMA. We use a semi-mechanistic approach that models an epidemic by assuming that the noise in the reproductive number R_t follows an ARMA time series (we will explain ARMA series in section 6.2.1). The resultant epidemiological model we produce is known as Epi-ARMA. In particular, the aim was to find a distribution over possible future cases and future deaths in the future for each region, as well as predicting $R_t^{(r)}$. In this way, we predict future cases and deaths in a highly interpretable way. We used the code of Sharma et al as our base code [12], and then adapted it. When an adaptation from the original model was done we will explain it, but if not mentioned

we will be using the framework already discussed.

This chapter investigates our first attempt at producing a model to predict future COVID rates. While this does not include any deep learning, this first attempt at prediction gave an understanding of how one would convert Sharma et al's model from one that can infer the value of R at times at which we have data, to one in which we can predict values of R at future times.

6.1 Prediction via Decomposing $R_t^{(r)}$

To predict the future values of $R_t^{(r)}$ we looked into the decomposition considered already in section 3.2.2.

$$R_t^{(r)} = R_{\text{basic}}^{(r)} R_{cms,t}^{(r)} R_{noise,t}^{(r)}$$

where

- $R_{\text{basic}}^{(r)}$ is a constant in each region
- $R_{cms,t}^{(r)} = \exp(-\gamma \cdot \mathbf{x}_t^{(r)})$ depends only on the active countermeasures in place in a region, $\mathbf{x}_t^{(r)}$
- $R_{noise,t}^{(r)} = \exp(z_t^{(r)})$ has $z_t^{(r)}$ follow a random walk in time, where we assume it only changes on a weekly basis. We also set the standard deviation of this random walk σ_{noise} to be inferred.

This decomposition consists of three terms, the first is the basic value of $R_t^{(r)}$, while the other two terms reduce this basic value based on the active countermeasures or noise. We will predict the future value of each of these terms in turn, and then use these to determine the value of $R_t^{(r)}$ in the future. This is enough to calculate the future cases and deaths in each region.

Once the model has inferred the value of $R_{\text{basic}}^{(r)}$ we note that this term should not be expected to change in the future. Hence in the future we would set this term to be the same as the value we inferred it to be.

When γ has been inferred we note that we can now calculate $R_{cms,t}^{(r)}$ at any time when the active countermeasures $\mathbf{x}_t^{(r)}$ are known. Hence if we are able to input the countermeasures that will be in place in the future then we are able to predict the future value of $R_{cms,t}^{(r)}$. We assume in this section that we have knowledge of which countermeasures will be set in the future, and so we have access to $\mathbf{x}_t^{(r)}$ at all times.

Lastly once we have sampled the value of $R^{(r)}_{noise,t}$ for all past times we can simply continue the random walk from the present. Each posterior sample defines all past values of $R^{(r)}_{noise,t}$, and it defines the standard deviation of the random walk σ_{noise}, so we can take the last inferred value of $R^{(r)}_{noise,t}$, and continue the random walk from there. We do this for each posterior sample individually.

In this way each sample from the posterior of the distribution can be modified to predict $R^{(r)}_t$ into the future, and so we get a posterior by looking at the spread of these samples. These trajectories of $R^{(r)}_t$ in turn give us posteriors over the future cases and deaths.

6.2 Generalising the Random Walk

Before looking at the results of this, we will generalise our random walk first. A random walk satisfies the following equations (where we ignore here the fact that we constrain our walk to only change at weekly intervals):

$$z_t = z_{t-1} + \epsilon_t$$

where $\epsilon_t \sim \mathcal{N}(0, \sigma_R^2)$.

To compute the next value of a random walk only two numbers are needed, the previous value and the standard deviation σ_R. This means the random walk has no knowledge of the history of the series before the previous value, and so there is no way for the random walk to learn a trend in the data. For example, if the inferred values of $z^{(r)}_t$ have a tendency to increase with time, this will not be captured by the continuation of the random walk.

In the original model in [13], this random walk was used to allow the value of $R^{(r)}_{noise,t}$ to change over time in such a way that it was able to fit the observed case and death data. We need it to do more than this as we are making predictions. In the future, where there are no observed cases or deaths, we need $R^{(r)}_{noise,t}$ to vary in a realistic way. This may require something different to a random walk. To see this, imagine if all of the datapoints lay on a straight line. The random walk would be able to fit this, but then its prediction of the future would still be a random walk, even though this is not a good extrapolation of the datapoints. Hence assuming $R^{(r)}_{noise,t}$ follows a random walk might not be the best assumption. We will use a generalisation of the random walk that allows us to consider momentum in the data, known as the ARMA model.

6.2.1 The ARMA Model

The ARMA model stands for Autoregressive-Moving-Average model. It, like the random walk, provides a way to sample the next term in the series $z_t^{(r)}$ using previous terms in the series. It is 'autoregressive' in that the next term depends on previous terms. The number of previous terms this is dependent on is denoted by the parameter p. It uses a 'moving average' of previous error terms to influence the next term. The number of previous error terms is denoted by the parameter q. Mathematically, the ARMA(p,q) model is:

$$z_t = c + \sum_{i=1}^{p} \phi_i z_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} + \epsilon_t \tag{6.1}$$

where $\epsilon \sim \mathcal{N}(0, \sigma_{ARMA})$.

Note the first two terms on the right hand side represents the autoregressive part of the model, and these tell us the expectation of the next z term. The third term on the right hand side is the moving average of the preceding noise values ϵ_t. The final term is the noise value at time t.

The benefit of using an ARMA(p,q) model over a random walk is that it is able to interact with previous terms in the series, as well as previous errors. Further we have not lost the random walk by using an ARMA series instead, as an ARMA(1,0) process with $c = 0$ and $\phi_1 = 1$ is the random walk.

6.2.2 The ARMA Parameters

There are a number of parameters required for the ARMA model. There are two discrete parameters (p and q) and $p + q + 2$ continuous parameters (c, ϕ_i, θ_i and σ_{ARMA}). We will treat p and q as hyperparameters for our model which must be user-defined. The continuous parameters however are more interesting.

We can learn these parameters by placing priors on them, and inferring their value using the model. We place priors of

$$\begin{aligned} c &\sim \mathcal{N}(0, 0.5) & \phi_i &\sim \mathcal{N}(0, 0.3) \\ \theta_i &\sim \mathcal{N}(0, 0.3) & \sigma_{ARMA} &\sim \text{HalfNormal}(0.15) \end{aligned}$$

In particular, we share these parameters across all of the regions in the model, so that we don't overfit the data in each region. We also note that setting $\sigma_{ARMA} \sim$ HalfNormal(0.15) is what the original random walk model did. This ensures that the value of σ_{ARMA} will be as small as possible (as the prior is larger for small values of σ_{ARMA}), while also fitting the data (otherwise the likelihood is very low). Hence this

hierarchical method (we sample σ_{ARMA} which is then used to sample ϵ_t) allows us to keep σ_{ARMA} as small as possible without underfitting the data. We also, similarly to the original random walk, only let the ARMA random walk change on a weekly basis.

Now that we have inferred good parameters for the ARMA series we are now able to continue any ARMA series into the future. In particular, this means we now have the ability to consider how $R^{(r)}_{noise,t}$ might look in the future.

There is a problem however. Predicting future values of the $R^{(r)}_{noise,t}$ ARMA series heavily depends on the values of the series at the present point, the time just before prediction happens. All of these values have been inferred by our model, and so we are constrained by the accuracy of our model. If the inferred value of $R^{(r)}_{noise,t}$ at the present point is inaccurate, then our prediction for the future values of it will also likely be inaccurate.

Now we note that the inference of the present value of $R^{(r)}_t$ is only done using past data, as there is no observed data after this point. We imagine therefore that the inference of what $R^{(r)}_t$ may be at the present time may be less accurate and more uncertain than at other times, when data exists for both before and after that time.

We introduced another hyperparameter to try and deal with this potential inaccuracy in the inference of $R^{(r)}_t$ at the present date.

6.3 Introducing *ignore_last*

We have already explained how we predict $R^{(r)}_t$ using the decomposition discussed. $R^{(r)}_{\text{basic}}$ doesn't change in time, and so we expect it to be constant. $R^{(r)}_{cms,t}$ only changes based on the active countermeasures $\mathbf{x}^{(r)}_t$, and so can be found in the future provided we input the $\mathbf{x}^{(r)}_t$ at future times. $R^{(r)}_{noise,t}$ follows the ARMA model, and so we can simply continue each sample onwards into the future, modelling it as ARMA with the parameters inferred by our model. Note importantly this requires our estimate of $R^{(r)}_{noise,t}$ at the end of the time series to be (relatively) accurate for this to work, as this is where the ARMA series is continued from. Unfortunately, this estimate is not necessarily accurate.

As we do not expect the ARMA $R^{(r)}_{noise,t}$ series to be perfectly accurate over the final few days, we will ignore our inferences on the value of this series over the last few days. We define a hyperparameter *ignore_last*, telling us how many of the last days to ignore, and then we continue the ARMA series from this point. This allows us to increase the uncertainty of the model over the final *ignore_last* days, which is needed as the model is typically overconfident over this time.

6.4 Posterior Plots

When we run our model, we must make sure that we have run our sampler for enough warmup steps that the samples taken are from the posterior of the model. If we do not run enough warmup steps, then the samples will not have converged to the posterior, and so any results will be useless.

The simplest way to do this is to check how well the model can infer the expected number of cases and deaths at the times at which it is able to see this data. We would hope that if the model has converged, that it would be able to infer the number of cases and deaths reasonably well.

We ran the ARMA predictor model for only 50 warmup steps due to time and computational constraints, but we note that this is enough to reach convergence to the posterior. We plot some graphs here (figure 6.1) so that we can verify this.

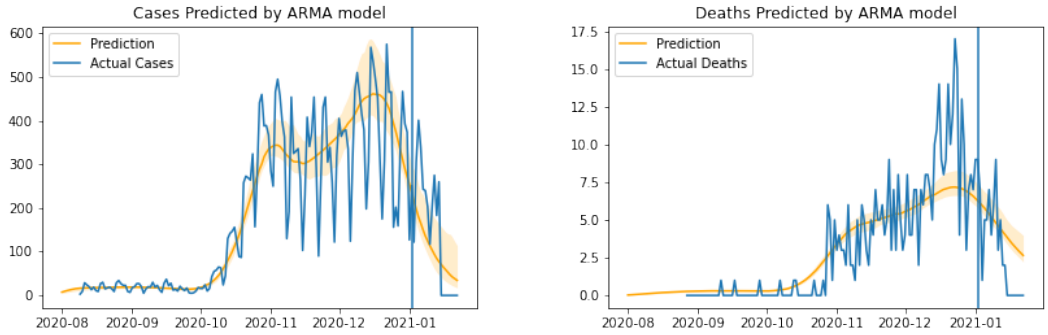

Figure 6.1: Checking the posterior by looking at how well the model can replicate the case and death data in Aargau. Time after the vertical line are predictions, where the model did not have access to any data. Shaded regions correspond to 90% confidence intervals.

