Multimodal Probabilistic Reasoning for Prediction and Coordination Problems in Machine Learning

Jaleh Zand

St Cross College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Michaelmas 2021

To my son,
Julian

Acknowledgements

First and foremost, my heartfelt thanks to my supervisor Stephen Roberts, who has been a tremendous source of support and encouragement to me throughout this process. He has been engaged in all aspects of my research, allowing me the freedom to explore my ideas whilst providing me with constructive insights and interesting perspectives. Above all, he has been patient and always considerate of my personal circumstances. I am truly privileged to have the opportunity to work with him. My special thanks to Jack Parker-Holder who collaborated with me on the second part of my thesis and made this final phase of my research even more exhilarating. I would like to thank everyone at the Machine Learning Research Group for the insightful discussions on various topics and their continued support during my time at Oxford. I am deeply thankful to the AIMS program for providing me with the opportunity, and in particular Wendy Poole, who has been attentive throughout and facilitated everything smoothly, despite the challenges posed by the pandemic. I also extend my sincere gratitude to the Oxford-Man Institute of Quantitative Finance and ESRC for providing me with the scholarship for this program.

I am indebted to Grigoris Pavliotis for believing in me and providing me the opportunity to return to academia to complete my Masters; and my extended thanks to Henrik Jensen for being an amazing MSc supervisor and for giving me the privilege to work on various problems with him that ultimately instigated me to pursue my PhD. I am grateful to my former mentors Ralph, Luc and Mbar for inspiring me down this path a long time ago and for their continued support over the years. I would like to thank my long-time friends, Sandie, Dora, Kate, Alex, Debora, Mike, Nasim and Azadeh whose support and encouragement over the years has meant a lot and I am fortunate to have them in my life.

To my beloved family - My sister, Ladan and her family who have always been there for me in time of need. My mother, Zohreh, who has been a tremendous support throughout my life, supporting and encouraging me through all the ups and downs during my research and always encouraging me to reach for the stars. I am forever indebted to her. Finally, my husband Ulrich and my son Julian, who have been patient and understanding throughout my PhD and motivating me in times when I found it difficult to continue. I am truly blessed to have you all in my life.

Statement of Originality

I declare that this thesis is my own work and it is submitted for the degree of Doctor of Philosophy at the University of Oxford. No part of this thesis has been submitted elsewhere for any other degree or qualification. Some of the work presented in this thesis has been published in journals and conference proceedings.

- **Chapter 3**:
  - Zand, J. and S. Roberts (2017). "MiDGaP: Mixture Density Gaussian Processes". *NIPS 2017 Time Series Workshop*.
- **Chapter 5**:
  - Zand, J. and S. Roberts (2019). "Mixture Density Conditional Generative Adversarial Network Models (MD-CGAN)". *ICML 2019 Time Series Workshop*.
  - Zand, J. and S. Roberts (2021). "Mixture Density Conditional Generative Adversarial Network Models (MD-CGAN)". *Signals* Vol 2, no. 3, pages 559–569.
- **Chapter 7**:
  - Zand, J., J. Parker-Holder, and S. Roberts (2021). "On-the-fly Strategy Adaptation for ad-hoc Agent Coordination". *NeurIPS 2021 Cooperative AI Workshop*.
  - Zand, J., J. Parker-Holder, and S. Roberts (2022). "On-the-fly Strategy Adaptation for ad-hoc Agent Coordination". *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS) 2022*.

Abstract

In this thesis we consider the role of *multimodality* in decision making and coordination problems in machine learning. We propose the use of a series of multimodal probabilistic methods, using extensions of (finite) mixture models to tackle challenges in time series forecasting, efficient uncertainty quantification in neural networks, adversarial models and multi-agent coordination.

In the first part of the thesis, we focus on the usage of multimodal uncertainty estimation for time series forecasting, showing that such approaches offer tractable, beneficial alternatives to point estimation methods, which remains a prevalent method of choice for prediction. We discuss the significance of multimodal uncertainty and show the necessity for more adept approaches to estimate posterior target distributions. We present a series of computationally efficient, yet capable, methods for estimating rich multimodal posterior distributions. We compare our models to techniques that estimate uncertainty with point measures or a unimodal distribution and conclude this part with an extension of the approaches developed, inspired by generative adversarial networks. We show that this method provides state-of-the-art robustness to additive noise, making it particularly suitable for data sets in which unknown stochastics abound.

In the second part of this work, we investigate the importance of multi-modal models for Cooperative Multi-Agent Systems (CMASs), extending our work to take a probabilistic approach. To date, the majority of research in this area has been confined to considering a *self-play* paradigm, even if the approaches tackle a variety of challenging problems. Whilst these advances are significant, the use of arbitrary conventions in self-play leads to coordination problems when agents play outside this setting. We consider *ad-hoc* CMAS settings, moving away from the self-play framework. This is a particularly challenging area in machine learning and one that has attracted significant attention in recent years, offering the promise of AI agents able to interact effectively with humans (and other agents) in real-world settings. We tackle the problem of ad-hoc coordination by formulating posterior beliefs over other agents' strategies. This is achieved using an extension of Gibbs sampling to obtain close-to-optimal ad-hoc performance. We

test our algorithm on the challenging game of Hanabi, one of the most prominent testbeds for cooperative multi-agent reinforcement learning and one which has gained momentum as a benchmark in recent years. We show that our method can achieve strong cross-play even with unseen partners, achieving successful ad-hoc coordination without up-front knowledge of the partners' strategies.

Contents

List of Figures

List of Tables

List of Abbreviations

AdaGrad . . . Adaptive Gradient

Adam . . . . . Adaptive Moment Estimation

AR . . . . . Autoregressive

BAD . . . . . Bayesian Action Decoder

BDL . . . . . Bayesian Deep Learning

BNN . . . . . Bayesian Neural Network

CBR . . . . . Common Best Response

CGAN . . . . Conditional Generative Adversarial Network

CMAS . . . . . Cooperative Multi-Agent System

CNN . . . . . Convolutional Neural Network

DB . . . . . Daily Birth

Dec-POMDP Decentralized Partially-Observable Markov Decision Process

Deep-MDN . . Deep Mixture Density Network

DL . . . . . Deep Learning

DQN . . . . . Deep Q-Network

fBME . . . . . fast Bayesian Mixture Experts

FX . . . . . Foreign Exchange

GA . . . . . Genetic Algorithm

GAN . . . . . Generative Adversarial Network

GMM . . . . . Gaussian Mixture Model

GP . . . . . Gaussian Process

HLE . . . . . Hanabi Learning Environment

HOAD . . . . Hanabi Open Agent Dataset

IMGPE . . . . Infinite Mixture of Gaussian Process Experts

KL . . . . . Kullback–Leibler

LSTM . . . . . Long Short-Term Memory

MAPPO . . . Multi-Agent Proximal Policy Optimization

MARL . . . . Multi-Agent Reinforcement Learning

MC . . . . . Monte Carlo

MCMC . . . . Markov Chain Monte Carlo

MD-CGAN . Mixture Density Conditional Generative Adversarial Network

MDN . . . . . Mixture Density Network

MDP . . . . . Markov Decision Process

MDT . . . . . Minimum Daily Temperature

ME . . . . . Mixture of Experts

MGP . . . . . Mixture of Gaussian Processes

MH . . . . . Metropolis-Hastings

MiDGaP . . . Mixture Density Gaussian Processes

MSE . . . . . Mean Square Error

NG . . . . . Natural Gas

OSA . . . . . On-the-fly Strategy Adaptation

POMDP . . . Partially Observable Markov Decision Process

PPO . . . . . Proximal Policy Optimization

RBF . . . . . Radial Basis Function

RBFN . . . . . Radial Basis Function Network

ReLU . . . . . Rectified Linear Unit

RNN . . . . . Recurrent Neural Network

SC . . . . . Small-Cap

SNN . . . . . Standard Neural Network

TRPO . . . . . Trust Region Policy Optimisation

USIJC . . . . . U.S. Initial Jobless Claims

VIX . . . . . Volatility Index

ZSC . . . . . Zero-Shot Coordination

1 Introduction

Contents

In modelling and evaluation of many machine learning problems, point estimation is often relied upon. However this fails to take into account the uncertainty involved on many levels. A more principled approach acknowledges uncertainty beyond point evaluation and estimates posterior distributions. This enables better decision making and forecasting.

1.1 The Case for Computation beyond Point Estimation

One of the most common reasons for choosing point estimation methods is that they are often computationally more efficient compared to probabilistic approaches. In particular, in problems where big data is available, there is tendency to choose

a point estimation method. While there is a balance to be struck between computational efficiency and improved accuracy when choosing a method, for many problems it is beneficial to choose uncertainty estimation. Some of the most common reasons for this are as follows:

- Data is not always available in abundance, and there are many real-world problems with limited amounts of data. This lack of abundance leads to uncertainty in model parameters and hyperparameters.

- Even in the case where data is readily available, observations might be noisy, lack accuracy or are missing.

- There is uncertainty in the choice of our model, as many models could potentially fit our observed data well.

- In any estimation we rely on minimising a loss function. For many problems the exact form of the loss function might not be available and quantifying uncertainty in these problems is more crucial.

We consider that point estimation fails to capture output complexity and risk for many real-world problems and so modelling posterior (predictive) uncertainty becomes paramount.

1.2 Posterior Uncertainty

For real-world problems, knowing merely the expected outcome, rather than an expected range or distribution of plausible outcomes, leads to loss of salient information at best and catastrophic underestimation of risk at worst. It is thus vital to entertain models that provide an (ideally well-calibrated) measure of outcome uncertainty at the very least. Although one may consider a variety of measures of such outcome ignorance, the only principled approach is that of probability theory (see, for example, Bishop 1995), which allows us to define a

probability density function over the outcome space. For real-valued regression or prediction problems, this density function can be seen as the probability associated with a range of outcome values. For classification or decision problems, the density lies over the class or decision beliefs. In both cases, reward (or loss) associated with the problem under consideration is defined via the probability distribution. For regression or forecasting problems the loss is defined via the negative (log-) likelihood of the true data evaluated with respect to the posterior probability function. For classification or decision making, the loss is defined in a similar manner, inducing the *cross-entropy* error function.

Particularly for regression problems (which we consider first in the thesis) the posterior distribution is often parameterised. This has the advantage of parsimony, with the Gaussian distribution being more widely used than any other (in the main due to its simplicity, ubiquity and conjugacy). Though a significant improvement to point estimation methods, reliance of a unimodal invariant functional form can lead to poor uncertainty modelling when the system in question has multiple modes, or is significantly non-Gaussian. Our approach is to retain the inherent simplicity of parameterised output densities, but to allow the parameters to encode a *mixture model*.

1.3 Thesis Outline

1.3.1 Part I

Mixture models are a common approach used to estimate multimodal probability density functions (Titterington et al. 1985). In Part I we consider the use of *Gaussian Mixture Models* (GMMs) to estimate the posterior distribution, with particular emphasis on time series forecasting. GMMs are a popular class of finite mixture models, in which the component density functions are assumed to be Gaussian. As mixture models offer considerable flexibility, all the models presented in Part I are capable of estimating a posterior distribution that is

non-Gaussian, including multimodal. This provides improved posterior flexibility, not only compared to models that make point estimations, but also compared to models that estimate uncertainty using unimodal distributions only.

In Chapter 3 we introduce methods of parameter inference for the posterior GMM model. Both approaches use Gaussian Process (GP) regression, inspired by the fact that the parameters in the GMM model can themselves be modelled as variables with uncertainty and evaluating these parameters using Gaussian Processes provides posterior distributions for each parameter. We compare these methods with other commonly used models that estimate the posterior distribution.

In Chapter 4, we extend our method and introduce Deep Mixture Density Network (Deep-MDN) models which estimate the latent variables, namely the GMM parameters, using deep neural networks; in particular *convolutional neural networks* (CNNs) (LeCun 1989) and *long short-term memory networks* (LSTM) (Hochreiter and Schmidhuber 1997). This is a natural extension of the Mixture Density Network model introduced by Bishop 1995. We compare the Deep-MDN methods with (approximate) Bayesian deep learning regression models, where dropout is used as a technique to estimate uncertainty. As previously discussed, the computational cost of probabilistic approaches is the main hindrance in choosing these models. We therefore show that Deep-MDN is significantly more time-efficient in comparison to Bayesian deep learning approaches using dropout.

In Chapter 5 we extend our approach further and introduce the Mixture Density Conditional Generative Adversarial Network (MD-CGAN) model. The MD-CGAN is an extension of the Deep-MDN model that is inspired by the *Generative Adversarial Networks* (GANs) (Goodfellow et al. 2014) and estimates a GMM posterior distribution. We show that this model improves the performance of the Deep-MDN models, and that it is also an effective method for noisy time series compared to some other commonly used methods in statistics.

1.3.2 Part II

In Part II, we investigate the benefits of multimodality in *Cooperative Multi-Agent Systems* (CMASs), using a probabilistic approach. In our experiments we consider ad-hoc CMAS settings, where agents interact with other agents with different policy types. This is a particularly challenging area in machine learning which has attracted significant attention in recent years.

In these settings, the environment can be viewed as a system with a population of policies, whereby each policy forms a component, or a mode, of the population. We can therefore model the system as a mixture of policies, similar to the mixture model framework of Part I. Approaching the problem in this form, we use Bayesian inference to identify the probability that an agent is interacting with each policy, π_i, in a set of policies $\pi_i \in \Pi$. This is equivalent to evaluating the probability of each component in a policy mixture model. To evaluate this in a partially-observable CMAS environment, agents need to estimate the joint probability distribution of the other agents' policies and the hidden information, which is not available to them. This joint probability distribution is often intractable and not possible to directly estimate. Using Bayesian inference, in particular Gibbs sampling, we estimate this joint probability for a set of fully-cooperative Markov games.