Note that the model's inferred number of cases and deaths neatly goes through the actual data, and so we believe that our model has converged.

While we are looking at posterior plot we will also look at the values of each component of $R_t^{(r)}$here. The three constituent parts of $R_t^{(r)}$can be seen in figure 6.2, and we can see how these vary with time.

In particular we draw our attention to the fact that the noise term is able to flexibly allow $R_t^{(r)}$to change on a weekly basis, something that $R_{cms,t}^{(r)}$ cannot do, as it is constrained to only change when the countermeasures change.

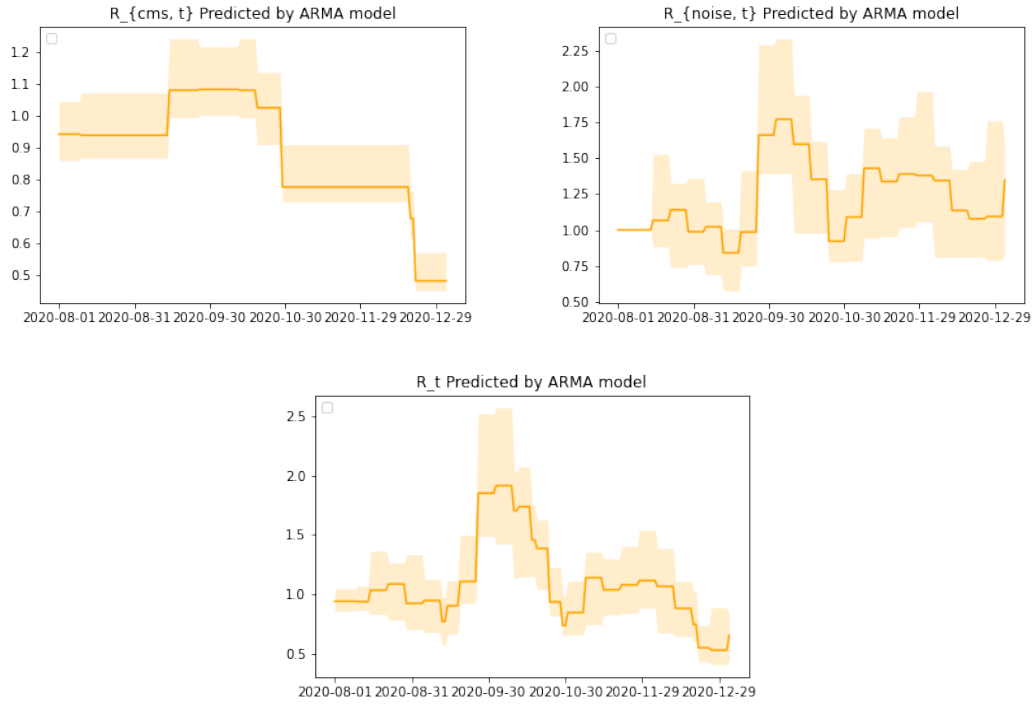

Figure 6.2: The decomposition of $R_t^{(r)}$. Top left shows $R_{cms,t}^{(r)}$, top right shows $R_{noise,t}^{(r)}$, and bottom shows $R_t^{(r)}$for the region of Aargau. Shaded area represents a 90% confidence interval.

6.5 Tuning the Hyperparameters

Epi-ARMA has 3 hyperparameters. The noise model is ARMA with hyperparameters p and q, representing the number of terms kept for the auto-regressive and the moving average parts of the process. After our Bayesian model has been run, and the ARMA coefficients have been learned, we then ignore *ignore_last* days from the end of the $R_{noise,t}^{(r)}$ term, and continue the ARMA process from that point. We would like to find the best values for the hyperparameters p, q and *ignore_last*.

We will find the NMSE and NCRPS for predicting the cases and deaths on the validation dataset for various combinations of these hyperparameters. We tried values of $(p, q) \in \{(1,0),(0,1),(2,0),(1,1),(0,2),(3,0),(2,1),(1,2),(0,3),(4,0),(3,1),(2,2),(1,3),(0,4)\}$ and values of $ignore\_last \in \{0,3,6\}$.

In figure 6.3 we see the case prediction NCRPS metric over time for predictions produced using the different hyperparameters. We show in table 6.1 the top 8 performing hyperparameters for each of the four metrics $\widehat{NMSE}$ and $\widehat{NCRPS}$, for both case prediction and death prediction.

Annoyingly, no set of hyperparameters performs best in all four metrics. We

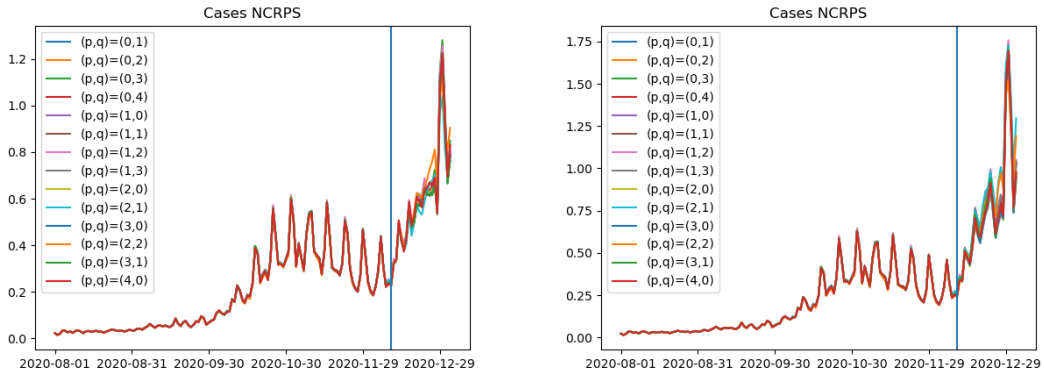

Figure 6.3: NCRPS over time for various hyperparameter combinations. On left is when *ignore_last*= 0, and on right is *ignore_last*= 6

Top Scoring Hyperparameters			
$(p, q, ignore\_last)$	$\widehat{NMSE}$ cases	$(p, q, ignore\_last)$	$\widehat{NCRPS}$ cases
$(2, 1, 0)$	37.2	$(2, 2, 3)$	35.1
$(3, 1, 0)$	39.9	$(2, 2, 6)$	35.5
$(3, 0, 0)$	41.1	$(2, 1, 3)$	35.8
$(1, 0, 0)$	41.5	$(2, 1, 0)$	36.2
$(2, 0, 0)$	41.7	$(2, 2, 0)$	36.3
$(0, 2, 0)$	42.1	$(1, 2, 0)$	36.9
$(1, 3, 0)$	42.1	$(0, 3, 0)$	36.9
$(4, 0, 0)$	43.0	$(0, 1, 0)$	37.0

$(p, q, ignore\_last)$	$\widehat{NMSE}$ deaths	$(p, q, ignore\_last)$	$\widehat{NCRPS}$ deaths
$(2, 1, 0)$	13.6	$(2, 2, 6)$	17.0
$(3, 1, 0)$	13.9	$(2, 2, 3)$	17.3
$(1, 0, 0)$	14.0	$(2, 2, 0)$	17.4
$(2, 0, 0)$	14.1	$(2, 1, 0)$	17.6
$(3, 0, 0)$	14.1	$(0, 3, 0)$	17.6
$(0, 2, 0)$	14.1	$(1, 2, 0)$	17.6
$(1, 3, 0)$	14.1	$(1, 2, 3)$	17.7
$(1, 1, 0)$	14.3	$(1, 0, 3)$	17.7

Table 6.1: Validation of model hyperparameters. Smaller $\widehat{NMSE}$ and $\widehat{NCRPS}$ are better.

therefore choose one that that does very well across each of them. We choose $p = 2$, $q = 1$ and $ignore\_last = 0$.

Out of the 84 hyperparameters combinations tested, this came 1st in the case $\widehat{NMSE}$ metric, 4th in the case $\widehat{NCPRS}$ metric, 1st in the death $\widehat{NMSE}$ metric and 4th in the death $\widehat{NCPRS}$ metric.

Now that we have selected these hyperparameters, we are ready to test our model on the test set alongside the existing models EpiNow2 and Prophet that we described in section 5.2.

6.6 Results

We now train Epi-ARMA with the chosen hyperparameters on both the training set and the validation set. This amounts to using all of the data for the first 155 days from 01/08/2020 until 02/01/2021 to infer the parameters of the model. We then use this model to predict the cases and deaths over the test set, which is from 03/01/2021 until 22/01/2021.

We do the same with the models we previously discussed in section 5.2, which were the semi-mechanistic forecaster EpiNow2 and the pure machine learning forecaster Prophet. Recall however, that we have used EpiNow2 to only predict case numbers and not death numbers, as this was not done using its default settings.

We compare these three models by looking at the 4 metrics already discussed. See figure 6.4 and table 6.2 for the results.

We can see from these graphs and metrics that our ARMA predictor performs much better than the other predictors. Of course this is not too surprising, as we have taken very simple Prophet and EpiNow2 models, and we have not investigated all of their features due to time constraints on this project. With more time, it would be interesting to optimise our usage of these models, so that they are more fairly compared to our new prediction method here. However, we do have a proof of concept that the semi-mechanistic ARMA predictor works.

	$\widehat{NMSE}$ Cases	$\widehat{NCRPS}$ Cases	$\widehat{NMSE}$ Deaths	$\widehat{NCRPS}$ Deaths
Epi-ARMA	14.3	9.07	21.2	13.1
EpiNow2	73.1	26.5	N/A	N/A
Prophet	46.3	17.4	49.2	15.9

Table 6.2: Testing Epi-ARMA against EpiNow2 and Prophet. Smaller $\widehat{NMSE}$ and $\widehat{NCRPS}$ are better.

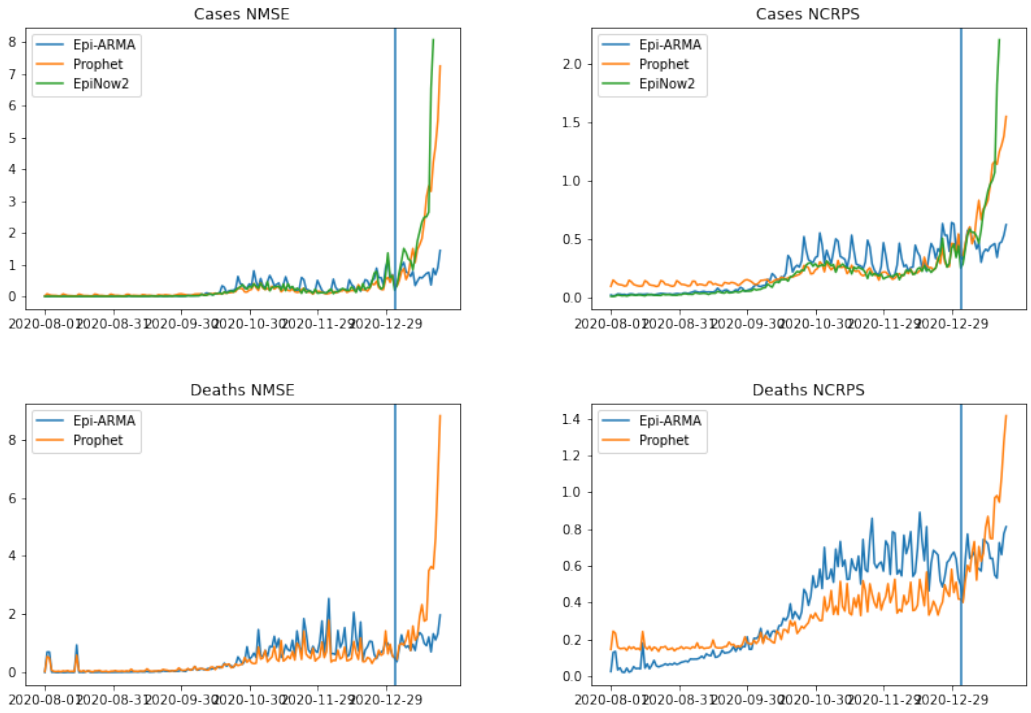

Figure 6.4: NMSE and NCRPS over time for Epi-ARMA, EpiNow2 and Prophet. Smaller $NMSE$ and $NCRPS$ are better

6.7 Summary

To summarise, we took the model produced by Sharma et al and we made some adaptations to it. Firstly we converted the random walk into an ARMA(p,q) series where the parameters were inferred by the model. This allowed us to try to find a time series that represented the noise reasonably well.

For each sample from the posterior of the model, we truncated $R^{(r)}_{noise,t}$ by *ignore_last* days and then continued the ARMA series from this point into the future. This, by looking across all samples, gave a distribution of what $R^{(r)}_{noise,t}$ looks like in the future.

We also assume that $R^{(r)}_{\text{basic}}$ doesn't change in the future, and that $R^{(r)}_{cms,t} = \exp(-\gamma \cdot \mathbf{x}^{(r)}_t)$ only depends on the active countermeasures $\mathbf{x}^{(r)}_t$. Hence $R^{(r)}_{\text{basic}}$ and (if future values of $\mathbf{x}^{(r)}_t$ are inputted) $R^{(r)}_{cms,t}$ are known in the future.

These three terms together give us a distribution for the future values of $R^{(r)}_t$. This in turn is used to give us a distribution over future cases and deaths.

We tuned the hyperparameters of Epi-ARMA, and found that $p = 2$, $q = 1$ and *ignore_last* $= 0$ gave us good results on the validation set, and so we used this on the test set. We then found that the resulting model performed very well

on the test set, doing much better the EpiNow2 and Prophet, albeit much less time was spent on finetuning these other models.

While the ARMA series is a good first attempt at trying to find order in the model's noise term, it is not able to learn complex relationships. Our next idea is to try to introduce deep learning to our model so that more complex patterns can be found in the data. This should hopefully allow it to predict the future trajectory of R_t better than we have already done here.

7

The Latent Neural Network Predictor: Epi-NN

Contents

Deep Learning has the power to find patterns in training data that allow it infer a lot of information. It can do this in such a way that it generalises well to previously unseen data, something an ARMA distribution cannot do as well. Here, we attempt to model the latent variable $R_t^{(r)}$ using a neural network, using case and/or death data as its input, in the hope that we can use this neural net to better predict future values of $R_t^{(r)}$.

We will describe our process here and the ideas that we have tried, so that the reader can get a sense of what worked and what didn't work. We hope that by walking through the process, one is able to apply the ideas used here to other semi-mechanistic model, so that other models can be augmented with deep learning.

Inferring the parameters of a model with a neural net is a harder task than what we have previously looked into, as the parameters don't have a clear representation in the physical world. This means that the posterior of these parameters is generally much more complex. For example, sampling the value of $R_{basic}^{(r)}$ (as done in [13] and in chapter 6) has a physical meaning, which means that it is clear that the posterior of this parameter should be reasonably localised - it will be something like 1.3 ± 0.2. There is no such guarantee for the weights in a neural net. These will not be local, and the posterior will have far more modes than a model without a neural network. This makes the task of inference much harder, as our model's Markov chain will not converge to the posterior quickly.

In this section we explain the structure of our model Epi-NN. In sections 7.1-7.4 we demonstrate how we got it to be able to infer reported cases, reported deaths and $R_t^{(r)}$time series accurately. Only then did we start to consider how well the model generalises on unseen datasets where it is predicting rather than inferring. Section 7.6 outlines how we actually performed prediction.

7.1 The First Attempt

Our model ignores the previous decomposition of $R_t^{(r)}$ that was used in [13] and in chapter 6. This is because we no longer seek to investigate how COVID countermeasures impact the R_t value. Instead we only care about being able to infer the value of $R_t^{(r)}$ from case and/or death data. The only part of the model which was changed was the method to find $R_t^{(r)}$, and so unless otherwise stated the rest of the model remains the same.

Our first try was to build a feedforward neural network (see 2.3 for some background) that calculated $R_t^{(r)}$ as its output, where the input was the previous 3 weeks worth of observed case data. We also gave the user the option of inputting the previous 3 weeks of observed deaths as well. The neural network was shared across all regions to prevent overfitting. It used a softplus activation function in its final layer (ie $\log(1+e^x)$) so that the outputted value of $R_t^{(r)}$ is positive but potentially unbounded above. In all other layers a tanh activation function was used. We began by having 3 hidden layers of dimension 10, 5 and 5, however we eventually set these hyperparameters during the validation process.

Feedforward neural networks work better when their inputs are standardised with mean 0 and standard deviation 1. Hence we standardise our case and death data before inputting it. Noting that epidemic data is generally exponential in nature, we also take a logarithm of the data in the preprocessing step, so that scaling the large case values in the epidemic doesn't collapse all of the small case values to almost zero. The inputs to the neural net were therefore scaled versions of $\log(c_t^{(r)}+1)$ and $\log(d_t^{(r)}+1)$, where the $+1$ is added to prevent problems when $d_t^{(r)}=0$.

This model requires the past 21 days worth of data to find $R_t^{(r)}$ on the 22nd day, and so the first 21 days worth of $R_t^{(r)}$ cannot be found using the neural network, as there is not enough data to input into this. Instead, we set $R_{t\leq 21}^{(r)}$ to be constant for the first 21 days.

As in all Bayesian machine learning, unknown parameters that we would like to infer must have either a prior or a Bayesian hierarchy put on them. In this case the parameters we seek are the (assumed constant in time) value of $R_{t\leq 21}^{(r)}$ for the first 21 days, as well as the weights and biases of the neural net which will give us $R_t^{(r)}$ from 22 days onwards. In our first attempt we simply place a prior on everything of

$$w_i^l, b_j^l \sim \mathcal{N}(0,1)$$

and

$$R_{t\leq 21}^{(r)} \sim \text{Truncated Normal}(1.35, 0.3^2)$$

where the truncation prevents values less than 0.1. This model worked reasonably well to infer case and death data (see figure 7.1).

There are two main problems with it however. Firstly the inferred cases from the model don't look perfectly smooth. We find that a number of the graphs have some small kinks in them, which is something we would rather not have. This is due to the model underfitting the data. Secondly when we look at the graph of $R_t^{(r)}$ we see that it varies very quickly, jumping wildly between very low values

of 0.5 and high values of 2. This is not a realistic model therefore, as we would not expect $R_t^{(r)}$ to vary so dramatically on a daily basis. Our next task was to therefore prevent these from happening.

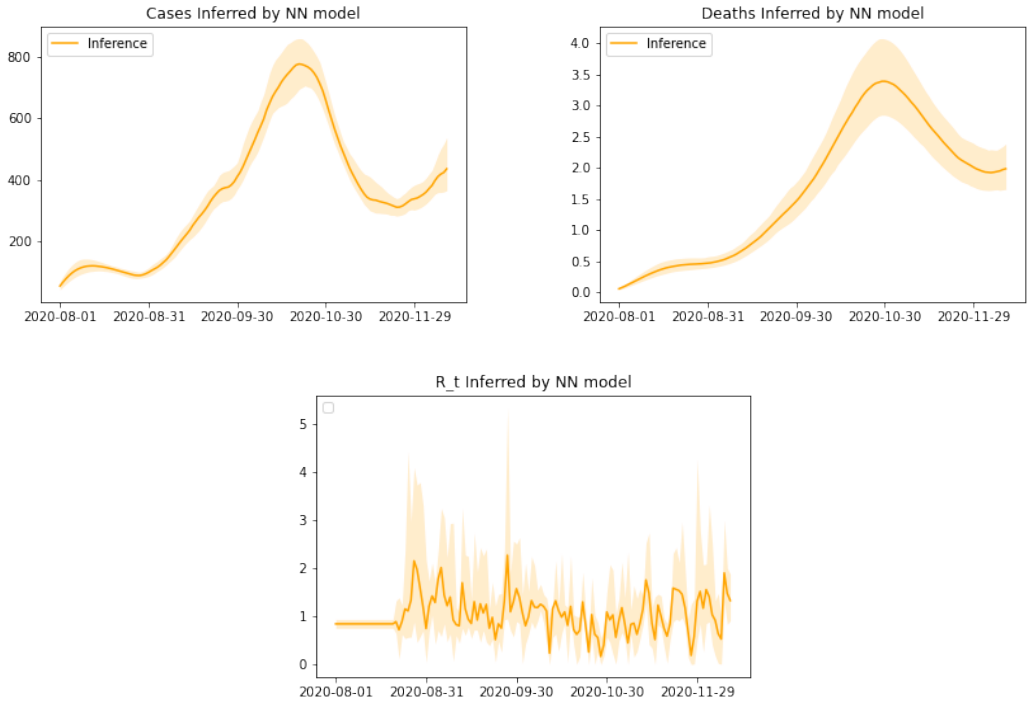

Figure 7.1: The first attempt at Epi-NN, with graphs of inferred cases, deaths and $R_t^{(r)}$for the region of Amsterdam-Amstelland. We can see that the cases are slightly underfit by the model as they are not completely smooth, and the value of $R_t^{(r)}$ is unstable, and varies very quickly.