Further in ad-hoc CMAS problems that we investigate, we assume that the agents interact in Decentralized Partially-Observable Markov Decision Process (Dec-POMDP) environments (Bernstein et al. 2002; Nair et al. 2003), using different set of policies. The Dec-POMPD is a particularly good setting in the presence of uncertainty given that it relaxes the assumption of the Markov decision process (MDP) framework, which requires the agents to have access to the states of the environment with full certainty. We discuss this setting in more details in Section 2.3.

In Chapter 6, we use Gibbs sampling in a three time-step card game. We show that using Gibbs sampling agents are able to accurately identify the other

agent in the majority of the games and therefore achieve higher rewards. We expand on the Gibbs sampling approach in Chapter 7 and introduce the On-the-fly Strategy Adaptation (OSA) algorithm. We run experiments with OSA first on the Slime Volleyball game (Ha 2020); and then on the challenging Hanabi learning environment, where the need for cooperation between agents, in addition to partial observation and limited communication between the agents, has made it a perfect game and a benchmark in recent years for CMASs in both self-play and ad-hoc settings (Bard et al. 2020).

Finally, in Chapter 8 we present a summary of the overall contributions of this work and discuss interesting extensions for future experiments.

2

Overview of Concepts and Methods

Contents

In this chapter we discuss background material, relevant concepts and methods that are central to this work. We provide an overview of the main concepts, followed by a brief discussion of relevant models. We then present multi-modal posteriors, and mixture models in particular, as a means for quantifying outcome uncertainty in non-Gaussian posterior distributions. This is followed by a discussion of the problem of coordination in cooperative multi-agent systems; and reinforcement learning as a prevalent methods for solving multi-agent problems. We end this chapter by briefly presenting sampling and optimisation methods that are used, implicitly or explicitly, throughout this thesis.

2.1 Underlying Concepts

2.1.1 Time Series

Time series analysis has a rich and varied history, potentially dating back thousands of years. In a modern context, it has been an area of considerable interest over the past decades with the accelerating pace of digital communications and a move towards ever more present algorithmic integration in areas such as mobile networks, finance, climate and weather studies to name but a few.

We start our overview by considering outcome representations that form the mathematical basis of stochastic processes, which we see as a foundation for time series analysis. In probability theory each individual outcome is called an *elementary event* denoted by ω, and the set of all elementary events is called a *sure* event and is denoted by Ω. If A is a subset of Ω, $\mathcal{F}$ the collection of such subsets, event ω is observed and ω is in A, then A has occurred and $P(A)$, the probability that A occurs, satisfies the axioms:

- $P(A) \geq 0$ for every A in $\mathcal{F}$.

- $P(\Omega) = 1$.
- If $A_1, A_2, ...$ is a sequence from $\mathfrak{F}$ and $A_i \cap A_j = \emptyset$ for all $i \neq j$ then $P(\cup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A_i)$.

For most problems it is impractical to consider a full probability space and compute all possible outcomes of an experiment (if this is even possible). In practice, therefore, we consider recording those outcomes that are assigned real values. Thus, for an observed outcome ω we record random variable $x(\omega)$ which we denote as x. Formally, the random variable x is defined as a real valued function on sample space Ω, where for every real number s, and the set of elementary events ω in Ω, $\{\omega \in \Omega \mid x(\omega) \leq s\} \in \mathfrak{F}$. Further the function $F_x(x) = P(\{\omega : x(\omega) \leq s\})$ is known as the distribution function of x (Tucker 2013). We refer the reader to Tucker 2013 for further discussion on probability spaces, random variables and stochastic processes.

Assuming $\mathfrak{F}$ to be a σ-algebra on the set Ω (this is required so that the three axioms above are satisfied) and T to be an index set, a *time series* or *stochastic process* is a real valued function $x(t, \omega)$ defined on $T \times \Omega$ where for each fixed value of t, $x(t, \omega)$ is a random variable on the probability space $(\Omega, \mathfrak{F}, P)$. The function $x(t, \omega)$ is usually denoted as x_t where the time series is a collection random variables $\{x_t : t \in T\}$, and the observed values are referred to as the *realisation* of the time series (Fuller 2009). These realisations can form sampled data which, for the most part, is discrete because of restrictions intrinsic to the methods of collection.

The correlation, originating from the sampling of neighbouring points, can restrict the relevance of many conventional statistical models that require these neighbouring points to be independent. The cogent approach by which one investigates the challenges posed by these time correlations is referred to as *time series analysis* (Shumway and Stoffer 2000).

One of the most important problems in time series analysis is that given a realisation consisting of n observations, how can one predict the $(n+k)^{\text{th}}$

observation in that realisation. This prediction is often noted as *time series forecasting* (Fuller 2009). The time series models that we present and discuss in this work are forecasting models. Further we use the term *time series* either for a particular realisation or the stochastic process depending on the context.

2.1.2 Regression Analysis

The regression of the random variable y, denoted as the *dependent variable*, the *response variable* or the *output*, on random variable x which we refer to as the *independent variable*, the *explanatory variable*, or the *input*, is any feature of the probability distribution of random variable y conditional on random variable x. The estimated feature by regression can be the mean, median, variance or any other variable of interest (Manski 1991).

Regression analysis are often used for two main purposes, to make a prediction or to determine if there are any causal relationships between the explanatory and response variables (Freedman 2009).

A best estimation of response variable y, conditional on explanatory variable x, minimises the expected loss given a predetermined loss function (examples include least square error, minimum absolute deviation). Using a valid loss function, the estimation of expectations (or other statistics) of the random variables within the data range is often referred to as model-fitting or interpolation, and the estimation outside the observed range is referred to as extrapolation. Extrapolation is commonly used for prediction and forecasting of a random variable y, conditional on random variable(s) x.

2.1.3 Uncertainty

Uncertainty is a key concept for any estimation or evaluation in machine learning. The sources and types of uncertainty are varied and include uncertainties over the collection and measurements of the explanatory variables, the estimation of the

response variables, the choice of model and its parameters, and the choice of the loss function.

There are two types of uncertainty. Aleatoric uncertainties arise from randomness and are irreducible, and more data or better models will not reduce them. An example of aleatoric uncertainty is the randomness in the outcome of tossing a fair coin. Epistemic uncertainties, on the other hand, are perceived as due to lack of knowledge- and can thus be reduced, in principle (Kendall and Gal 2017).

We start with a short overview of probability theory and its relationship to uncertainty estimation. Probability theory provides arguable the longest standing representation of uncertainty and the most developed mathematically (Stigler 2005). Probability theory thus forms a rational approach to quantify uncertainty enabling, for example, better decision making (Bishop 2006). We note that there are still multiple schools of thought regarding the interpretation, and manipulation, of probabilities. The two prevalent paradigms are the *frequentist approach* and the *Bayesian approach*. The frequentist approach has an objective view of probability where one collects a set of observations, an n-element subset $\Omega(n)$ of the sample space Ω. As n increases $\Omega(n)$ becomes representative of Ω, and the frequencies f_i converge to probability values at the limit, therefore $p_i = \lim_{n\to\infty} f_i$. In the frequentist approach one needs a large amount of sample observations and events have to be repeatable (Marquis et al. 2020).

The Bayesian method is the other popular probabilistic approach that is powerful when dealing with uncertainty. The Bayesian inference relies on the prior distribution which is based on one's initial beliefs. The Bayesian inference then uses Bayes' rule to form a posterior distribution using the prior and the fresh evidence. While the Bayesian method has its origin in the 18^{th} century, its use had been limited until recent decades when it increased with the development of sampling methods (e.g. Markov Chain Monte Carlo) and the improvements in the speed and memory of modern computational designs (Bishop 2006).

In a Bayesian setting a epistemic uncertainties are represented via posterior probability distributions. These posterior probabilities represent, as example, the confidence in an estimation, given all the available information - with the possibility of updating the confidence as more information becomes available.

Furthermore, under the Bayesian paradigm, the separation of the epistemic and aleatoric uncertainty, and the distinction between them, is left to interpretation and is conditional on both context and the ability to gather knowledge (Oden et al. 2010) as well as the manner in which explicit noise terms are encoded in the model.

Bayesian Methods

Inference, or learning, in Bayesian models relies ultimately on Bayes' theorem. Although simple, this provides an explicit mechanism for estimating posterior beliefs and for sequentially updating such beliefs as more evidence becomes available. We discuss the very basics only here, allowing more focused detail to be presented in subsequent chapters as and when needed by other content.

We start by defining $\mathbf{x}$ to be the input to a model, $\mathbf{y}$ to be the output variable associated with the input and θ to be the vector of parameters which define the input-output mapping of the model. Most of the time, it is this set of parameters we wish to infer. We note that $\mathbf{x}$ is observed independent of the parameter vector θ, hence using Bayes' theorem parameter inference involves updating the *prior* belief $p(\theta)$ to the posterior, having been provided $\mathbf{x}, \mathbf{y}$, namely $p(\theta \mid \mathbf{x}, \mathbf{y})$. We thus may write:

$$\begin{aligned} p(\theta \mid \mathbf{x}, \mathbf{y}) &= \frac{p(\theta, \mathbf{x}, \mathbf{y})}{p(\mathbf{x}, \mathbf{y})} \\ &= \frac{p(\mathbf{y} \mid \theta, \mathbf{x}) p(\mathbf{x}) p(\theta)}{\int p(\mathbf{x}, \mathbf{y} \mid \theta) p(\theta) d\theta} \\ &= \frac{p(\mathbf{y} \mid \theta, \mathbf{x}) p(\theta)}{\int p(\mathbf{y} \mid \mathbf{x}, \theta) p(\theta) d\theta} \end{aligned} \tag{2.1}$$

Here the posterior is $p(\theta \mid \mathbf{x}, \mathbf{y})$, $p(\mathbf{y} \mid \theta, \mathbf{x})$ is the likelihood function, encoding the probability of an observed output from the model conditioned on the

current vector of parameters, θ and input observation, $\mathbf{x}$. The denominator in Equation 2.1 can be formulated as $p(\mathbf{y} \mid \mathbf{x})$ and is the marginal likelihood or model evidence. It is a normalisation constant and ensures the posterior to be a valid probability density distribution integrating to one. Therefore Equation 2.1 can be reformulated as follows:

$$\text{posterior} \propto \text{likelihood} \times \text{prior} \tag{2.2}$$

The formulation in Equation 2.1 is appealing as it provides a solution for continually updating information about θ as new observations become available. It is a process of learning from experience that provides an approach to iteratively combining new and past knowledge (Box and Tiao 2011).

Given posterior distributions for the parameters, a new observation x^* induces a posterior predictive distribution over the output y^* as:

$$\begin{aligned} p(y^* \mid x^*, \mathbf{x}, \mathbf{y}) &= \int p(y^*, \theta \mid x^*, \mathbf{x}, \mathbf{y}) d\theta \\ &= \int p(y^* \mid x^*, \theta, \mathbf{x}, \mathbf{y}) p(\theta \mid \mathbf{x}, \mathbf{y}) d\theta \end{aligned} \tag{2.3}$$

We note that in the frequentist approach, we try to optimise the vector parameter θ that best explains the observed data. This may readily be achieved either by directly maximising the data likelihood $p(\mathbf{y} \mid \theta, \mathbf{x})$ (Maximum Likelihood Estimation - MLE) or by maximising $p(\mathbf{y} \mid \theta, \mathbf{x})p(\theta)$ (Maximum *A Posteriori* - MAP), with the MLE approach being arguably the most commonly used method:

$$\theta_{\text{MLE}} = \arg\max_{\theta} p(\mathbf{y} \mid \theta, \mathbf{x}) \tag{2.4}$$

We may see this *plug-in estimator* for the parameters as reducing the marginal integral of Equation 2.3 by replacing the distribution over θ by a delta-function at $\theta = \theta_{\text{MLE}}$.

In a fully Bayesian framework the *marginal integral* of Equation 2.3 is evaluated, albeit often approximately. This preserves quantification of predictive uncertainty by integrating out over possible values of θ (MacKay 1996a). Performing exact

inference therefore requires evaluating integrals over the parameter space, which can be computationally costly depending on the nature of the distribution and the number of parameters defining the space. Therefore, practical Bayesian inference methods often use approximations. The two most popular approaches for approximation are sampling methods (Gelfand and Smith 1990), and distribution approximation methods, which allow tractable re-estimates of the desired quantities to be obtained. The latter approaches are, in a modern setting, dominated by variational Bayes methods (Bishop 2006).

2.2 Function Modelling

In this section we present a brief overview of several models with *universal approximation* (Park and Sandberg 1991) properties, which are common in the literature for learning unknown functions or mappings. A more detailed discussion of the methods is presented, as appropriate, when they appear later in the thesis.

2.2.1 Basis Function Models

We start with simple non-linear extensions to linear regression methods. A linear regression model can be formulated as follows:

$$y_t = w_0 + w_1 x_{t1} + ... + w_D x_{tD} + \epsilon_t, \quad t \in \{1, ..., T\} \tag{2.5}$$

where the model is a linear function of the weight parameters $\mathbf{w} = [w_0, w_1, ..., w_D]^T$ as well as the explanatory variables $\mathbf{x_t}$.

The family of basis function models are an extension, whereby the response variable is a linear combination of a set of non-linear functions as follows:

$$\begin{aligned} y_t &= w_0 + \sum_{i=1}^{P-1} w_1 \phi_i(\mathbf{x}_t) + \epsilon_t \\ &= \mathbf{w}^T \phi(\mathbf{x_t}) + \epsilon_t \end{aligned} \tag{2.6}$$

where $\phi_t(\mathbf{x_t})$s are the basis functions, $\phi = [\phi_0, ..., \phi_{p-1}]^T$ and $\phi_0(\mathbf{x_t}) = 1$. We note that the mean of the posterior distribution in a Gaussian Process (we discuss

Gaussian Processes in Section 2.2.4) is a linear function of n kernel functions, acting in effect as a basis function; and that a Gaussian Process regression can be seen as Bayesian linear regression with an infinite number of basis functions (Rasmussen and Williams 2005). In addition, using Mercer's theorem, it can be shown that for every positive definite covariance function k, there exists an expansion in terms of an infinite number of basis functions (we refer the reader to Section 4.3 in Rasmussen and Williams 2005 for further details on this).

Radial Basis Functions

Radial basis functions (RBFs) are a subclass of basis function models, in which the kernel function is dependent on a distance measure (typically the L2 distance) from a centre (or location) parameter μ_i for each basis, such that:

$$\phi_i(\mathbf{x}) = h(\| \mathbf{x} - \mu_\mathbf{i} \|) \tag{2.7}$$

$$y_t = \sum_{i=1}^{P} w_i h(\| \mathbf{x} - \mu_\mathbf{i} \|) + \epsilon_t, \quad t \in \{1, ..., T\}. \tag{2.8}$$

We note that, at their introduction (Powell 2001), RBF models were formulated such that:

$$y_t = \sum_{t=1}^{T} w_t h(\| \mathbf{x} - \mathbf{x}_t \|) + \epsilon_t, \quad t \in \{1, ..., T\}. \tag{2.9}$$

In this form, the number of basis functions is equal to the number of data points (in the training data set for example) with basis functions centered on each datum.

Equation 2.9, with a number of basis functions equal to the number of data, has clear computational cost implications when there are a very large number of data points. This is because we need to calibrate any kernel parameters (typically a kernel "bandwidth" - essentially a width parameter, akin to similar parameters found in Parzen density estimation methods (Parzen 1962)and the linear coupling weights associated with each basis function.

We may avoid some costs by restricting the number of basis functions (Equation 2.8), such that the number of basis functions P is less than the total number

of data points T. The locations of the kernel centres, $\mu_{\mathbf{i}}$, in this extended form are usually determined based on the input variables $\{\mathbf{x_t} : t \in T\}$ alone. One of the simplest options is to choose a subset of points from $\{\mathbf{x_t} : t \in T\}$ at random (Bishop 2006). Alternatively, we can select centre points through supervised and unsupervised methods such as clustering, classification, or vector quantization techniques (Schwenker et al. 2002).

The optimal kernel is dependent on the problem and the characteristics of the dataset, and its choice can have a significant impact on the performance and stability of the model.

Training of RBF methods is often achieved through a two-phase process, thus separating training of lower- and upper-layers in the network with consequent computational savings (of course, it is possible to jointly optimise, in a supervised manner, all the parameters in the model simultaneously. This adds to compute overheads and is rarely performed). In the first phase, the kernel is chosen (common choices include the Gaussian kernel and thin-plate splines) along with the location of the kernel centre points and the kernel width parameter(s). We can optimise the kernel width as a global parameter, or it can be estimated individually for each basis function. The choice of a global or local kernel width is dependent on the distribution of data in the input space. For both cases the kernel width is determined using the input data $\mathbf{x}_t$ and this kernel optimisation phase is akin to density estimation. In the second stage of RBF optimisation, we infer the weights coupling the kernels to the output nodes, using both input $\mathbf{x}_t$ and target data y_t, via a supervised learning method. As the output nodes are normally linked to the kernels by a set of linear parameters (perhaps through a softmax link function in the case of classification), it is possible to use a simple (regularised) linear or logistic regression model. For linear regression, under a least-squares error function, this makes the second stage of RBF training particularly easy, as the weights may be estimated using standard extensions of matrix (pseudo-)inversion

methods. Even more sophisticated approaches to inference, such as variational Bayes, are computationally tractable for linear weight problems.

2.2.2 Deep Neural Networks

We start by considering the simplest building block of the neural network architecture, namely the Multilayer Perceptron (Gardner and Dorling 1998).The Multilayer Perceptron consists of a mapping from input $\mathbf{x_t}$ (associated with time t say) to output y_t via a function $f(.)$ of a linear basis expansion of the input:

$$y_t = f\left(\sum_{i=1}^{P} w_i \phi_i(\mathbf{x_t})\right). \tag{2.10}$$

Here, $\phi_i(.)$ are the basis functions operating on the input $\mathbf{x_t}$, w_i are a set of weights (or parameters, which need to be estimated) and $f(.)$ maps the resultant linear basis combination to the output. In this formulation both $\phi(.)$ and $f(.)$ are fixed function maps.

We take the first layer of the network to be a linear combination of a D-dimensional input variable $\{x_1, ..., x_D\}$, each time instance t, $\mathbf{x}_t \in \mathbb{R}^D$. We consider there to be P neurons in the first layer. Dropping for the moment the explicit time suffix on $\mathbf{x}_t$, we may write this combination as:

$$a_i = \sum_{j=1}^{D} w_{ij}^1 x_j + w_{i0}^1 \tag{2.11}$$

where $i \in \{1, ..., P\}$ and the parameters w_{i0}^1 are the bias weights of the first layer (effectively setting a "mean" level in the response). The variable a_i is denoted as the activation and is subsequently transformed via $z_i = h(a_i)$, where function $h(.)$ is denoted the activation function. Assuming layer n to have Q neurons, the output of the q^{th} neuron, $q = \{1, ..., Q\}$, in this layer is as follows:

$$a_q = \sum_{k=1}^{S} w_{qk}^n z_k + w_{q0}^n \tag{2.12}$$

where S is the number of the outputs in layer $n-1$ and parameters w_{q0}^n are the bias weights of the n^{th} layer.

We later discuss two other types of feed-forward deep neural networks; convolutional neural networks (CNNs) and long short-term memory networks (LSTMs) in Sections 4.3.1 and 4.3.2 respectively, in Chapter 4.

2.2.3 Bayesian Deep Learning

Deep learning (DL) has been a very successful supervised learning method and able to learn powerful representations in various machine learning systems. DL models allow learning from experience through a hierarchy of concepts, where each concept is defined by simpler concepts. This hierarchy of relationships between the concepts is not determined upfront and is part of the learning process. The graph connecting these concepts is deep and therefore this type of reasoning is noted as DL (Goodfellow et al. 2016).

In their basic form however, DL models do not assign uncertainty levels to their predictions and make only point estimations. However, as discussed in Chapter 1, point estimation alone (with no quantification of uncertainty) can lead to high risk decision making, forecasting and actions. Bayesian deep learning (BDL) models try to overcome this deficiency by combining Bayesian inference techniques with DL methods. These approaches naturally cover a broad area of research that includes Bayesian neural networks (BNN) (Denker and LeCun 1990; MacKay 1992; Neal 1995) as well as deep generative models (Kingma et al. 2014). We discuss the BNN model further in Section 4.4 in Chapter 4.

2.2.4 Gaussian Process Regression

Gaussian Processes form an important subset of Bayesian non-parametric methods, in which we may entertain the notion of placing prior distributions over a space of functions. Observed data, via Bayes' theorem, can then be used to update the prior to a posterior over functions (Rasmussen and Williams 2005; Roberts et al. 2013).In the examples of Gaussian Process usage in this thesis, we perform regression

rather than classification, so the below introduction to Gaussian Processes is biased towards this usage.

We start by considering a set of response (dependent) variables $\{y_t : t \in T\}$ and independent variables $\{x_t : t \in T\}$ associated with a regression problem:

$$y_t = f(x_t) + \epsilon_t, \quad t \in \{1, ..., T\} \tag{2.13}$$

where $f(.)$ is the (unknown) regression function and ϵ_t the residual, assumed to be an additive white noise term in the Gaussian Process formalism, such that $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$.

Common approaches to solve this regression problem necessarily make assumptions about the nature of the function f, often assuming a particular parametric form and so limiting the class of functions that we consider. Gaussian Process methods avoid such explicit restrictions, by considering a prior distribution over functions which fit the constraints of the problem (e.g. boundary conditions, smoothness etc). We may hence see the Gaussian process as a functional generalisation of standard probability distributions for (scalar) samples. This is achieved by noting that a Gaussian process is defined as a collection of random variables (possibly an infinite collection), in which any finite subset of these random variables has a joint Gaussian distribution (Rasmussen and Williams 2005). The Gaussian process prior over possible functions is thus:

$$p(\mathbf{f} \mid x_1, x_2, ..., x_T) \sim \mathcal{N}(\mathbf{m}, K) \tag{2.14}$$

where $\mathbf{f} = [f(x_1), f(x_2), ..., f(x_T)]^T$ is a vector of function values associated with the (finite) set of independent variables, $\{x_1, ..., x_T\}$, $\mathbf{m}$ is the *mean function* (which is more often than not taken to be zero, or a constant value - such as the prior mean of the dependent variables in the observed data set) and K is the covariance matrix, defining the relationship between function evaluations. The covariance matrix itself is determined by *covariance function*, $k(.,.)$, such that $K_{f_i,f_j} = k(x_i, x_j)$. As we may expect from knowledge of kernel methods (including, for example,

RBF networks), the covariance function, $k(x_i, x_j)$, must be of functional form so as to induce K to be a valid covariance matrix, namely symmetric and positive semi-definite. We note that a stationary covariance function is a function of τ such that, $\tau = \mathbf{x} - \mathbf{x}'$ and in this case the covariance function can be represented as $k(\tau)$. Using Bochner's theorem (we refer the reader to Stein 1999 for further details on the theorem), the covariance function of a stationary process $k(\tau)$, can be represented as Fourier transform of a positive finite measure μ with density $S(\mathbf{s})$,

$$k(\tau) = \int_{\mathbb{R}^D} e^{2\pi i \mathbf{s}.\tau} d_\mu(\mathbf{s}) \tag{2.15}$$

where S is known as the *spectral density* corresponding to k, and the spectral density and the covariance function are Fourier duals of each other (Rasmussen and Williams 2005).

Consider a set of training example pairs of the form $(\{x_1, x_2, ...\}, \{y_1, y_2, ...\})$ which we may denote as $(\mathbf{x}, \mathbf{y})$. We further define a test set, for which we observe only the independent variables associated with the locations at which we wish to infer the Gaussian process output. Let this set be $\mathbf{x}^* = \{x_1^*, x_2^*, ...\}$. We define the Gaussian process function values on the training set to be $\mathbf{f} = (f(x_1), f(x_2), ...)^T$ and on the test set to be $\mathbf{f}^* = (f(x_1^*), f(x_2^*), ...)^T$. For inference over the test data, we allow a joint Gaussian process prior on the training and the test function values, $\mathbf{f}$ and $\mathbf{f}^*$ respectively. Combining this joint prior with the likelihood over the training data, $p(\mathbf{y} \mid \mathbf{f})$, and using Bayes' rule, the joint posterior may be written as:

$$p(\mathbf{f}, \mathbf{f}^* \mid \mathbf{y}) = \frac{p(\mathbf{f}, \mathbf{f}^*) p(\mathbf{y} \mid \mathbf{f})}{p(\mathbf{y})}. \tag{2.16}$$

We then marginalise over the training set to estimate the posterior $p(\mathbf{f}^* \mid \mathbf{y})$ as:

$$p(\mathbf{f}^* \mid \mathbf{y}) = \int p(\mathbf{f}, \mathbf{f}^* \mid \mathbf{y}) \mathbf{df} = \frac{1}{p(\mathbf{y})} \int p(\mathbf{y} \mid \mathbf{f}) p(\mathbf{f}, \mathbf{f}^*) \mathbf{df}. \tag{2.17}$$

In the problems for which we use Gaussian processes there is no particular domain knowledge to suggest the form for the mean function. We thus take the mean function to be a constant, and after suitable domain normalisation and

re-representation (without loss of generality) we are further able to assume zero mean function. We note that for many practical applications, this is acceptable if it is truly our prior belief (we refer the reader to Rasmussen and Williams 2005 for further details on GPs with non-zero mean functions). Further, the likelihood $p(\mathbf{y} \mid \mathbf{f})$, and the joint Gaussian process prior $p(\mathbf{f}, \mathbf{f}^*)$, are both Gaussian, thus:

$$\mathbf{y} \mid \mathbf{f} \sim \mathcal{N}(\mathbf{f}, \sigma^2 I) \quad \text{and} \quad \begin{bmatrix} \mathbf{f} \\ \mathbf{f}^* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K_{\mathbf{f},\mathbf{f}} & K_{\mathbf{f}^*,\mathbf{f}} \\ K_{\mathbf{f},\mathbf{f}^*} & K_{\mathbf{f}^*,\mathbf{f}^*} \end{bmatrix}\right) \tag{2.18}$$

where I is the identity matrix and σ^2 is the variance of the white noise term of Equation 2.13. Using the Gaussian properties of $p(\mathbf{y} \mid \mathbf{f})$ and $p(\mathbf{f}, \mathbf{f}^*)$, we can formulate the Gaussian predictive distribution as the following:

$$\mathbf{f}^* \mid \mathbf{y} \sim \mathcal{N}(K_{\mathbf{f}^*,\mathbf{f}}(K_{\mathbf{f},\mathbf{f}} + \sigma^2 I)^{-1}\mathbf{y}, K_{\mathbf{f}^*,\mathbf{f}^*} - K_{\mathbf{f}^*,\mathbf{f}}(K_{\mathbf{f},\mathbf{f}} + \sigma^2 I)^{-1}K_{\mathbf{f},\mathbf{f}^*}). \tag{2.19}$$

We note that the performance of the GP regression relies on the choice of the covariance function, which is specific to the data and the similarity between the inputs. The covariance function often has some free hyperparameters, denoted as Θ. The most common way to estimate the hyperparameters of a GP model is through optimisation and by maximising the $\ln p(\mathbf{y} \mid \mathbf{x}, \Theta)$ which is

$$\ln p(\mathbf{y} \mid \mathbf{x}, \Theta) = -\frac{1}{2}\mathbf{y}^T(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y} - \frac{1}{2}\ln \mid \mathbf{K} + \sigma^2\mathbf{I} \mid -\frac{T}{2}\ln(2\pi). \tag{2.20}$$

GPs are effective probabilistic models with easy implementation, albeit at some computational cost when data is high-dimensional or abundant. However, in common with many Bayesian modelling approaches, the method has some shortcomings. While it is important to measure uncertainty, in practical applications we need to make a point estimation guess that is most optimal for decision making. We therefore need a loss function $\mathscr{L}(y^*, y)$ that estimates the penalty of choosing the estimated value y^* for the true value y. In non-Bayesian methods, the model is often trained by minimising this loss. In Bayesian settings, on the other hand, the loss and the likelihood must be matched carefully. For example, by assuming the posterior to be Gaussian, the mean and the median become the same (in effect

producing a solution that is both a minimiser of L2 loss norm and L1 loss norms. However, there are many problems that can have asymmetric loss functions, a problem that is commented on in Rasmussen and Williams 2005.