7.2 Preventing Underfitting

The main source of underfitting in this model is from regularisation on the neural net. When the prior on the neural net's weights forces them to take small values, we find that the network is not able to fit data perfectly due to these constraints. We consider priors of the form $w_i^l, b_j^l \sim \mathcal{N}(0, \sigma^2)$.

Generally speaking, very large σ^2 corresponds to a weakly informed prior on the weights, meaning they are able to fit the data very well as the weights are not constrained much. However, this also makes the neural net less likely to generalise well to unseen data, as it is likely to overfit the training data. Alternatively very small σ^2 force the weights to be very small, meaning that the network can struggle to fit the training data.

We originally set $w_i^l, b_j^l \sim \mathcal{N}(0,1)$, where our choice of $\sigma^2 = 1$ was arbitrary. As this underfit the data, we imagine that $\sigma^2 = 1$ is too small a value, constraining the network too much.

To fix this problem we no longer make σ^2 a user-inputted value, but instead we place a prior on it. In this way we make our neural network into a Bayesian hierarchy, where the model finds a good value of σ^2, which is then used to sample the neural net weights.

Across the lth layer in the neural net, the Bayesian hierarchy for the weights is

$$\sigma_l \sim \text{Half Normal}(1)$$

which ensures $\sigma_l > 0$, and

$$w_i^l, b_j^l \sim \mathcal{N}(0, \sigma_l^2)$$

Note that in our implementation we allow each layer to have a different value of σ_l^2, meaning that different layers have different levels of regularisation.

When this was implemented we see that our underfitting problem is removed (see figure 7.2). These priors ensure that the σ_l are as small as possible (so that the prior probability density is larger), while not underfitting (as then the likelihood is smaller). Hence this allows us to prevent underfitting while also keeping σ_l as small as possible. Keeping σ_l as small as possible is also an important task, as otherwise our model will overfit the training data and not generalise well when we use it on unseen data.

7.3 Stabilising $R_t^{(r)}$

Now that we have dealt with the underfitting problem, we turn our attention to making $R_t^{(r)}$ more stable. We found two solutions to this. We firstly tried making $R_t^{(r)}$ a variable that could only change on weekly basis. However we then did away with this in favour of changing the structure of our neural network. Specifically, we made the output of neural network represent the daily change in $R_t^{(r)}$, rather than the value of $R_t^{(r)}$. We look at these two solutions in turn.

7.3.1 Making $R_t^{(r)}$change Weekly

In the Epi-ARMA model described in chapter 6 we modelled a $R_{noise,t}^{(r)}$ term by letting it be an ARMA time series. We only allowed this time series to change

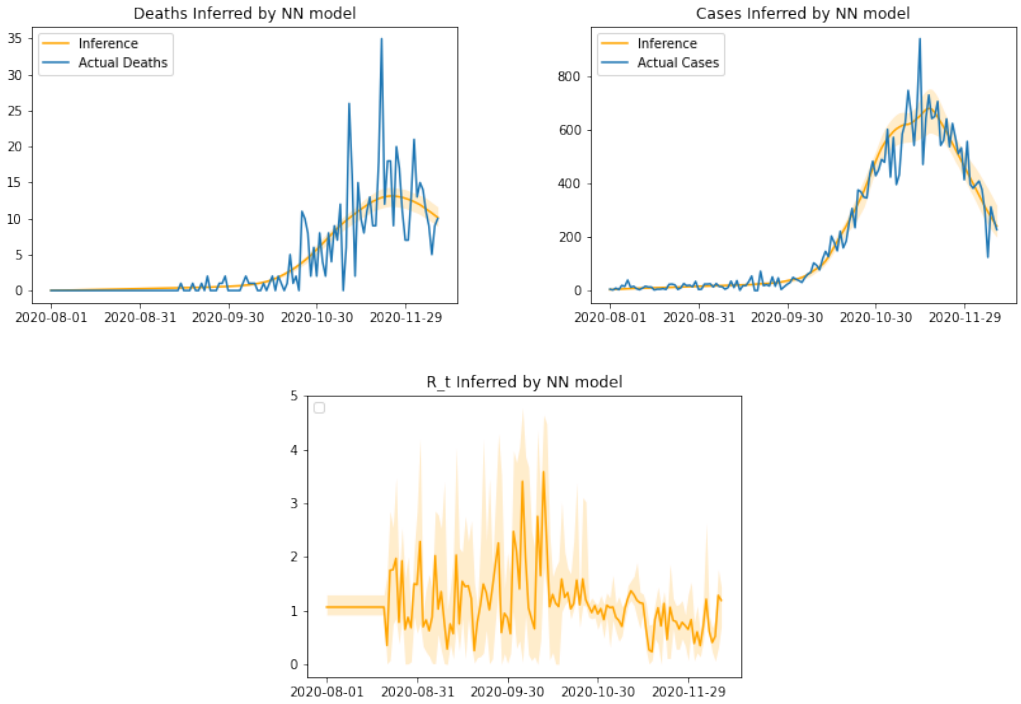

Figure 7.2: Inferred $R_t^{(r)}$, Case and Death data for Abruzzo. The Neural Network weights uses a Bayesian Hierarchy now, and we can see that the data is fit more smoothly.

on a weekly basis, forcing the time series to be constant across each week. We borrow this idea for Epi-NN.

We only calculate the neural network's output every 7 days, and assume that it is constant over the course of the week. We obtained graphs like figure 7.3. This method works very well for inferring the case and death time series at points where data is seen, as can be seen from these figures.

There is a potential problem with this idea however that caused us to abandon it. As the neural net is only being run to calculate $\frac{1}{7}$ of the $R_t^{(r)}$values, we are actually training the network on far fewer data points. Hence we imagine that the network may struggle to generalise. We therefore expect that while this model is good at performing inference, it is likely not going to be good at prediction, as it does not make use of all the data available to it. Due to this, we did not continue with this idea.

7.3.2 Neural Net Models the Change in $R_t^{(r)}$

Our prefered method to stabilise the value of $R_t^{(r)}$was to change the meaning of the output of the neural network. Previously this output was used to be the daily

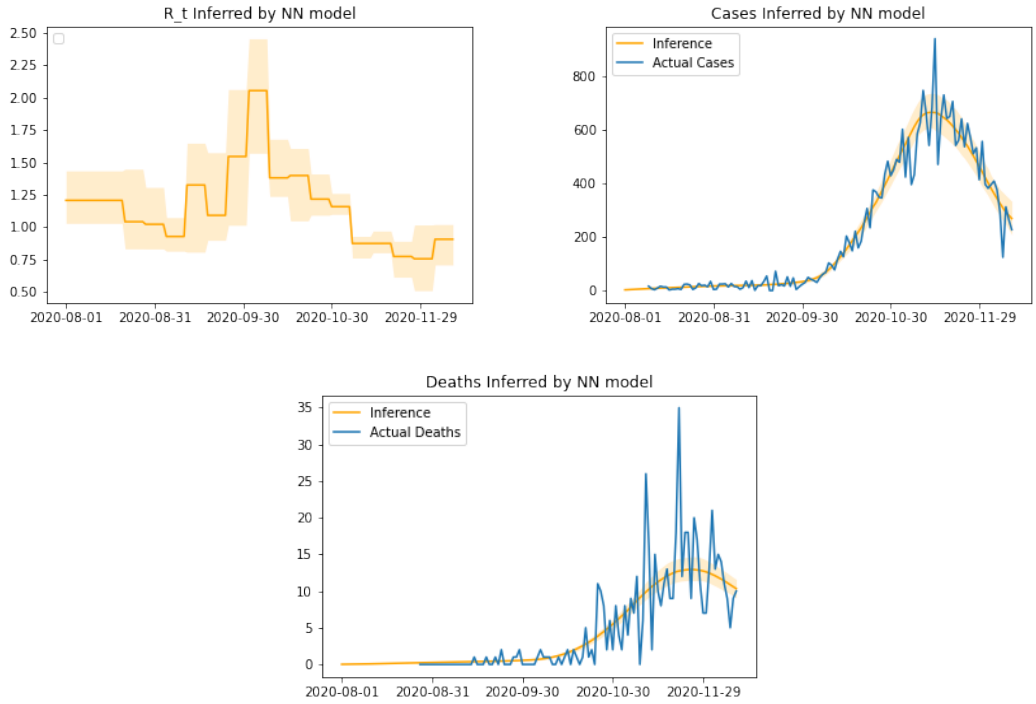

Figure 7.3: The Neural Net is only used to calculate $R_t^{(r)}$on a weekly basis, meaning $R_t^{(r)}$is constant each week. Graphs are of inferred cases, deaths and $R_t^{(r)}$value, for the region of Abruzzo. Shaded regions show 90% confidence intervals.

value of $R_t^{(r)}$. We changed this to make the output the daily *change* in the value of $R_t^{(r)}$. To do this we changed the final activation function in our neural net from a softplus ($\log(1+e^x)$) to a tanh, multiplied by a parameter m that represents the maximum value that $R_t^{(r)}$can change by in a day. As the tanh function has a range between -1 and 1, this means that the maximum daily change in $R_t^{(r)}$ that is allowed in our neural net is a value of m. We would like m to be reasonably small to prevent the value of $R_t^{(r)}$from varying wildly on a daily timescale. Instead of setting the value of m ourselves, we use a weak prior of $m \sim$ Half Normal(0.2) and infer this parameter using our model.

As before, we still set the value of $R_{t\leq 21}^{(r)}$ to be constant in time, sampled from a prior of Truncated Normal$(1.35, 0.3^2)$ where truncation prevents values less than 0.1. This is because we cannot use the neural net at these times due to a lack of input data for the network.