Furthermore, to fit a GP model to a large number of data-points T, we need to invert a covariance matrix of size $T \times T$, which could be computationally costly. Various methods are presented to ease this computational cost by taking advantage of certain properties of the covariance matrix (Zhang et al. 2005; Tsiligkaridis and Hero 2013; Gibbs and MacKay 1997). These methods, however, rely on specific assumptions that may not be satisfied in all cases. There are other approaches that try to overcome this problem by reducing the size of the covariance matrix, using methods such as sampling a subset of the dataset (Foster et al. 2009; Quinonero-Candela and Rasmussen 2005). Despite these advances, computational efficiency remains a challenge for Gaussian Processes, especially in dataset with high dimensions.

2.3 Multimodal and Mixture Models

In a probability density function, a mode is a local maximum. For a density function with a single point of maximum, all data points are considered part of this single mode (Minnotte 1997).

A distribution is *multimodal* when a distribution exhibits more than one mode, and in this case, the mean of the distribution is misleading for any prediction or evaluation purposes. For a multimodal distribution, a good estimation would require evaluating each mode and its variance separately. One of the most common approaches in estimating distributions with more than one mode is to use *finite mixture models*. The models offer extreme flexibility, and can approximate distributions with any shape.

2.3.1 Finite Mixture Models

The *finite mixture model* constitutes arguably the most common form of multi-modal distribution (or function) modelling. In this approach, a datum is considered as drawn from one of a (finite) set of mixture components, selected via an indicator variable over which we place a degree of belief (often referred to as the component responsibility or component weight, but more strictly this is a categorical posterior distribution).

Let us consider a model with m components. If we were to sample a datum $X = x$ from the i^{th} component (such that the indicator variable $I = i$), we may denote the probability of obtaining x as:

$$P(X = x \mid I = i) = f(x; \Theta, \epsilon_i). \tag{2.21}$$

Here f represents some known density function (referred to as the component density), ϵ_i is an (unknown) parameter set that encodes the distributional attributes of the i^{th} component and Θ is a vector of parameters that encodes unknown characteristics that are common to the population.

The expected proportion of the total population that we believe is generated via sampling from the i^{th} component is also an unknown parameter, which we denote as α_i (the component responsibility or component weight) and $\sum_i \alpha_i = 1$. Therefore, the variables (X, I) are a random sample from the joint density described by:

$$P(X = x, I = i) = P(X = x \mid I = i)P(I = i) = f(x; \Theta, \epsilon_i)\alpha_i. \tag{2.22}$$

In practice, the component labels are unknown and we observe only the samples $X_1, ..., X_n$. Thus, the observed data are a sample from the mixture density model:

$$g(x; \alpha, \Theta, \varepsilon) = \sum P(X = x \mid I = i)P(I = i) = \sum f(x; \Theta, \epsilon_i)\alpha_i \tag{2.23}$$

where g is the mixture density function (Lindsay 1995). We note that, of course, a unimodal model is a special case of the mixture model, in which $m = 1$.

2.3.2 Mixture of Experts Framework

Jacobs et al. 1991 were the first to present the Mixture of Experts (ME) framework. The method is based on the divide-and-conquer paradigm, in which a complex problem is decomposed to simpler sub-components, each associated with an expert. Using the distributions of the data space, the model identifies distinct experts and creates a partition between them, training the collective set of experts either individually or as a cohort (Jordan and Jacobs 1994).

Each distinct expert component typically constitutes a solution based upon a feed-forward neural network. In addition to these expert modules, the aggregate model employs a gating network that allocates which expert to use for each training data point, based on a (stochastic) responsibility variable. The internal weights of the constituent networks are treated as independent of each other and, in many formulations, the networks are encouraged to compete with one another. In learning the parameter sets of the expert networks, Jacobs et al. 1991 use different error functions, with the aim of striking a balance between a global error function - to incentivise coherence across the experts - and a local error function that leads to diversification of expertise among the experts.

Since its introduction, there has been various extensions of the mixture of experts framework. Notably, the work of Tresp 2001, presents an alternative version of the method - the Mixture of Gaussian Processes (MGP) model, in which the original neural network experts are replaced by Gaussian Processes. We discuss further extensions of both mixture of experts and MGPs in Section 3.2 in Chapter 3.

2.3.3 Other models using mixture distribution posteriors

Independent Component Analysis

Independent component analysis is a method that transforms the data to linear maximally statistically independent components (Everson and Roberts 1999). The

algorithm attempts to recover source signals $\mathbf{s}$ from observation vector $\mathbf{x}$ as follows:

$$\mathbf{x} = V\mathbf{s}, \tag{2.24}$$

where V is a matrix of unknown mixing coefficients. The algorithm estimates the inverse of V, denoted as W where:

$$\hat{\mathbf{s}} = W\mathbf{x} \tag{2.25}$$

such that the components of $\hat{\mathbf{s}}$ are maximally independent. In order to define such independence, an explicit (proxy) measure of the mutual information between the components is required. This cannot be Gaussian, as this leads simply to linear decorrelation and hence to Principle Component Analysis. Several approaches to modelling the distributions of the elements of $\hat{\mathbf{s}}$ have been proposed, from the use of heavy-tailed parametric distributions (Bell and Sejnowski 1995; MacKay 1996b), generalised exponentials (Everson and Roberts 1999) to explicit formulations which use mixtures of Gaussians (Choudrey and Roberts 2003), inferred using (approximate) Bayesian approaches.

Generalised Autoregressive Models

Autoregressive models constitute a class of linear regressors, the parameters of which can, under a Gaussian noise model, be estimated using standard extensions of least-squares optimisation. If we are to assume that residuals, or noise processes are non-Gaussian - indeed even multi-modal - then we require a more expressive model. Generalised autoregressive processes model the noise process using a Gaussian Mixture Model. Therefore, in a generalised autoregressive model of order p, we may express the next datum in a time series, y_t, as a linear combination of p past samples"

$$y_t = \mathbf{x}_t\mathbf{w}^T + e_t \tag{2.26}$$

where $\mathbf{x}_t = [y_{t-1}, y_{t-2}, ..., y_{t-p}]$, and $\mathbf{w}$ is the vector of autoregressive parameters. The noise term e_t is treated as non-Gaussian, and evaluated via a mixture of Gaussians with m components, say. The overall parameter vector is $\theta = \{\alpha, \mu, \sigma, \mathbf{w}, \gamma\}$,

where α is the mixing coefficient vector of the noise mixture model, μ and σ are the mixture of Gaussian means and variances respectively, and γ is the precision for the Gaussian priors.

The model can be estimated using Markov Chain Monte Carlo (MCMC) sampling methods (Bell and Sejnowski 1995; Barnett et al. 1996) or variational Bayes (Roberts and Penny 2002) which is shown to be computationally more efficient.

2.4 Multi-Agent Systems

We define an agent as a actor that is imbued with a level of autonomy, allowing it to make decisions, perform actions and interact with its environment based on feedback that it receives. In a multi-agent environment we consider more than one agent, interacting both with each other and the environment. Often, these environments are *cooperative*, whereby multiple agents attempt to jointly complete tasks and maximise global utility (Panait and Luke 2005). Such multi-agent settings have extra difficulties when compared to single-agent environments, particularly the requirement to consider the behaviour of other agents and the evaluation of strategic responses that maximise local (or global) utility (Bard et al. 2020).

In most multi-agent systems the agents' behaviour can be broadly categorised as *cooperative* or *adversarial*. Further, agents can exhibit behaviour which is a mix of cooperative and adversarial strategies. In the problem domains we investigate in Part II, we consider the issue of learning effective cooperation between multiple agents.

2.4.1 Cooperative Multi-Agent Systems

Cooperation is widespread in nature amongst many species, but human cooperation arguably exceeds others in terms of magnitude and the range of cooperative activities. Indeed, this has conceivably led to the global success of humankind. Some of the key characteristics of human cooperation lie in the extensive behaviours

observed between strangers, who may be hitherto completely unrelated (Melis and Semmann 2010). In addition, the range of cooperation seen between humans varies from easy tasks such as helping move a large piece of furniture, to more complex challenges such as groups of hunters, driving in a highway, building a house or negotiating lasting peace. In a world in which software as well as human agents are ubiquitous, the ability of software agents to cooperate with both human and other software agents in the system, has made cooperative multi-agent systems (CMAS) an important area in machine learning research. With the advent of cheaper high-performance computing, hitherto complex multi-agent problems have been given more attention in recent years. Although much machine learning research to date has focused on improving the capabilities of single agents, using complex algorithms, cooperative multi-agent algorithms offer improved ability for agent groups to cooperate successfully, improve their social intelligence and provide arguably better solutions compared to even the best single-agent models (Dafoe et al. 2020).

Broadly, the cooperative multi-agent problem can be further split into two types of CMAS environments. The first that of *self-play*, in which the agents interact with the copies of themselves. In the second, so-called *ad-hoc CMAS*, the agents exhibit different behavioural policies and thus entertain different responses to the same environment. Ad-hoc CMAS is arguably the most commonly encountered environment in the real world of agent interactions. However, it is also considerably more complex and still remains a challenging problem area in machine learning, particularly in partially-observable environments, in which agents may only have access to different sets of information. We discuss our approaches to these problems in Part II of the thesis.

2.4.2 Decentralised Partially-Observable Markov Decision Processes

A Decentralised Partially-Observable Markov Decision Process (Dec-POMDP) (Nair et al. 2002) is a probabilistic model which defines degrees of belief over states and actions in the presence of uncertainty, induced by only partially observability in the system and no assumption being made regarding centralised aggregation of information. The Dec-POMDP is an extension of the the partially observable Markov decision process (POMDP) which in turn is the generalisation of the Markov decision process (MDP). The Dec-POMDP provides a natural framework for many real-world problems and its use has become ubiquitous (Nair et al. 2003; Oliehoek and Amato 2016; Cassandra 1998; Somani et al. 2013).

Methods using the MDP framework have been very successful in solving many machine learning problems (Beynier et al. 2013).However, one of the assumptions of the MDP is that the agent has access to the state of the environment with full certainty. In most real-world problems, however, agents have limited access to their environment or may not be able to observe the environment fully. This limiting assumption is avoided by the POMDP, which allows estimation of a policy which maps from the state beliefs to actions. This is in contrast to the original MDP policy that maps from known states to actions (Oliehoek et al. 2008).

In multi-agent systems we consider, however, problems must be tackled via coordinated collaboration across a set of agents. In several problems we consider the agents have differing sets of observations and further do not have access to observations from other agents - so precluding a centralised framework. Fortunately, decentralised frameworks provide the flexibility to avoid this impasse. In particular, the Dec-POMDP allows for effective decentralised decision making, as an extension to the POMDP model (Amato et al. 2013), making them well-suited for the problems we consider in this thesis, in which each agent may have a local (and unique) policy.

The Dec-POMDP environment is a partially observable setting, defined as a tuple $G = \{\mathcal{S}, \mathcal{U}, P, R, \Omega, O, \gamma\}$, with states $s_t \in \mathcal{S}$. There are $i = 1, \ldots, N$ agents who each choose actions, $u_t^i \in \mathcal{U}$ at each time step, in which Ω is the the set of observations for each agent, and $o_t^i \sim O(o|i, s_t)$ is each agent's (stochastic) observation function. At time t each agent has an action-observation history $\tau_t^i = \{o_0^i, u_0^i, r_0^i, \cdots, o_t^i\}$ and selects action u_i^t using stochastic policies of the form $\pi_\theta^i(u^i|\tau_t^i)$. The transition function, $P(s'|s, \mathbf{u})$, is conditioned on the joint action, $\mathbf{u}$.

A solution in the Dec-POMDP setting provides a set of policies for each agent in the environment, where each policy maps the agent's observations to actions, while the total global reward is maximised.

2.5 Reinforcement Learning

In reinforcement learning, agents interact with an environment sequentially, and through a transition function, where interaction with the environment (including perhaps other agents) is considered to be governed via a Markov Decision Process (MDP). Assuming s_t to be the state of the environment at time t and u_t an agent action at each time-step; the environment is characterised by the transition probability to the next state $p(s_{t+1} \mid s_t, u_t)$, action space U, the reward function $R(s_t, u_t)$, and discount factor $\gamma \in (0, 1)$.

We note that the Markov property implies that this transition is conditioned only on the current state and actions $s_{t+1} \sim p(s_{t+1} \mid s_t, u_t)$. We consider the agent's (optimal) action as determined by the agent's policy, π such that $\pi(u_t \mid s_t) = p(u_t \mid s_t)$. We note further that the agent's action space can be continuous or discrete. We consider discrete action spaces only for the problems discussed in this thesis, due to the nature of the applications considered.

The discount factor γ, specifies the significance of future rewards. The discount factor is required to guarantee convergence to a finite value for an infinite sum of rewards. Assuming $\tau = \{s_0, u_0, s_1, u_1, ...\}$ to be the trajectory; $R(\tau) = \sum_t \gamma^t r_t$,

where the per-step reward r_t, is available to the agent once the environment is transitioned from state s_t to the next state s_{t+1}. The agent(s) interacting with the environment aim to find a policy that maximises the total expected return $J(\pi) = \mathbb{E}_{\tau\sim\pi}R(\tau)$. Therefore, the optimisation problem in reinforcement learning can be formulated as follows:

$$\pi^* = \arg\max_{\pi} J(\pi) \tag{2.27}$$

where π^* is the optimal policy.