We note here that this neural network could potentially produce values of $R_t^{(r)}$which are negative, even though this would physically correspond to nonsense. To deal with this we simply softplus the value of $R_t^{(r)}$suggested by the neural network, forcing it to be positive.

See figure 7.4 to see the results of changing our neural network in these ways. The model still is able to infer the case and death data well, while $R_t^{(r)}$is now much more stable.

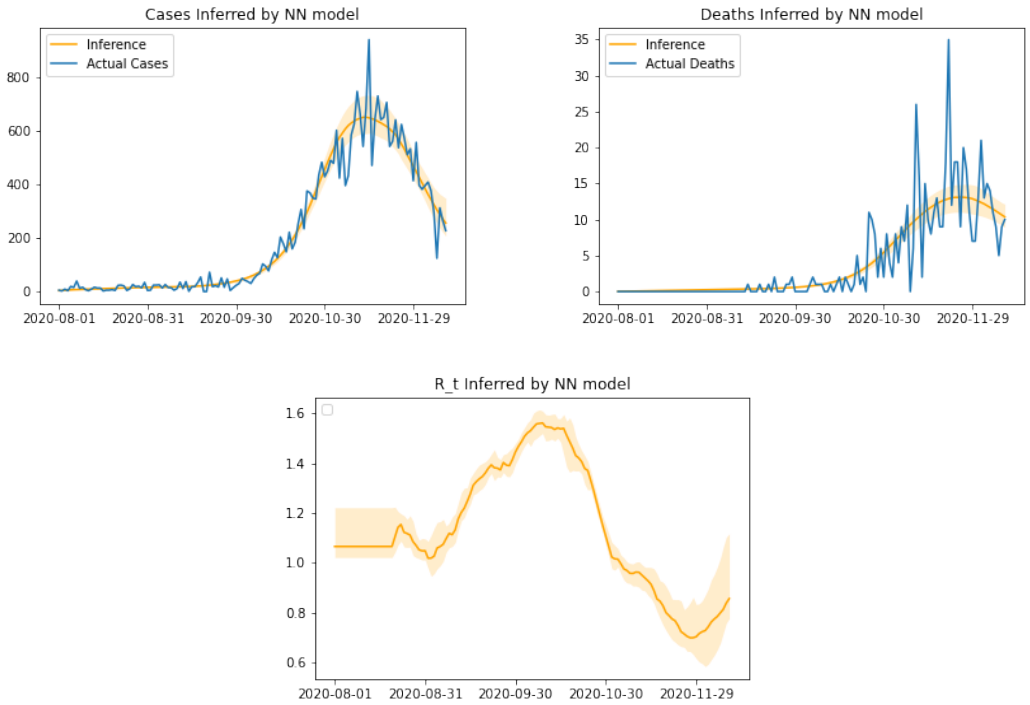

Figure 7.4: The Neural Net is modified to output the daily change to $R_t^{(r)}$. Graphs are of inferred cases, deaths and $R_t^{(r)}$, for Abruzzo, where shaded regions represent 90% confidence intervals.

We note that this method allows the neural network to see all of the training data, and so we prefer this to the previous method of making $R_t^{(r)}$ a weekly process.

7.4 Introducing Reporting Periodicity

We would like Epi-NN to not only tell us what the actual cases and deaths are likely to look like, but also what we expect the reported cases and deaths to look like. This will help us when we come to actually predict the pandemic's future, which we explain in section 7.6.

Reported COVID case data has a large weekly periodicity to it. On weekends, cases typically were under reported. This may be because fewer people would get tested on weekends when testing centres had lower capacity. Whatever the reason, it is clear from the data that there is a delay between a case becoming symptomatic and it actually being reported as a case. See figure 7.5 to see the

periodic weekly cycle present in the case data. We note here that this periodicity is less noticeable in death data.

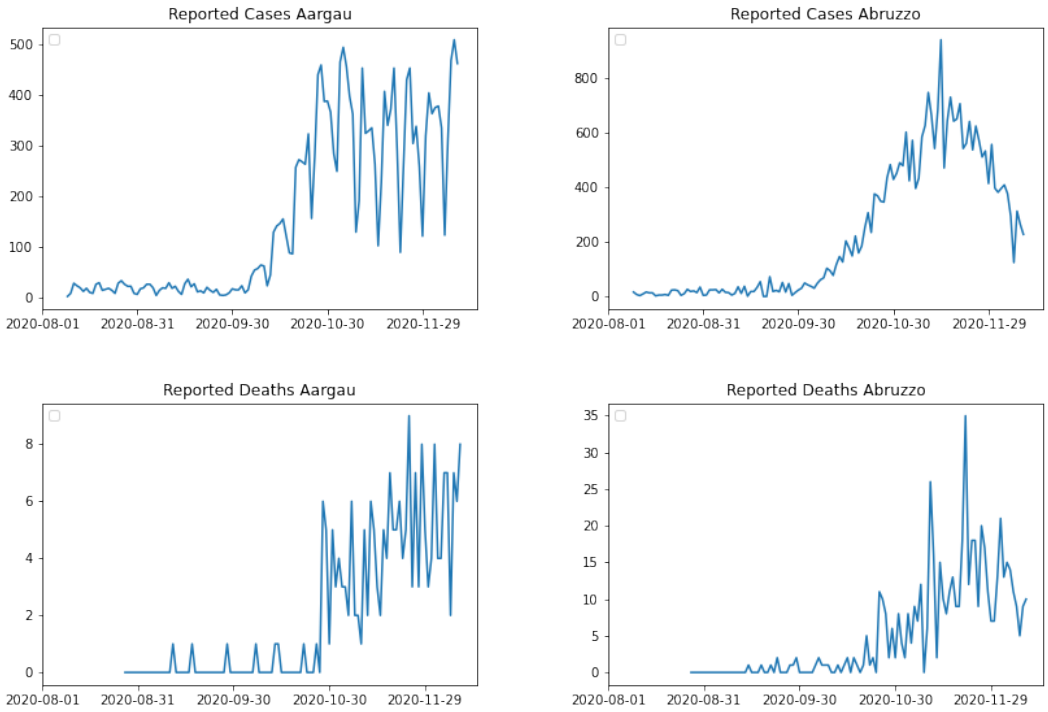

Figure 7.5: Case data (top) and Death data (bottom) for Aargau (left) and Abruzzo (right). We aim to replicate the weekly periodicity seen in the data.

So far our model has not attempted to replicate this reporting periodicity at all. As can be seen from all of our plots up until now, the inferred case and death data has been smooth, as it represents the number of cases and deaths that the model actually thinks were present at those times. We now introduce a way to allow the model to output an inferred value of the reported cases and deaths, so that it includes the periodic terms.

We model this periodic term reasonably simplistically. On each day we multiply the case and death data by a parameter $\gamma_{day}^{(r)}$ that varies over the region and over the days of the week.

For 6 of the 7 days in the week we place a prior on this parameter of

$$\gamma_{day}^{(r)} \sim \text{Log Normal}(0, 1)$$

We chose this prior as this ensures that the multiplier is equally likely to increase or decrease by a given factor. We then set the final day of the week so that the geometric mean of the $\gamma_{day}^{(r)}$ is 1. This ensures that in a week, the average

reported cases is similar to the average actual cases. This very simple method works reasonably successfully, as can be seen from figure 7.6.

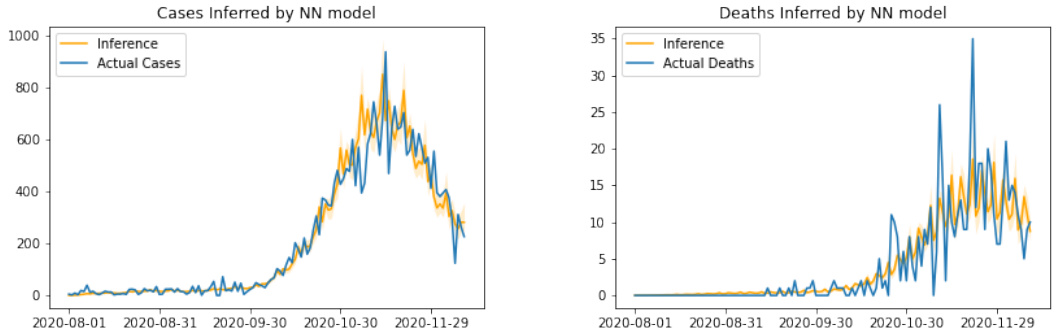

Figure 7.6: Inferring the reported cases and deaths in Abruzzo, where a weekly periodic term has been added to the neural net.

7.5 Summarising Input Data

Another step we took was to investigate how we could initially process the input data for the neural network. Currently we are inputting 3 weeks worth of the logarithm of case and/or death data. The ideas that follow attempt to try to prevent overfitting the neural network, as inputting large amounts of data to the neural net can allow it to overfit to noise.

We had two ideas here.

1. We simply take a 7 day rolling average of the data. This should remove the large variation seen in the inputted data, as was seen in figure 7.5. This would hopefully remove a large amount of noise.

2. We input a summary of the past 21 days of data, rather than the data itself. This summary includes the standard deviation, the overall slope, and then a number of percentiles. We set the number of dimensions in this summary to be a hyperparameter of the model. In particular, we will always include the standard deviation and the slope, and the remaining dimensions are all equally spaced percentiles. This reduces the number of inputs to the neural network.

We could also do these two ideas in conjunction. We firstly take a rolling average, and then we summarise the past 21 days of rolling averages. We do not

plot the results from these here, but we note that both methods still were able to fit the data well. We will decide which preprocessing steps to do when we test our model on the validation set.

7.6 Autoregressive Prediction

So far we have explained how Epi-NN converts reported case data into inferences for $R_t^{(r)}$at the times at which data is observed. To do this we input reported case and/or death data into the neural net, allowing us to output $R_t^{(r)}$. We have managed to ensure that our model infers the reported case and death data reasonably well. We have not explained how we use this latent network to predict future values of $R_t^{(r)}$when we have no reported case data to input into it however. We show here how this is done using an autoregressive framework once Epi-NN has been run and the neural network has been made.

Given 21 days worth of reported cases and/or deaths, our neural network attempts to predict the value of $R_t^{(r)}$on the 22nd day. Assuming that our neural network is accurate, we use our value of $R_t^{(r)}$with the previous infection's inferences to predict the number of infections on the 22nd day. This in turn can be used to predict the actual number of cases and deaths on the 22nd day, as well as the reported number of cases and deaths (by multiplying the actual number of cases by the $\gamma_{day}^{(r)}$).

We now shift the input of our neural network forwards one day, so that the reported case and death data we just predicted are part of the network's new input. We can repeat the process described above indefinitely. Doing this allows us to predict future time series for cases, deaths, infections and $R_t^{(r)}$.

7.7 Improving Convergence in a Neural Network

As we have already mentioned, the posterior distribution when a neural network is involved is very complicated. We expect the posterior to have a very large number of modes, which can make the task of sampling it with NUTS more difficult.

The reason for having so many modes comes from a symmetry in the neural network. Specifically, a neural net is invariant to permutations of the neurons in a given layer, so long as the weights and biases associated with that neuron are permuted in the same way. This means that a hidden layer of size d can be permuted in $d!$ ways, without the output of the model changing. If there are L of these layers, then there are $(d!)^L$ ways to set the weights and biases of a neural net

that produce functionally equivalent networks. Note this is generally a huge number, for example having three hidden layers of size 10 gives $\approx 5 \times 10^{19}$ functionally equivalent networks. The consequence of this is that if a given parameterisation of the network corresponds to a mode in the posterior, then any parameterisation that creates any of the $(d!)^L$ functionally equivalent networks is also a mode of the posterior. This is certainly not ideal, and it means that NUTS will struggle to sample from this distribution, making the sampling process take a long time.

To deal with this problem we will parameterise our model in such a way that only 1 of these $(d!)^L$ functionally equivalent neural networks can be built. This will mean that our posterior distribution will have far fewer modes. In particular we constrain our model to only allow neural networks in which the biases of each layer are ordered from smallest to largest. Our method is as follows.

Instead of sampling the biases themselves, we sample the difference between the biases. We use a prior of

$$b_i^l - b_{i-1}^l \sim \text{Half Normal}(\tau_l^2)$$

where we use a Bayesian hierarchy with $\tau_l \sim \text{Half Normal}(0.5^2)$ shared across a layer. This ensures that we have our all-important ordering of $b_0^l < b_1^l < \dots$. However we have not yet obtained the value of $b_0^l = 0$, and so we do not have the actual values of the biases yet. To find $b_0^l = 0$ we do one last sampling, which is

$$b_{mean}^l \sim \mathcal{N}(0, \lambda^2)$$

where we use a Bayesian hierarchy with $\lambda \sim \text{Half Normal}(0.3)$. We then set b_0^l so that the mean of the biases in the lth layer are b_{mean}^l.

By doing this, we have parameterised our neural network in such a way that only 1 of the $(d!)^L$ functionally equivalent neural networks is buildable.

While we did not rigorously test the change in speed in sampling after this idea was introduced, we note that it lowered the training time of 40 warm-up steps from around 6 hours to 4 hours. Due to this speed up, we used this new parameterisation for our model. We reiterate however that this testing was not rigorous due to time constraints, and it is possible that this speed up is actually just within normal variation in speed between chains.

7.8 Tuning the Hyperparameters

We have many hyperparameters to set here, which we explain in table 7.1.

Hyperparameter	Tested Values	Explanation
Number of Previous Days to Input	14, 21, 28	The number of past days of data that are used in the neural network to infer the value of R
Hidden Layer Architecture	[10, 10, 5, 5], [10, 10, 5], [20, 10]	The dimensions of the neural networks hidden layers, where the length of the list is the number of hidden layers.
Input Death Data	True, False	Whether we are inputting data from the death time series, as well as the case time series. Note this doubles the size of the neural net input.
Use Rolling Average	True, False	Whether the inputted data is converted into a rolling average before the neural net runs
Use Summarised Data	True, False	Whether the inputted data is summarised into various percentiles, standard deviation and slope.
Summarised Data Dimensions	7, 11, 15	Only used if we are using summarised data, and it represents the number of dimensions that inputted case or death data is summarised to. Two of these dimensions are slope and standard deviation, and the rest are (linearly spaced) percentiles.

Table 7.1: Hyperparameters to tune for Epi-NN.

We would like to train various models with different hyperparameter combinations on the training data, and then we would like to compare these on the validation set. Ideally, we would train and test all 144 different models, however this would take us a long time, as each model takes roughly 2 days to run 100 warmup steps with 50 samples. Instead we test a random sample of the models we would like to test, and we will choose the best performing of this random sample. We considered 14 models for validation purposes.

Unfortunately, running a model for 100 warmup steps with 50 samples does not

allow the model to reach convergence. We notice this as the inferred deaths do not perfectly align with the reported deaths when we consider their plots. We are limited by time and computational power, and so we will have to make do with 100 warmup steps when we run our models for hyperparameter validation. This means that we are comparing a number of our models when they have not converged, and so at this point our case and death $\widehat{NMSE}$ and $\widehat{NCRPS}$ will likely be larger than had we reached convergence. Hence, our hyperparameter tuning will presumably be much less useful than if we had more time to allow each model to converge, but nonetheless we continue by choosing the model which performs best on this validation set.

The top performing models are presented in table 7.2, where we show which of our models did best in each of the four metrics. We note that of the 14 models tested, there was one that did best on both the case $\widehat{NMSE}$ and the case $\widehat{NCRPS}$. It came 3rd on death $\widehat{NCRPS}$ and 5th on death $\widehat{NMSE}$. This model had settings of:

Number of Previous Days Inputted	28
Hidden Layer Architecture	[10,10,5,5]
Input Death Data	True
Use Rolling Average	False
Use Summarised Data	False
Summarised Data Dimensions	N/A

We use these settings when we test Epi-NN on our test set.

Hyperparameters that have Smallest Cases $\widehat{NMSE}$

Days Inputted	Hidden Layers	Input Deaths	Rolling Average	Summarise	Summary Dims	$\widehat{NMSE}$
28	[10,10,5,5]	True	False	False	N/A	65.0
21	[10,10,5,5]	True	True	True	15	66.6
28	[20,10]	False	False	True	11	69.1

Hyperparameters that have Smallest Cases $\widehat{NCRPS}$

Days Inputted	Hidden Layers	Input Deaths	Rolling Average	Summarise	Summary Dims	$\widehat{NCRPS}$
28	[10,10,5,5]	True	False	False	N/A	16.9
21	[20,10]	True	False	True	7	20.2
14	[10,10,5,5]	False	False	False	N/A	20.2

Hyperparameters that have Smallest Deaths $\widehat{NMSE}$

Days Inputted	Hidden Layers	Input Deaths	Rolling Average	Summarise	Summary Dims	$\widehat{NMSE}$
21	[20,10]	True	False	True	7	28.8
14	[10,10,5,5]	False	False	False	N/A	28.8
14	[20,10]	True	False	True	7	32.0

Hyperparameters that have Smallest Deaths $\widehat{NCRPS}$

Days Inputted	Hidden Layers	Input Deaths	Rolling Average	Summarise	Summary Dims	$\widehat{NCRPS}$
14	[20,10]	True	False	True	7	16.3
28	[20,10]	False	False	True	11	17.2
28	[10,10,5,5]	True	False	False	N/A	17.4

Table 7.2: Validation of model hyperparameters. Smaller $\widehat{NMSE}$ and $\widehat{NCRPS}$ are better.

7.9 Results

Now that we have tuned our hyperparameters for Epi-NN, we are ready to test it against the two previously produced models (Prophet and EpiNow2), as well as against Epi-ARMA which we produced in chapter 6. We train all four models using the first 155 days from 01/08/2020 until 02/01/2021. We then use these model to predict the cases and deaths over the test set, which is from 03/01/2021 until 22/01/2021.

We compare these models by looking at the 4 metrics already discussed. See figure 7.7 and table 7.3 for the results. We find that Epi-NN, like Epi-ARMA, beats the benchmarks of Prophet and EpiNow2 comfortably. We also see that in the case $\widehat{NMSE}$ and case $\widehat{NCRPS}$ Epi-NN beats Epi-ARMA. This was not true for death $\widehat{NMSE}$ and death $\widehat{NCRPS}$.

We demonstrate therefore that the installation of a neural net is potentially able to increase the predictive power of the semi-mechanistic model, as was seen in the case data metrics. However it did not make improvements across the board, as the augmentation was not enough for the model to outperform Epi-ARMA in the death metrics.

We make an important note on convergence here. The model that we used to predict the data on the test set was run for 200 warmup steps and then sampled 250 times on each of 4 chains. This, like at the validation stage, was not enough to reach convergence - the $\widehat{R}$ of our model was only 1.3, and it is recommended to only use a model if the samples had $\widehat{R}$<1.05 [14]. This took 6 days to run in total however, and so due to time constraints we were not able to run this for longer. What this means is that if the model had been able to converge, say if it was run for more warmup steps and more samples were drawn, then the results may have improved slightly. Perhaps with convergence, Epi-NN would have beaten Epi-ARMA in the death $\widehat{NMSE}$ and $\widehat{NCRPS}$ metrics. However, it would take a long time to reach convergence. In future, it would be interesting to consider how these results would change if Epi-NN was run for more time.

While Epi-NN is currently slow to run, we believe that it demonstrates that neural networks can potentially be used to increase the predictive performance of a semi-mechanistic model. Now that we have covered the predictive abilities of Epi-NN, we briefly consider how it can be used for inference purposes. In particular, we have already seen how it can infer the value of $R_t^{(r)}$over time, but now we would like to see its ability to infer multiple time dependent latent parameters.