2.5.1 Value Functions

Value functions define a partial ordering over policies and therefore assist our search of the optimal policy, π^*. The term *value*, refers to the expected total return conditional on either the state or the state and the action.

The on-policy value function $V^\pi(s_t)$ is defined as the expected return for policy π, conditional on the current state s_t as follows (Achiam 2020):

$$V^\pi(s_t) = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau) \mid s_0 = s]. \tag{2.28}$$

The action-value function also known as the Q-function is defined as (Achiam 2020):

$$Q^\pi = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau) \mid s_0 = s, u_0 = u]. \tag{2.29}$$

The Q-function measures the expected return for taking action u_t in state s_t and following policy π afterwards.

The difference between the Q-function and the value function is known as the advantage function:

$$A^\pi(s_t, u_t) = Q^\pi(s_t, u_t) - V^\pi(s_t) \tag{2.30}$$

The advantage function measures the difference between the expected return for choosing action u_t, and the average expected return of all actions. Therefore,

it provides us with the relative advantage of an action instead of evaluating it in absolute terms.

In Section 2.5 we discuss model-free reinforcement learning algorithms that we use in this thesis for training value functions. We refer the reader to Sutton and Barto 2018 for further details and discussion on reinforcement leaning methods.

2.5.2 Q-Learning

In Q-learning we try to learn the optimal value for the Q-function in Equation 2.29, where the optimal value $Q^*(s,u) = \max_\pi Q^\pi(s,u)$ is inline with the Bellman optimality equation:

$$Q^*(s,u) = \mathbb{E}_{s'}[r + \gamma \max_{u'} Q^*(s',u') \mid s,u]. \tag{2.31}$$

In Deep Q-Network (DQN) models (Mnih et al. 2015), we learn the optimal Q-function using a deep neural network. The DQN model is optimised by minimising the bellman error at each iteration i as follows:

$$\mathscr{L}_i(\theta_i) = \mathbb{E}_{s,u,r,s'}[(r + \gamma \max_{u'} Q^*(s',u';\theta_i^- - Q(s,u;\theta_i))^2] \tag{2.32}$$

where θ is the vector of parameters for the network, and θ_i^- is the vector of parameters that are fixed and updated with θ_i every C steps. The method uses the ϵ-greedy policy to choose the action, where the chosen action maximises the Q-value with probability $1-\epsilon$ or is chosen randomly with probability ϵ to explore the environment. We use the DQN model to train the rainbow policy in Chapter 7.

2.5.3 Actor-Critic Models

Actor-critic methods (Barto et al. 1983; Widrow et al. 1973; Mnih et al. 2016; Konda and Tsitsiklis 2000), consist of an actor network and a critic network. The actor model is responsible for learning what action to take conditional on the state of the environment, and it follows a gradient that is dependent on, and in

the direction suggested by the critic model. The critic model is responsible for assessing the actions taken by the actor model.

A common choice for the critic model is defined as $R_t = Q(s_t, u_t) - b(s_t)$, where $b(s_t)$ is noted as the baseline and is intended for reducing variance (Weaver and Tao 2013). A popular choice for the baseline is the value function, where $b(s_t) = V(s_t)$. In this case R_t is equal to the advantage function $A(s_t, u_t)$. We use the actor-critic model for training some of our policies in Chapters 6 and 7.

2.6 Sampling and Optimisation Methods

In this section we provide a brief overview of the core inference and estimation methods used throughout the thesis for parameter evaluation and numerical estimation.

2.6.1 Markov Chain Monte Carlo Methods

A sequence of random variables $\{s_1, ..., s_n\}$ is a *Markov chain*, if the conditional distribution of s_{n+1}, given the sequence, depends on s_n only. Therefore,

$$p(s_{n+1} \mid s_n, ..., s_1) = p(s_{n+1} \mid s_n). \tag{2.33}$$

Furthermore, the Markov chain is stationary if the transition probabilities $p(s_{n+1} \mid s_n)$ are not dependent on n, for all possible values of s_i in the state space of the Markov chain.

Markov Chain Monte Carlo (MCMC) methods allows for simulation of a stochastic process by simulating dependent realisations that form an irreducible Markov chain that has some desirable distribution, $p(s)$ say, as its stationary form (Geyer 1992). The method normally requires extensive samples to be taken, due to dependence between the samples. The samples from the final chain may be used as a proxy for samples from the posterior distribution. Typically we will use these samples to perform numerical evaluation of expectations via integration,

$\mu = \int g(s)dp(s)$, where g is a real valued function on the state space. The integral is approximated by averaging over the chain $\hat{\mu}_n = \frac{1}{n}\sum_{i=1}^{n} g(s_i)$.

There are numerous approaches to effective sampling using MCMC methods. Arguably, however, one of the simplest, yet most popular MCMC algorithms, is the Metropolis-Hastings (MH) algorithm (Metropolis et al. 1953; Hastings 1970). In the MH algorithm we evaluate $\tilde{p}(s)$, a non-normalised $p(s)$ (i.e. there is no guarantee that it integrates to unity). We sample some s^* from a proposal distribution $q(s^* \mid s^\tau)$ that depends only on the current sample state s^τ. Under the MH approach, the sample s^* is accepted with probability:

$$A(s^*, s^\tau) = \min\left(1, \frac{\tilde{p}(s^*)q(s^\tau \mid s^*)}{\tilde{p}(s^\tau)q(s^* \mid s^\tau)}\right), \tag{2.34}$$

where A is the probability of acceptance. If s^* is accepted then $s^{\tau+1} = s^*$, otherwise $s^{\tau+1} = s^\tau$ and another sample is drawn from the proposal distribution (Bishop 2006).

Gibbs' sampling (Geman and Geman 1984) is an MCMC method and can be seen as a special case of the MH algorithm, in which the order of sampling allows for an acceptance probability of unity at each step - thus aiding the efficiency of the sampling process. Gibbs', and its extensions, are used in Part II of this thesis. We discuss the method in more detail in Chapter 6.

2.6.2 Monte Carlo Dropout

Monte Carlo dropout is a recent technique that is extensively used to approximate posterior distributions in (approximate) Bayesian neural networks. The method was introduced by Gal and Ghahramani 2016, who show that, by drawing Bernoulli-distributed random variables that correspond to the dropout mask which is applied to weights in the network, we may sample over models with multiple network connectivities and configurations. It may be shown that these samples asymptotically approach variational Bayes' posterior approximation and thus may be used to evaluate uncertainty as well as measures such as posterior expectations.

We discuss the method further in Chapter 4, in which we develop an alternate approach to dropout to estimate the posterior predictive distribution, comparing our method with dropout.

2.6.3 Gradient Descent Optimization

Gradient Descent is one of the most commonly used algorithms for optimisation, in particular for optimising deep neural networks. The method minimises an objective function $J(\theta)$, using the gradient of the objective function with respect to the parameters θ. Denoting the gradient with respect to the parameters as $\nabla J(\theta)$, simple gradient descent alters the parameter values along the direction of (negative) gradient according to:

$$\theta_{\tau+1} = \theta_\tau - \eta \frac{1}{T} \sum_{t=1}^{T} \nabla_\theta J(\theta_\tau; x_t, y_t), \quad \eta \in \mathbb{R}^+ \tag{2.35}$$

in which η represents a (scalar) learning rate. As the gradient is computed over all data points, this simple approach can be computationally cumbersome for large data sets.

Stochastic gradient decent improves the tractability of simple gradient methods, where the updates are performed associated with a selected data point, (x_t, y_t), as follows:

$$\theta_{\tau+1} = \theta_\tau - \eta(\nabla_\theta J(\theta_\tau; x_t, y_t)), \quad \eta \in \mathbb{R}^+ \wedge t \in \{1, ..., T\}. \tag{2.36}$$

The algorithm can thus be implemented on a subset of the data. However, when the subsets are small there is high variability in gradient estimates and thus between the parameters updates, leading to volatile fluctuations in the cost, error or reward function. This leads to poor convergence to a (local) minimum. Decreasing the learning rate η can to some extend reduce these fluctuations and improve convergence, at the cost of longer optimisation runs.

To avoid such issues, mini-batch stochastic gradient decent updates the gradient estimate using a mini-batch of S randomly selected training points as follows:

$$\theta_{\tau+1} = \theta_\tau - \eta \frac{1}{S} \sum_{s=1}^{S} \nabla_\theta J(\theta_\tau; x_s, y_s), \quad \eta \in \mathbb{R}^+ \wedge s \in \{1, ..., S\} \wedge \{s\} \subset \{t\}. \quad (2.37)$$

This is found to decrease the cost function volatility during optimisation yet retaining computational efficiency compared to standard (full data) methods. However, the choice of a learning rate, η is critical to the success of the approach. Large values for η can lead to poor solution convergence, while small values lead to achingly slow optimisation. To alleviate these issues, adaptive learning rate methods have been widely adopted, the most notable of which is Adaptive Gradient (AdaGrad) algorithm (Duchi et al. 2011). Recent developments has seen improvements due to the addition of momentum terms to parameter updates (Zou et al. 2018).

Adaptive Moment Estimation Algorithm

Although simple gradient-based optimisation can be effective, it has well-known failures when the gradients, associated with different directions on the loss surface, have magnitude differences. A *scalar* learning, or adaption, rate parameter is thus ineffective. Of course, second-order approaches, such as quasi-Newton methods, avoid this problem by re-scaling (and re-orientating) the gradient operator using the *Hessian* matrix. Evaluating the latter is prohibitive for all but the smallest problems, however. To provide an improvement over scaler learning rate solutions, yet retain computational tractability, the Adaptive Moment Estimation (Adam) algorithm (Kingma and Ba 2015) computes an adaptive learning rate in the direction of each of the parameters and also uses past gradient values, similar to momentum algorithms. It can thus be thought of as a diagonalised Hessian method. Assuming $g_{\tau+1} = \nabla_\theta J(\theta_\tau)$; the exponentially decaying mean of the

gradient $m_{\tau+1}$, and the exponentially decaying mean of the squared gradient $\nu_{\tau+1}$ (second moment), are computed as follows:

$$m_{\tau+1} = \beta_1 m_\tau + (1-\beta_1) g_\tau \tag{2.38}$$

$$\nu_{\tau+1} = \beta_2 \nu_\tau + (1-\beta_2) g_\tau^2 \tag{2.39}$$

where β_1 and β_2 are the exponential decay rates for the moment estimates. However the moment estimations in Equations 2.38 and 2.39 are biased towards zero and therefore the biased corrected estimations are as follows:

$$\hat{m_{\tau+1}} = \frac{m_\tau}{1-\beta_1^\tau}, \quad 1-\beta_1^\tau = (1-\beta_1)\sum_{i=1}^{\tau} \beta_1^{\tau-i} \tag{2.40}$$

$$\hat{\nu_{\tau+1}} = \frac{\nu_\tau}{1-\beta_2^\tau}, \quad 1-\beta_2^\tau = (1-\beta_2)\sum_{i=1}^{\tau} \beta_2^{\tau-i}. \tag{2.41}$$

The parameters are then updated such that,

$$\theta_{\tau+1} = \theta_\tau - \eta \frac{\hat{m_{\tau+1}}}{\sqrt{\hat{\nu_{\tau+1}}} + \epsilon} \tag{2.42}$$

The algorithm's hyperparameters, $\{\nu, \beta_1, \beta_2, \epsilon\}$, can be optimised for best performance. Throughout this work, we have chosen the Adam algorithm as our method of choice for all optimisations.

2.6.4 Reinforcement Learning Policy Optimisation

In this section we briefly cover the underpinning approaches for policy optimisation in reinforcement learning. As the literature is extraordinarily large, we focus on canonical methods only, particularly those approaches which are explicitly or implicitly used later in this thesis.

Vanilla Policy Gradient Method

We note initially that, although many machine learning methods define a *loss* or error function, reinforcement learning (and policy learning in general) tends to prefer the maximisation of a *reward or value* function. In particular rewards

are often defined as a reward associated with a (time) index, R_t say, which we aggregate (summing, or taking expectations) under some parameterised policy, π_θ say. Defining $J(\pi_\theta) = \mathbb{E}(R_t)$ as the aggregate value function, we may use standard gradient methods (in this case gradient *ascent*) to iteratively adjust the parameters of the policy model. This leads to the Vanilla Policy Gradient method (Sutton et al. 2000):

$$\theta_{\tau+1} = \theta_\tau + \eta \nabla_\theta J(\pi_{\theta_\tau}) \tag{2.43}$$

where $\eta \in \mathbb{R}_+$ is the scalar learning rate.

The method is *on-policy*, meaning that exploration in the policy/value space is achieved by sampling actions based on the latest version of the policy.

Proximal Policy Optimisation Method

In line with many simple gradient-based optimisation methods, Vanilla Policy Gradient suffers from high gradient variance, leading to volatile changes in parameters and consequent poor convergence properties. As discussed in Section 2.5.3, removing a baseline measure from R_t, such as using an "advantage function", reduces this variance.

Trust region policy optimisation (TRPO) method (Schulman et al. 2015), decreases the variance even further by ensuring the updated policy is not too dissimilar to the old policy. This is achieved by measuring and limiting the change in the policy for each update using the KL-divergence. However, the method is complicated and not compatible with certain network structures such as using dropout or parameter sharing. Proximal Policy Optimization (PPO) (Schulman et al. 2017) method uses trust region for searching the global optima, similar to TRPO, leading to more stability and better performance. Same as TRPO, in order to achieve this, the method uses multiple epochs of stochastic gradient ascent to perform a policy update. However, PPO method has a much easier overall implementation compared to TRPO.

The TRPO method has a constraint objective function as follows:

$$\max_{\theta} \hat{\mathbb{E}}_t[r_t(\theta)\hat{A}_t] \tag{2.44}$$

$$\text{subject to } \hat{\mathbb{E}}_t[\mathrm{KL}[\pi_{\theta_{\text{old}}}(. \mid s_t), \pi_\theta(. \mid s_t)]] \leq \delta \tag{2.45}$$

where $\hat{A}_t$ is an estimator for the advantage function; θ_{old} is the vector of policy parameters before they are updated; and $r_t(\theta)$ is the probability ratio such that

$$r_t(\theta) = \frac{\pi_\theta(u_t \mid s_t)}{\pi_{\theta_{\text{old}}}(u_t \mid s_t)}. \tag{2.46}$$

Without the constraint, TRPO would lead to an excessively large policy update. This constraint, however, could be replaced with a penalty. In PPO, the objective and constraint from the TRPO are replaced with the following objective:

$$\max_{\theta} \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{2.47}$$

where ϵ is a hyperparameter. The rationale behind this objective function is that the second term inside the minimum function in Equation 2.47 clips the probability ratio, and taking the minimum of the clipped and unclipped objectives will lead to a lower bound (Schulman et al. 2017). The TRPO and PPO objectives are the same to first order around θ. Therefore, the PPO method produces the same reliability as TRPO but with much easier implementation. The PPO method has received significant attention since its introduction in 2017. We use the method to train some of the policies in Sections 7.4.1 and 7.4.2 in Chapter 7.

2.6.5 Genetic Algorithms

Although only a single instance of genetic algorithm optimisation is performed in the work of the thesis, we present a (very) brief overview here.

Genetic algorithms were originally inspired by evolution and natural selection as a method of adapting, selecting and merging "genomes" of parameters so as to extremise a loss (or reward) function. Solutions in these approaches are encoded by parameter string structures that mimic chromosomes. Parameter

strings which lead to higher-value solutions are allocated a higher reproductive opportunity (Mirjalili 2019).

Initialisation occurs via initial string generation (often achieved randomly), subsequent to which each string of parameters is evaluated and assigned a fitness value, defined as $\frac{f_i}{\bar{f}}$, where f_i is the evaluation score assigned to individual i and $\bar{f}$ is the average of all evaluations. Genetic algorithms differ in the details of how populations of strings are propagated, sliced and altered - though the basis concepts are common to them all.

In the *canonical* genetic algorithm each population member is defined via a binary string, and population propagation is achieved via the tournament selection sampling method (Goldberg and Deb 1991). In this approach strings are randomly sampled from the population and selected stochastically depending on their fitness. This selection process continues until an intermediary population is filled. The process is repeated across multiple generations, either for a fixed generation number or until the fitness function changes fall consistently below a pre-defined threshold. We use tournament selection to train one of the policies used in Section 7.4.1 in Chapter 7.

Part I

Modelling Multimodality in Time Series

Part I Summary

In this part we focus on the problem of multimodality in time series prediction. We use GMM models to estimate a multimodal posterior for the models discussed in this part.

In Chapter 3, we present *Mixture Density Gaussian Processes* (MidGaPs) allowing multi-modal posterior distributions from GPs. We make explicit comparison with alternate models, namely the Mixture Density Network (MDN) model, Mixture of Experts (ME) model and standard GP models. Unlike MDN approaches, we allow full probability distributions over the latent variables that encode the mixture posterior, allowing uncertainty to propagate in a principled manner. Unlike the ME models such as Mixture of Gaussian Processes methods, we achieve non-Gaussian posteriors within a single GP model. We showcase the performance of the approach on synthetic and real time series data sets. Our results indicate that not only is the approach competitive in terms of error metrics but also provides further insight into the multiplicity of potential paths a time series may take in the future.

In Chapter 4, we introduce deep mixture density networks (Deep-MDN) as a computationally efficient framework to estimate even complex posterior distributions. We compare our method to standard DL approaches and the widely used Bayesian Deep Learning (BDL) method of Gal and Ghahramani 2016, where *dropout* is used to approximate the posterior distribution. We show that our approach has significantly faster convergence in comparison and is computationally more efficient. Further, increasing the number of mixture coefficients in the Deep-MDN not only improves the posterior estimation, but further increases the rate of convergence.

In Chapter 5, we present the *Mixture Density Conditional Generative Adversarial Model* (MD-CGAN), with a focus on time series forecasting. We show that our model is capable of estimating a probabilistic posterior distribution over forecasts and that, in comparison to a set of benchmark methods, the MD-CGAN model

performs well, particularly in situations where noise is a significant component of the observed time series. Further, by using a Gaussian mixture model as the output distribution, MD-CGAN offers posterior predictions that are non-Gaussian.

3

MiDGaP: Mixture Density Gaussian Processes

Contents

3.1 Introduction

Most prediction models with uncertainty focus on inferring a unimodal posterior distribution, often assumed to be Gaussian with a mean estimating the conditional average of the target data conditioned on the input. Although this provides valuable models in a range of problem domains, it fails to entertain the notion

that the distribution over predictions, or missing data, might be *multimodal*. One of the most notable and core techniques for prediction with uncertainty with numerous extensions and applications in machine learning, are the Gaussian processes (GPs). Although several approaches exist for warping the conditional Gaussian posterior distribution to other members of the exponential family, most tacitly assume a unimodal posterior.

In this chapter we consider the posterior distribution to be modelled as a (finite) mixture of Gaussians, allowing for a rich variety of possible posterior forms, including multimodality, asymmetric and non-Gaussian, all of which may be approximated by the appropriate mixture model. We note that *Mixture Density Networks* (MDNs) have been long known in the literature, since their introduction in Bishop 1995. We extend the approach by placing a Gaussian Process (GP) over the latent variable set which encodes the final posterior distribution, thus allowing full measures of uncertainty to propagate. Furthermore, we allow the model to be dynamic enabling us to look at changes in the posterior over time, identifying regions in a data stream where multiple possible futures are likely. We apply our approach to synthetic data to show its functioning as well as showcasing its operation on several real-world datasets.

3.2 Related Work

There have been many attempts to estimate the GMM models using various frameworks. Bishop 1995 presented MDN, which is a static GMM for the conditional probability distribution $p(\mathbf{y}|\mathbf{x})$. The parameters of the mixture model are taken to be transformations of a set of latent variables $\mathbf{z}^{\alpha}$, $\mathbf{z}^{\sigma}$, and $\mathbf{z}^{\mu}$, conditioned on $\mathbf{x}$. To estimate the latent variables $\mathbf{z}$, Bishop uses a feed-forward one-layer neural network, where the hidden layer is a set of neurones with activations governed by a sigmoid function. Once the neural network is trained, he uses the weights of the trained network to further evaluate the model

on static multimodal test datasets. Uncertainty is therefore not incorporated in the estimation of the parameters.

Jacobs et al. 1991 present the Mixture of Experts (ME) model that breaks down the domain and learning procedure is composed of many separate expert networks. This framework is further discussed in Section 2.3.2. The ME model has been an inspiration for many other frameworks. Tresp 2001 introduces mixture of Gaussian processes (MGP) model which is derived from the ME model. The model breaks down the domain similar to the ME model but replaces the networks with a GP and produces M GPs. Depending on the region of the input the MGP decides which GP is appropriate to use. The MGP model has soft boarders and there are some overlap for the points close to the domain border.

Rasmussen and Ghahramani 2002 present an extention of the MGP model, the Infinite Mixture of Gaussian Process Experts (IMGPE), where unlike the MGP, the number of GP experts is not predetermined. Further in contrast to the MGP model, the experts have strict domains and there is no spillover.

Meeds and Osindero 2006 extend on the IMGPE model by using a generative model over the input and output space rather than just a conditional model, which has an advantage of dealing with incomplete or missing data as well as allowing for inverse functional mappings. In their model the experts are comprised of two parts, a density over the input space, specifying the distribution of the domain points associated with the expert, and a GP model over the outputs associated with that expert.

Shi et al. 2005 present a Hierarchical Gaussian Process mixture model. The main difference of their model to the IMGPE is that unlike IMGPE they assume that each batch of observations belongs to one GP, and it is the heterogeneity amongst the different batches which creates the mixture of GPs. The identity of each GP is however missing, which gives rise to the hierarchical structure, where a Bayesian approach is used for analysing the hierarchical model. Their model has parallels with the random-effect type approach. The data they consider is

gathered from individuals, and the assumption is that all the data collected from each individual belongs to one GP, and the GPs can be broken down based on heterogeneity amongst the individuals, e.g. age, weight, height, etc.

Sun 2013 extends the IMGPE model to multivariate GPs. The model has the advantage of modelling multimodal data, and also computationally alleviates the complexity of the multivariate GPs. They compare the results for a synthetically created two dimensional data with a multi-task learning neural network as a benchmark, and their multivariate GP model outperforms this benchmark.

Our models in this Chapter are different to these methods. With the exception of the the MDN model, all the other approaches look to decompose the input domain into a set of regions with either soft or strict borders. The MiDGaP models we present in this Chapter, however, aim to infer a multimodal posterior distribution, extending the range of posterior models that the Gaussian Process framework can contend with. We do not break down the input domain and as such the posterior distribution is valid for the whole domain.

3.3 Model Framework for MiDGaP and RBFN Models

We present MiDGaP and RBFN models in this section. There are two adaptations of the MiDGaP model, MiDGaP-I and MiDGaP-II that is a two-step version of MiDGaP-I.

Most generally, we consider a time series $\mathbf{y_t}$ and aim to make a prediction for $\mathbf{y_t}$ using the input variable set $\mathbf{x_t}$ (in our experiments we use the lagged output variable as our input). In order to make the prediction we model the conditional density $p(\mathbf{y_t}|\mathbf{x_t})$.

We start with a model using m mixture components, similar to that presented by Bishop 1995:

$$p(\mathbf{y_t}|\mathbf{x_t}) = \sum_{i=1}^{m} \alpha_i(\mathbf{x_t})\phi_i(\mathbf{y_t}|\mathbf{x_t}), \tag{3.1}$$

in which

$$\phi_i(\mathbf{y_t}|\mathbf{x_t}) = \frac{1}{(2\pi)^{\frac{1}{2}}\sigma_i(\mathbf{x_t})} \exp\left\{-\frac{\|\mathbf{y_t} - \mu_i(\mathbf{x_t})\|^2}{2\sigma_i^2(\mathbf{x_t})}\right\}. \tag{3.2}$$

Noting that the mixture model $p(\mathbf{y_t}|\mathbf{x_t})$ in equation 3.1 offers extreme flexibility in modelling the posterior distribution over $\mathbf{y_t}$, and has a generic property of modelling arbitrary density functions.

We assume the parameters of the mixture model, the means $\mu_i(x)$, the mixing coefficients $\alpha_i(x)$ and the variances of each mixture, $\sigma_i(x)$, to be continuous functions of $\mathbf{x_t}$. The parameters of the mixture model are taken to be transformations of a set of latent variables $\mathbf{z}^\alpha, \mathbf{z}^\sigma, \mathbf{z}^\mu$, conditioned on $\mathbf{x_t}$. We make the following assumptions our model parameters conditioned on the latent variables (we note that these transformations are also common in many probabilistic programming languages). The mixture coefficients, α_i, must satisfy $\sum_{i=1}^m \alpha_i(\mathbf{x_t}) = 1$. To enforce this we use the softmax transform from the latent variables:

$$\alpha_i = \frac{\exp(\mathbf{z}_i^\alpha)}{\sum_{j=1}^m \exp(\mathbf{z}_j^\alpha)}. \tag{3.3}$$

The standard deviations have to be non-zero and positive and therefore we adopt the following transform, conditioning on latent variables:

$$\sigma_i = \exp(\mathbf{z}_i^\sigma). \tag{3.4}$$

Finally for the mean priors we use the below:

$$\mu_i = \mathbf{z}_i^\mu. \tag{3.5}$$

The data likelihood function is

$$L = \prod_{t=1}^n p(\mathbf{y_t}, \mathbf{x_t}) = \prod_{t=1}^n p(\mathbf{y_t}|\mathbf{x_t})p(\mathbf{x_t}). \tag{3.6}$$

Noting that the error function is simply the negative-log of the likelihood provides an error for each data point (dropping the final term $-\log p(\mathbf{x_t})$):

$$E_t = -\ln\left\{\sum_{i=1}^m \alpha_i(\mathbf{x_t})\phi_i(\mathbf{y_t}|\mathbf{x_t})\right\} \tag{3.7}$$

and total error is the sum of all errors.

The MiDGaP models that we present here estimate the latent variables $\mathbf{z}$, using GPs. Therefore $\mathbf{z} = \mathcal{GP}(\mathbf{x_t})$. For these models we use a GP with an RBF kernel and hyperparameters $\theta = \{\sigma_\mathbf{n}, \sigma_\mathbf{f}, \mathbf{L}\}$, where σ_n is the noise standard deviation, σ_f is the output scale, and L is the lengthscale hyperparameter of the kernel.

3.3.1 Mixture Density Gaussian Process Model I

As we lack direct observations for the latent variables $\mathbf{z}$, we augment the set of unknown parameters in the model with $\mathbf{z}$, which are successively re-inferred to maximise the (log) marginal likelihood of the data conditioned on recent observations. Once inference has taken place, at each step sequentially, we can make a prediction for the successor latent values and infer Equations 3.1 and 3.2.

It is worth noting that for both MiDGaP models, we have a distribution for each predicted hidden variable, $\mathbf{z}$. While all estimated hidden variables have a Gaussian distribution, their transforms will no longer have a Gaussian distribution (with the exception of the hidden variables of the means of the component distributions in the GMM, the $\mathbf{z}^\mu$s). It is further analytically cumbersome to estimate the distribution of the transforms, and therefore we use sampling to evaluate $p(\mathbf{y_t}|\mathbf{x_t})$. Each set of sample points provides an estimation for $p(\mathbf{y_t}|\mathbf{x_t})$ at time t, noted as $p^s(\mathbf{y_t}|\mathbf{x_t})$, and an error term, E_t^s. The error for each point is therefore estimated as $E_t = \mathbb{E}(E_t^s)$. Figure 3.1 illustrates the sampling method.