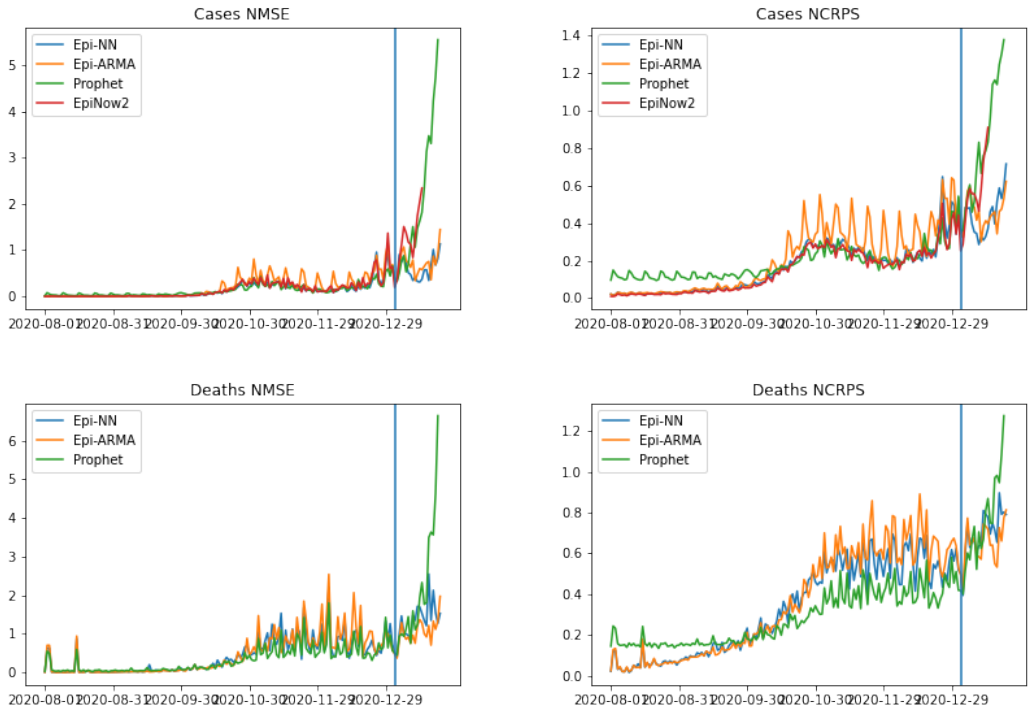

Figure 7.7: NMSE and NCRPS over time for Epi-NN, Epi-ARMA, EpiNow2 and Prophet. Top row shows cases, bottom shows deaths. Left shows $NMSE$, right shows $NCRPS$. Smaller $NMSE$ and $NCRPS$ are better

	$\widehat{NMSE}$ Cases	$\widehat{NCRPS}$ Cases	$\widehat{NMSE}$ Deaths	$\widehat{NCRPS}$ Deaths
Epi-NN	12.1	8.8	27.4	14.1
Epi-ARMA	14.3	9.1	21.2	13.1
EpiNow2	73.1	26.5	N/A	N/A
Prophet	46.3	17.4	49.2	15.9

Table 7.3: Testing the Epi-NN against Epi-ARMA, EpiNow2 and Prophet. Smaller $\widehat{NMSE}$ and $\widehat{NCRPS}$ are better.

7.10 Inferring Multiple Time Dependent Parameters

Up until this point we have only inferred a single time dependent parameter of the system, namely $R_t^{(r)}$. We would now like to try introducing a second time dependent parameter, such as the *case fatality rate*, $\text{cfr}_t^{(r)}$.

So far we have assumed that the *infection ascertainment rate* $\alpha_t^{(r)} = \alpha^{(r)}$ and the *infection fatality rate* $\beta_t^{(r)} = \beta^{(r)}$ are constant in time. This has the consequence, as explained in 3.2.3, of us being able to rescale the value of the infections $i_t^{(r)}$ by

a factor of $\alpha^{(r)}$, which is equivalent to setting $\alpha^{(r)} = 1$ in the renewal equations. This system is identical to one in which all cases are ascertained, and $\beta^{(r)}$ also represents the (time-constant) case fatality rate $\text{cfr}^{(r)}$, as we have

$$\text{cfr}^{(r)} = \frac{\beta^{(r)}}{\alpha^{(r)}} = \beta^{(r)}$$

This assumption that $\text{cfr}^{(r)}$ is constant in time may not be a reasonable one. In particular, we imagine that $\alpha_t^{(r)}$ might increase as COVID testing improved, and $\beta_t^{(r)}$ might change as various COVID variants become more prominent.

The process we have introduced to infer $R_t^{(r)}$could as easily be used to infer the value of a time dependent $\text{cfr}_t^{(r)}$. Instead of sampling a regional cfr directly, we now sample a region independent neural network that takes regional case and death data and outputs the daily change in a regional cfr. A softplus is then used on the calculated cfr to enforce positivity. This neural net is independent from the neural network that infers the value of $R_t^{(r)}$.

We ran a number of experiments in which both $R_t^{(r)}$and $\text{cfr}_t^{(r)}$ were time dependent, and we plot the results in 7.8. We found that the model was able to infer the values of $R_t^{(r)}$and $\text{cfr}_t^{(r)}$ over time. This is because we are using two observed time series to infer two time dependent parameters. We note that we would not have enough types of data to be able to infer all three of $R_t^{(r)}$, $\alpha_t^{(r)}$ and $\beta_t^{(r)}$. For this, we would also need to observe a third type of data, specifically infection prevalence data.

To see whether there is enough data to infer these parameters it is worth checking the correlation between the samples of the different latent parameters. If two parameters have almost perfect correlation then we expect that our model did not have enough information to infer their values properly, as it cannot disentangle them. This would suggest we are trying to infer too much from the data available to us. We did not check the correlation between $R_t^{(r)}$and $\text{cfr}_t^{(r)}$ explicitly, as all we wanted to do here is show that it is possible for our model to infer multiple time dependent parameters. Hence we do not worry about potential issues with this model where we may lack sufficient data to infer $\text{cfr}_t^{(r)}$ rigorously.

Including a second time dependent variable to the model can help improve the ability of the model to infer the latent parameters of the system, as we are now able to make fewer assumptions about them. However, at the time of writing we have not tested whether this addition improves the prediction power of Epi-NN. We instead note that this is proof of concept that we can install multiple neural networks using the framework produced, and obtain reasonable inferences from this. We now generalise the work done in producing Epi-NN to explain a possible

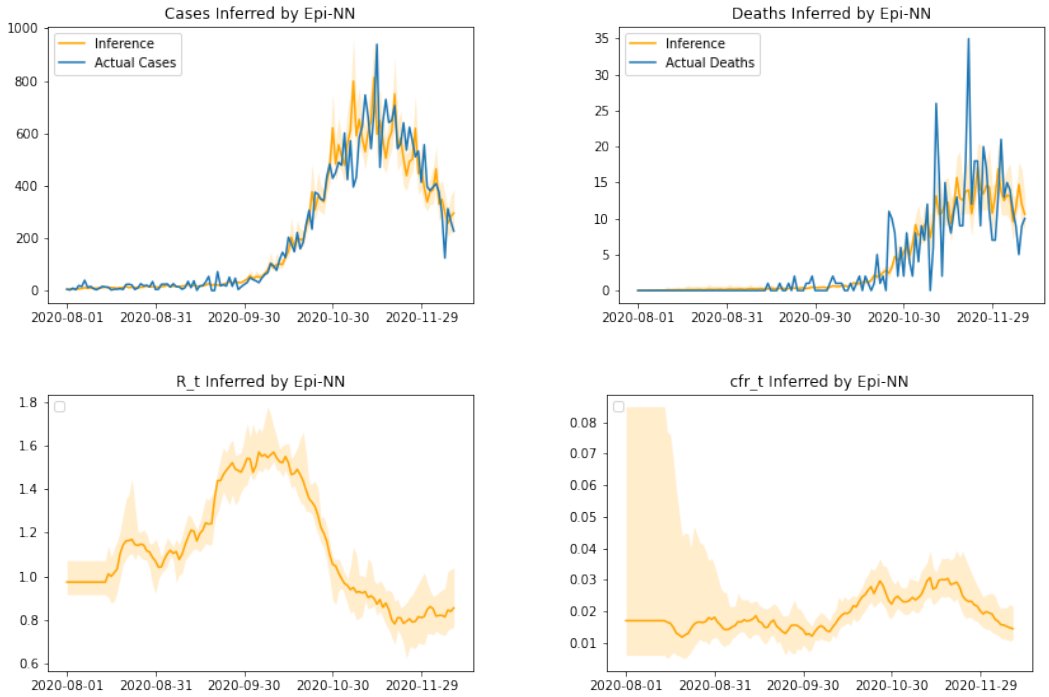

Figure 7.8: Epi-NN also infers the cfr here. Posterior is for the region of Abruzzo, and shaded regions correspond to a 90% confidence interval. Top left: Cases. Top right: Deaths. Bottom left: R. Bottom right: cfr

framework for improving inference and prediction in semi-mechanistic models.

7.11 Lessons Learned for other Semi-Mechanistic Models

We end this section with a discussion on how our process can be extended to other semi-mechanistic models. Epi-NN took a probabilistic program for the renewal equation, and made its key parameter $R_t^{(r)}$be the output of a neural network. We also showed that we could make the cfr rate be time dependent in an identical way to $R_t^{(r)}$. In particular, the neural network infers the change in these time dependent parameters one day ahead, using past data. The aim was to use the data to train the neural net, so that we could autoregressively use this to calculate the trajectory of the pandemic. Our framework is not specific to Epi-NN, and so we would like to generalise it.

The work in this project suggests the following framework for the augmentation of semi-mechanistic models with deep learning:

1. Identify all time dependent parameters of the model. In this case these are $R_t^{(r)}$, $\alpha_t^{(r)}$ and $\beta_t^{(r)}$, which gives 3 per region.

2. Identify how many types of time series are observed in the model that it will be conditioned on. In this case there are 2 per region, representing the case data and the death data.

3. Take, at most, this number of the time dependent parameters that we will model as time dependent. These are now denoted by θ_t. The rest of the (potentially) time dependent parameters must be fixed - otherwise there is not enough data to inform the model. In our case we fix $\alpha_t^{(r)} = 1$, and consider time dependent $R_t^{(r)}$and $\beta_t^{(r)} = \text{cfr}_t^{(r)}$. If too many time dependent parameters are considered, then the posterior has a 'bad' geometry as it introduces extra degrees of freedom in the model that the model cannot fit properly.

4. Build neural networks that find θ_t by inputting raw data from times before t. In particular we use our trick from section 7.7 to build the network in such a way that we drastically reduce the number of modes in the posterior.

5. If we expect θ_t to be reasonably smooth in time, we set the neural network to output how these parameters change in one time step. We use an activation function of $\tanh(z)$ multiplied by a maximum change parameter m, to limit how much we allow θ_t to change in one time step. We learn the value of m using our model.

6. If there are any restrictions on these parameters (eg they must be positive) then we explicitly enforce that in the model (eg softplus the outputted parameters).

7. If our aim is to predict the future, then we use these parameters to predict what the raw data would look like in the future, allowing us to autoregressively perform prediction.