3.3.2 Mixture Density Gaussian Process Model II

In this adaptation of the method in order to estimate the posterior conditional probability distribution, $p(\mathbf{y_t}|\mathbf{x_t})$, we follow two steps. The first step is to estimate the mixture density model parameters using the training data. This is achieved with a single layer feed-forward neural network in a similar framework to Mixture Density Network (MDN) models in Bishop 1995 but with a different structure. For this step the hidden variables $\mathbf{z}$, are the output of a neural network, that takes $\mathbf{x_t}$ as an input.

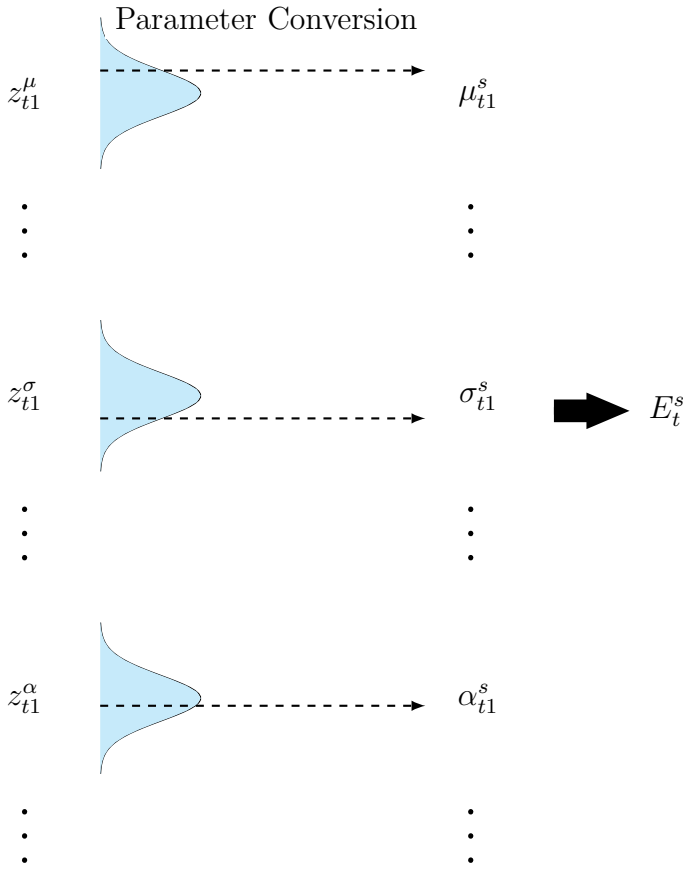

Figure 3.1. To estimate the average error for the MiDGaP model, we sample from the hidden variables, evaluated by the GP function. Each set of sample points, $\mathbf{s} = \{s^{\mu}_1, ..., s^{\mu}_m, s^{\sigma}_1, ..., s^{\sigma}_m, s^{\alpha}_1, ..., s^{\alpha}_m\}$, drawn from a set of hidden variables $\mathbf{z} = \{z^{\mu}_1, ..., z^{\mu}_m, z^{\sigma}_1, ..., z^{\sigma}_m, z^{\alpha}_1, ..., z^{\alpha}_m\}$ provides an error estimation point, E^{s}_{t}.

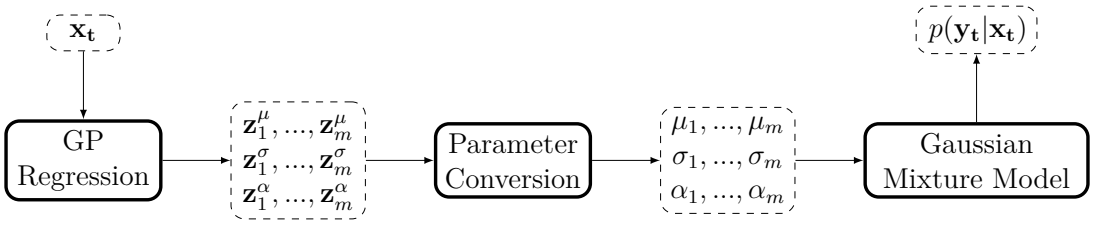

Figure 3.2. *Schematic of the second step of the MiDGaP-II model. A GP regression is fitted where* $\mathbf{z} = \mathcal{GP}(\mathbf{x_t})$*. Parameter Conversion is further achieved through sampling.*

For the hidden layer, we use a radial basis function network (RBFN) described in Section 3.3.3. The error function is then minimised by a suitable optimisation method (we use the Adam optimisation algorithm). Once this is completed we have a best estimate of the hidden variables $\mathbf{z}$. This completes step one.

The second step is to predict the posterior probability density. We perform a GP regression on the hidden $\mathbf{z}$ variables, using a factored covariance kernel. For ease of comparison across the methods our kernel choice is also a radial basis function. We then transform the predicted $\mathbf{z}$ values to estimate the mixture coefficients α_i, means μ_i and variances σ_i, and this enables us to estimate the posterior distribution $p(\mathbf{y_t}|\mathbf{x_t})$. Figures 3.2 illustrates the schematic of the second step.

3.3.3 Radial Basis Function Network Model

For the RBFN model we make the same assumptions presented in Equations 3.1 to 3.7 earlier in this Section. For this model for the estimation of the parameters in the mixture model we use a feed-forward neural network with a single hidden layer. For the hidden layer, we use a radial basis function network as follows:

$$\gamma_l(\mathbf{x_t}) = \exp\left\{-\frac{1}{2}(\mathbf{x_t} - \nu_l)^T(\sigma_l^2 I)^{-1}(\mathbf{x_t} - \nu_l)\right\} \tag{3.8}$$

$$\mathbf{z}(\mathbf{x_t}) = \mathbf{w}\gamma \tag{3.9}$$

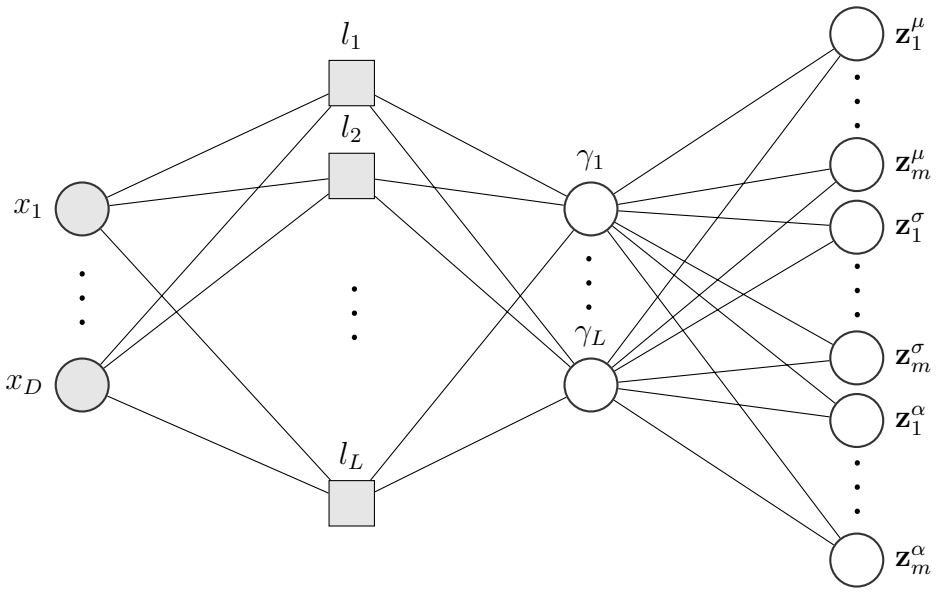

Figure 3.3. The structure of the RBFN network.

where L is the number of basis functions ($l \in \{1, .., L\}$), γ_l is the localised Gaussian basis function with mean ν_l and variance σ_l, and $\mathbf{w}$ is the weight matrix. Figure 3.3 shows the structure of the RBFN network.

We use this model as a first step in the MiDGaP-II model, and also for model-fitting as well as estimating the posterior distribution of $p(\mathbf{y}|\mathbf{x})$ in the RBFN model in Table 3.1 in Section 3.4.

In Section 3.4, we compare the MiDGaP models to the RBFN model, a standard GP model, and the "fast Bayesian Mixture Experts" (fBME) model, which is an ME model presented in Bo et al. 2008.

3.4 Experiments

As a first step we test the MiDGaP models on synthetic data, where we know the solution. Following this we test the model on two real time series, the sunspot data (Clette 2015), and the Mackey-Glass equation dataset (Mackey and Glass 1977). Sunspots are the regions on the surface of the sun with lower temperature that appear darker than the surrounding regions. The sunspot dataset is the average

of these dark spots on a monthly basis. The Mackey-Glass equations are a class of a delay-differential equations often used in biology. For a high levels of delay the equations result in a chaotic solution. We further make a comparison of our results with the predictions estimated by standard GP, fBME, and RBFN models. For each time series we use the lagged data as input $\mathbf{x_t}$. The lag is set to 10 for all time series and we predict the next 10 data points at each time step with a rolling window. For all time series the number of mixture coefficients m, is set to 3.

For the RBFN model, for all datasets, L is set to 20, and σ_l is fixed at 0.1. For the MiDGaP models the number of generated sample points is 1000. It is worth noting that whilst the parameters of the RBFN hidden layer can be optimised, our results have a low sensitivity to the chosen levels. Further, the number of sample points is set to be sufficiently large in order to achieve stable results. The experiments are performed using the Python programming language.

3.4.1 Synthetic Dataset

We construct a simple synthetic data, which provides regions of multi-model output density, as follows:

$$\mathbf{y_t} = \begin{cases} 0.3 + \epsilon, & p(y_t = 0.3) = \alpha_t \\ 0.7 + \epsilon, & p(y_t = 0.7) = 1 \text{ - } \alpha_t \end{cases} \tag{3.10}$$

where $\epsilon \sim \mathcal{N}(0, 0.01)$. The value of α_t changes gradually from 1.0 to 0.0 in a time span of 200 data points. We concatenate $\mathbf{y_t} = 0.5 + \epsilon$ for 220 time steps with the above multimodal time series, to create the synthetic dataset which is initially unimodal in the target variable then evolves multimodality.

3.4.2 Results

Figure 3.4 illustrates the predicted posterior distribution for the synthetic time series by each model. The heatmap in each subfigure represents the value of the posterior density at each time-step. We note that MiDGaP models (right panels) predicts the multimodal distribution most accurately, that the RBF model

performs fairly well on this data and the standard GP (as a reference) is expected to perform poorly. The mixture of experts approach (fBME) does not manage to capture the underlying duality of the data in this instance.

Over the synthetic dataset, as well as two candidate real-world datasets (the classic sunspot data time series, as well as a section of the Mackey-Glass chaotic time series, each consisting of 400 points) our approach that we present here performs well, as measured using the negative log-likelihood of future data conditioned on the model. All methods capable of multimodel predictive posteriors are expected to outperform a standard GP on data where multiple future trajectories are possible. Table 3.1 illustrates the average errors for all models. We note that our approach does considerably better than the other models on the synthetic data, better than all models on the Mackey-Glass time series and similarly to all models on the sunspot data. This is expected as the synthetic dataset is, by design, the most multimodal out of the three datasets.

Table 3.1. Negative log-likelihood on out-of-sample future data. The lowest error model is shown in bold in each case.

	Average Errors				
Time series	Standard GP	fBME	RBFN	MiDGaP-I	MiDGaP-II
Synthetic data	10.49	17.60	-0.68	**-1.36**	-1.21
Sunspot dataset	-1.07	**-1.26**	-1.08	-1.10	-1.15
Mackey-Glass	0.06	0.11	0.78	**0.04**	0.76

3.5 Discussion

In this Chapter we present a dynamic model that allows a non-Gaussian, multimodal posterior distribution over forecasts, via a finite mixture of Gaussians. The parameters of the mixture model are themselves uncertain variables, whose posterior variability we propagate upwards into the mixture model itself. This

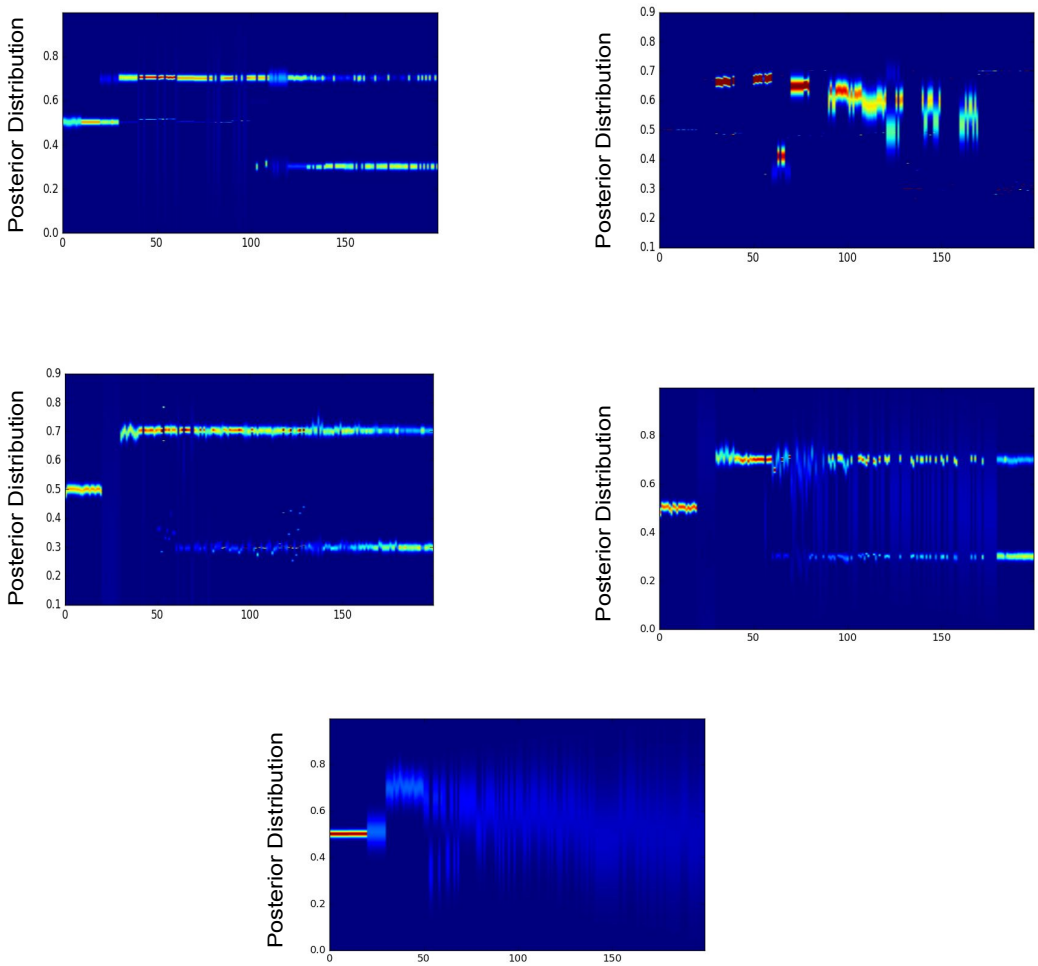

Figure 3.4. Synthetic data: Comparison of predicted posterior distribution by model. Top left - MiDGaP-I mixture density model. Top right - Mixture of experts. Middle left - MiDGaP-II mixture density model. Middle right - RBFN mixture density model. Bottom - Standard Gaussian Process.

provides an additional level of uncertainty inference, where the model provides confidence levels around the shape of the multimodal distribution as well.