8. After the model has been run, it should be retroactively checked that there is not a large correlation between the different time dependent parameters in θ_t, as this would suggest that there was not enough data for the model to disentangle them. If the correlation is deemed too large, reduce the number of inferred time dependent parameters (eg by making one of them constant).

The hope is that this framework would allow one to infer time dependent latent parameters (like $R_t^{(r)}$and cfr_t) in a general and flexible way in other Bayesian semi-

mechanistic models. We also hope that it would allow prediction to be performed in a way that makes fewer assumptions about the form of many of the parameters.

We also note that the uncertainty in the predictions made by this framework is quite different in character to the uncertainty in the predictions made by Epi-ARMA. Once the neural net is sampled, the model makes predictions about the future value of the latent parameters. The uncertainty in this comes from the uncertainty in the neural net weights and biases. This is completely different to how Epi-ARMA worked. To find future values of latent parameters, Epi-ARMA sampled future noise ϵ_t, which injects a lot of noise into the model. This was the source of most of its uncertainty.

7.12 Future Work

Sampling a latent neural network is a slow process due to the complexity of the model's posterior. Further, the computational resources available to the author over the course of this project were relatively weak. Due to these reasons, testing each iteration of the model took a number of days, and so there were some ideas for this project that were not able to be considered due to time constraints. We mention these here as potential future work.

7.12.1 The Type of Neural Network

In our work we used a feedforward neural network. We chose this as it is fast to build. However there is a better choice for this type of work. As we are using time series data as the input to our neural network, a recurrent neural network (RNN) would presumably be better, as it is better suited to inputs with temporal order. We did not do this due to time constraints. It would be interesting to see how an RNN would perform.

7.12.2 Sensitivity Analysis

Epi-NN used a number of priors that are user-defined. An important task for our model is to verify that the choice of these priors has little effect on the models outputs. This takes a long time, and so there was not enough time over this project to do this.

7.12.3 Investigate the New Sampling Method

In section 7.7 we considered a new way to sample the biases and weights of the neural network which enforces an order on the neurons. This reduces the number of modes in the posterior by a huge number. It should be investigated whether this actually gave speed up, or whether the apparent speed up was within natural variation. If this gave speed up, this could be a preferred way to sample from Bayesian neural networks where the posterior is very complex.

7.12.4 Test the Framework

We believe this is the most important piece of future work. We have presented in 7.11 a framework with which one can augment a semi-mechanistic model with neural networks. This framework was produced by considering the renewal model in its inference and prediction of latent parameters in the COVID pandemic. A key job in the future would be to verify that this process would work for other semi-mechanistic models, rather than just the one described here. If this works, then we would have produced a simple, general and flexible process with which key time dependent parameters can be modelled in semi-mechanistic models, allowing us to predict the future.

8

Conclusion

In this project we have produced 3 separate models that analyse the second wave of the COVID pandemic. These were all semi-mechanistic Bayesian models based on the renewal process. While the first two do not use deep learning, the third model (Epi-NN) augmented the semi-mechanistic model with a feedforward neural network and used this to both infer time dependent latent parameters, and predict the future. The project was named in light of Epi-NN, and we hope that it shows that neural networks could be powerful in helping Bayesian semi-mechanistic models infer and predict latent parameters.

8.1 Model 1: Interacting Countermeasures

Our first model analysed how COVID interventions interact with one another, something that previous models did not do. We looked at this through the lens of informing health policy. The results suggest that the interventions generally reduce R more when other interventions are already in play. This suggests that we do not get diminishing returns when further restrictions are put in place. Not all interventions behaved in this way however, specifically the closing of educational institutions. We saw that closing schools has a reduced effect when done in tandem with other countermeasures. This may suggest that closing schools may be less useful as a countermeasure than previous work made out. Sharma et al showed that the effect of closing schools was much less than the effect of shutting down

businesses and enforcing curfews, and the work presented here suggests that these effects are lessened further when other interventions are present.

8.2 Model 2: Epi-ARMA

Our second model was called Epi-ARMA, and it aimed to predict future cases and deaths due to COVID. It did this by decomposing R into three parts. The first was the basic value of R, the second was a reduction due to active countermeasures, and the third was a noise term that allows the model to change its value of R on a weekly basis. We set this noise term to follow an ARMA(p,q) series, and we inferred the coefficients of this series. We then continued this into the future to allow us to predict how we expected R to change, and then in turn used this to find the predicted number of cases and deaths in each region. We compared Epi-ARMA to EpiNow2 and Prophet, two methods that could predict COVID rates. While we did not finetune EpiNow2 and Prophet perfectly, we found that Epi-ARMA outperformed both of them across all of our metrics.

8.3 Model 3: Epi-NN

Our third model, Epi-NN, was made by augmenting the semi-mechanistic model with deep learning. This model did not make use of knowledge about which countermeasures were active, and instead inferred the value of R using a neural network with only past cases and deaths as its input. We showed proof of concept for two ideas here.

Firstly, we used Epi-NN to predict future COVID rates. It was able to outperform Prophet, EpiNow2 in all metrics, as well as Epi-ARMA in two of the four metrics. This demonstrates that installing a neural network into the semi-mechanistic model is able to improve its predictive powers compared to modelling the latent parameters as ARMA, but this is not always true. The main issue we saw was that Epi-NN took a long time to produce, due to slow sampling times because of a complex posterior. This meant our final testing model hadn't even reached convergence (even though this took 6 days), and so we expect that if more samples had been taken to improve convergence that our model may have improved in performance. However, we can see despite problems with convergence that Epi-NN has the potential to do well.

Secondly, we demonstrated how the neural network allows the model to infer latent parameters of the system. We showed that multiple parameters (R and the

case fatality rate) were able to be treated as time dependent at the same time. This allows us to remove the assumption that some of the parameters in the model are constant, adding to its flexibility. We also suggest a framework with which other Bayesian semi-mechanistic models can be augmented with a neural network, allowing a number of its time dependent parameters to be inferred. This provides a simple, general and flexible way to keep track of parameters over time.

Epi-NN has shown some promise in being able to outperform Epi-ARMA in predicting future cases even before it finished converging fully. It does however have the significant drawback of being slow, something which makes it (currently) only useful for doing offline work. To be more useful, work must be done to try and improve the speed of convergence.

8.4 Future Work

We have mentioned at the end of chapters 4 and 7 some future work that follows on from this project. We reiterate these here.

1. Reaching convergence was very slow in Epi-NN due to the complex posterior when neural networks are introduced. It would be worth investigating ways to reduce the posterior complexity to hopefully speed up convergence.

2. Test the general framework (section 7.11) for augmenting Bayesian semi-mechanistic models with neural networks on other processes. This would allow us to verify that our framework is sufficiently general.

3. Investigate the new method for sampling neural networks (section 7.7). This method enforces an order on the neurons, meaning that not all of the functionally equivalent networks are allowed using our sampling method. While this method appeared to give us some speed up, we did not rigorously test this. It would be interesting to see if this speed up was significant.

4. Epi-NN should be changed to use an RNN rather than a feedforward NN. This is better suited to time series data.

5. Robustness should be tested in all three of the models. In all three cases, arbitrary parameters and distributions were chosen to help define priors. We should verify that our choice of priors does not hugely influence the results of the model.

The first three ideas not only would benefit Epi-NN, but also other semi-mechanistic models, and hence we believe these are more important than the final two. Our hope from this project is that neural networks can be seen to be useful for inference and prediction in Bayesian semi-mechanistic models, something that they are not generally used for currently.

Bibliography

[1] P. Brémaud. *Markov Chains*. 1999. DOI: 10.1007/978-3-030-45982-6_1.

[2] R.M Neal. *MCMC using Hamiltonian dynamics*. 2010. DOI: 10.48550/arXiv.1206.1901.

[3] M.D Hoffman and A. Gelman. "The No-U-turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo". In: *Journal of Machine Learning Research* 15 (2014). DOI: 10.48550/arXiv.1111.4246.

[4] B. Letham S.J. Taylor. "Forecasting at scale". In: *PeerJ Preprints* (2017). DOI: 10.7287/peerj.preprints.3190v2.

[5] D. Champredon, J. Dushoff, and D.J.D. Earn. "Equivalence of the Erlang-distributed SEIR epidemic model and the renewal equation". In: *SIAM Journal on Applied Mathematics* 78 (2018). DOI: 10.1137/18M1186411.

[6] S. Abbott et al. "EpiNow2: Estimate Real-Time Case Counts and Time-Varying Epidemiological Parameters". In: (2020). DOI: 10.5281/zenodo.3957489.

[7] José M. Carcione et al. "A Simulation of a COVID-19 Epidemic Based on a Deterministic SEIR Model". In: *Frontiers in Public Health* 8 (2020). DOI: 10.3389/fpubh.2020.00230.

[8] S. Bhatt et al. "Semi-Mechanistic Bayesian modeling of COVID-19 with Renewal Processes". In: (2020). DOI: 10.48550/arXiv.2012.00394.

[9] S. Flaxman et al. "Estimating the effects of non-pharmaceutical interventions on COVID-19 in Europe". In: *Nature* 584 (2020). DOI: 10.1038/s41586-020-2405-7.

[10] Z. Yang et al. "Modified SEIR and AI prediction of the epidemics trend of COVID-19 in China under public health interventions". In: *Journal of Thoracic Disease* 12.3 (2020). DOI: 10.21037/jtd.2020.02.64.

[11] R. Challen et al. "Meta-analysis of the SARS-CoV-2 serial interval and the impact of parameter uncertainty on the COVID-19 reproduction number". In: *medRxiv* (2020). DOI: 10.1101/2020.11.17.20231548.

[12] M. Sharma et al. *GitHub for 'Understanding the effectiveness of government interventions in Europe's second wave of COVID-19'*. 2021. URL: https://github.com/MrinankSharma/COVID19NPISecondWave.

[13] M. Sharma et al. "Understanding the effectiveness of government interventions in Europe's second wave of COVID-19". In: *Nat Commun* 12.5820 (2021). DOI: 10.1101/2021.03.25.21254330.

[14] *Convergence and efficiency diagnostics for Markov Chains*. URL: https://mc-stan.org/rstan/reference/Rhat.html.

[15] *Neural network approach to Iris dataset*. URL: https://www.kaggle.com/code/louisong97/neural-network-approach-to-iris-dataset/notebook.

[16] *Neural Network Model for House Prices (TensorFlow)*. URL: https://www.kaggle.com/code/zoupet/neural-network-model-for-house-prices-tensorflow/notebook.