In contrast to a standard GP regression which is restricted in its prediction to a unimodal Gaussian distribution, our predicted distribution is multimodal and its shape is not restricted. We applied our approach to synthetic and real problem sets, showing that there are clear regions in which multiple future outcomes

are postulated.

We note however that the MiDGaP models have some limitations which resemble that of a GP model. In particular, when there are a large number of data points, estimating a large covariance function could be computationally cumbersome, similar to GPs. This is an important shortcoming of the GP models and their extensions, that makes them less desirable in many practical applications. To overcome computational inefficiency, in Chapter 4 we discuss the Deep-MDN model, where we replace the GP method with deep neural networks as an alternative method for estimating the GMM's parameters.

3.6 Conclusion

We present a Gaussian Process mixture density model, capable of inferring flexible multimodal posterior distributions. We consider the performance of the model on three candidate datasets, one synthetic and two real. In two of the three cases our approach significantly outperforms alternate models, including a standard GP. We see our method as useful in not only offering multimodal forecasts, but also in monitoring the complexity changes in such forecasts, which can be valuable in determining state changes and tipping points in complex systems.

4
Deep Mixture Density Networks

Contents

4.1 Introduction

In many application domains it is vital to produce well calibrated posterior uncertainty estimates, even if full Bayesian inference is not performed on a model. To this end, techniques such as *Monte Carlo dropout* (Gal and Ghahramani

2016), have proved effective and popular. In this Chapter we develop deep learning models with mixture densities as the output, as an alternative approach to providing posterior distributions. We show that the approach we take is significantly more efficient compared to dropout-based methods and highlight its performance advantages over Bayesian neural networks and standard deep learning models. The standard deep learning models that we use in this chapter are the convolutional neural network (CNN) and long short-term memory (LSTM) network. We discuss these in further details in Sections 4.3.1 and 4.3.2. We compare our model to these and the Bayesian neural network models that have CNN and LSTM structures.

4.2 Background

Neural networks models have been around for many decades since 1940s, however they have received the most attention over the last decade. The initial interest in the field came from mathematical biology and were designed to imitate biological learning. One of the notable works in this period was the introduction of perceptron models in Rosenblatt 1958 and Rosenblatt 1961. These linear perceptron models could learn the weight values and define categories. While neuroscience remains an inspiration to the field, it is no longer the dominant driver (Goodfellow et al. 2016).

The backpropagation was formally introduced by Rumelhart et al. 1985 in the neural networks literature and this greatly improved computing of the gradients in deeper layers. This renewed interest in the field in the 80s and the 90s. The principal driver of research for this era was distributed representations (Hinton 1984); the concept of connecting small computational units in order to solve large complex problems.

The latest and the current wave was initiated in 2006 following works such as Hinton and Salakhutdinov 2006; Bengio et al. 2007. These were great breakthroughs which significantly improved the training efficiency of neural networks and prompted training of deeper and larger networks (Goodfellow et al. 2016).

Research in probabilistic approaches applied to deep learning was, for the most part, initiated in the early 1990s. One of the most notable of the papers from this era was the work of MacKay 1992 showcasing the use of Bayesian inference for neural networks, in many ways initiating the field of Bayesian neural network (BNN) research which is undergoing a revival at present. Indeed, in recent years BNN models have undergone significant progress, both in their practical tractability and in methodology, allowing their use in many real-world applications. We discuss BNNs further in Section 4.4.

4.3 Standard Deep Learning Neural Networks

In this section we present two popular structures for deep learning, the CNN and LSTM. In their standard form, these models make point estimations, which we use as reference deep learning methods. We extend both approaches by using techniques from Bayesian neural networks and Deep Mixture Density Network (Deep-MDN) models, which we present in Section 4.5.

4.3.1 Convolutional Neural Network Models

Convolutional neural network models, commonly known as CNNs (LeCun 1989), are a type of a neural network structure whereby the input layer of the network operates on data with a grid form topology, often allowing the network to work from raw data, such as images (Goodfellow et al. 2016). CNNs have had tremendous success across many application domains, particularly those whose data topology fits well with the implicit assumption of gridded data, such as images and many time-series. At the heart of the methodology lies a standard convolution operation with (a set of) convolutional filters whose parameters (in effect the filter coefficients) are learning along with all other parameters in the network, typically via supervised training to extremise a final objective function. Across a discrete set of data, $x(t)$

say, convolution with some convolution filter $g(a)$, defined over an index set a, is:

$$x * g = \sum_{t'=t-a}^{t} x(t')g(t-t') \tag{4.1}$$

CNNs use a convolution operation, instead of matrix multiplication, in at least one layer of the network and typically in multiple layers throughout a deep architecture. CNNs offer an advantageous choice for a deep learning structure. Convolutional layers impose sparse connectivity between neurons in adjacent layers by making the kernel smaller than the input. This improves the memory requirements as well as computational efficiency. Further convolution operations in CNNs use parameter sharing, which accelerates learning given that the weights are shared for parts of the network. Finally, the convolution operation is invariant to translation of the input, which allows, for example, for a certain level of resilience to object location in applications such as image analysis.

The convolutional operation within a CNN network typically has three stages (Figure 4.1). In the first layer a number of convolution transformations are performed (these can occur in parallel, allowing for e.g. GPU speedups). This results in a set of filtered outputs. In the second stage a non-linear activation function is employed, conditioned on these outputs. The prevalent nonlinear activation function used for this stage is the rectified linear unit, commonly known as the ReLU function. In the final stage a pooling function is used to effectively reduce the dimensionality of the set of convolved outputs. Pooling summarises the statistics of neighbourhoods in the convolved output fields, effectively sub-sampling the latent representation and can naturally handle inputs of varying size (Goodfellow et al. 2016). The most popular pooling functions replace neighbourhoods with local maximum values (noted as max pooling) or take an unweighted average of the neighbourhood. Max pooling is recognised as the better choice in most problems, likely due to the intrinsic non-linear nature of the operation.

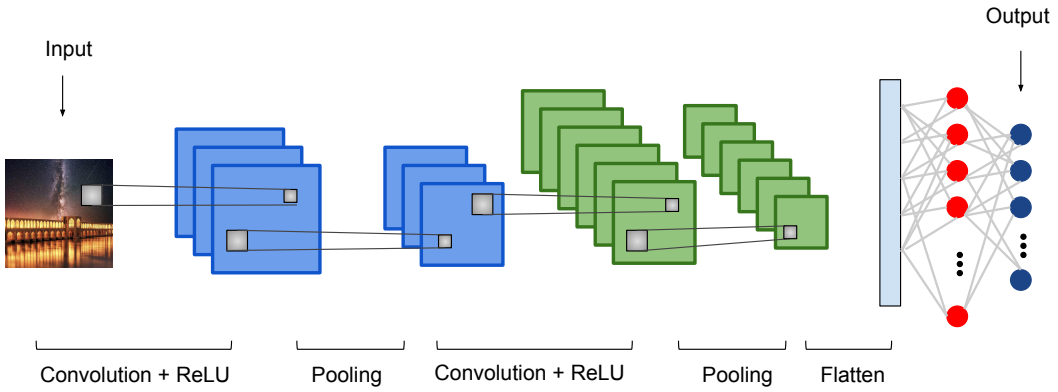

Figure 4.1. *Schematic of the structure of a typical CNN network with two convolutional layers.*

4.3.2 Long Short-Term Memory Networks

LSTMs (Hochreiter and Schmidhuber 1997) are a widely used methodology for sequence-to-sequence learning applications, including time-series forecasting (Orozco et al. 2018).The LSTM is a special case of a recurrent neural network (RNN) with recurrent gates. RNNs themselves are networks with self-excitation loops, enabling information to be preserved over time and hence for system dynamics to be learned (Figure 4.2a). RNN models update their output from time $t-1$ using incoming information at time t and a recurrent feedback loop such that $h_t = f(x_t, h_{t-1}; \theta)$, where x_t is the input vector and h_t is the output of the RNN hidden layer. In their simplest form, however, RNNs struggle to learn long-term dependencies, as prolonged gradient propagation over many cycles tends to dissipate the gradient information, a difficulty known as the vanishing-gradient problem. Goodfellow et al. 2016 discuss and analyse this issue in detail, suggesting a putative solution.

The LSTM structure (Figure 4.2b) is equipped with recurrent gates that are capable of learning long-term dependencies and consists of two main streams. The first is the memory cell, C_t, shown at the top of Figure 4.2b. Information is added to the memory cell stream through structures known as gates. The roles

of the gates are to allow information to flow optimally through the memory cell stream, by keeping a stable gradient. This stable gradient overcomes the issue of vanishing/exploding gradients to a large extent. Each gate is composed of a sigmoid layer, denoted as σ in Figure 4.2b, and a pointwise product operation. The sigmoid function's output is between zero and one, and determines the percentage of information that is passed through from each gate. Considering an input sequence x_t, input gate i_t, output gate o_t and forget gate f_t the estimation sequence in the LSTM may be written as:

$$f_t = \sigma(g_f(x_t, h_{t-1})), \quad i_t = \sigma(g_i(x_t, h_{t-1})), \quad o_t = \sigma(g_o(x_t, h_{t-1})). \tag{4.2}$$

The output of the LSTM layer, h_t, and the memory cell, C_t, are updated as

$$h_t = o_t \odot \tanh(C_t) \tag{4.3}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{4.4}$$

where the function $\odot$ is the pointwise product operator, and $\tilde{C}_t = \tanh(g_{\tilde{C}}(x_t, h_{t-1}))$.

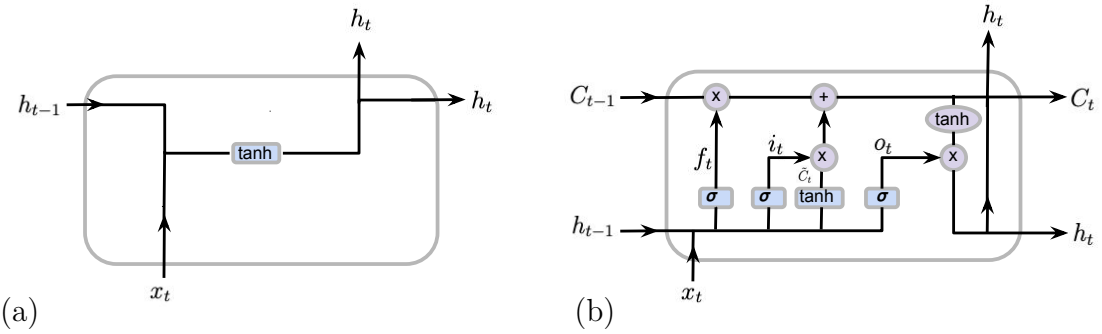

Figure 4.2. *Schematics of the structure of an RNN network (a) and an LSTM network (b). The operators in purple are pointwise operators. Those in blue are the activation functions.*

4.4 Bayesian Neural Networks

Bayesian Neural Networks (BNN) have seen a resurgence in interest in recent years (Ober and Aitchison 2021; Kwon et al. 2020; Ghosh et al. 2018). In common

with all Bayesian approaches, BNN models allow estimation (and propagation) of uncertainty through the model, typically with the aim of inferring a posterior distribution over the output variable $\mathbf{y}$. BNNs (indeed most Bayesian parametric models) do this by inferring the posterior distribution over parameters $\mathbf{w}$, given the data, $p(\mathbf{w} \mid \mathbf{x}, \mathbf{y})$, and a prior $p(\mathbf{w})$ which normally imposes the conjugate form of the posterior. In many Bayesian models, the prior is taken to be a multivariate Gaussian distribution (which can allow an element of analytic tractability). Taking the output likelihood, conditioned on $\mathbf{w}$ to be $p(\mathbf{y} \mid \mathbf{x}, \mathbf{w})$, we may impute a predictive distribution over some output variable, y^*, given a new input variable, x^*, via the marginal integral:

$$p(y^* \mid x^*, \mathbf{x}, \mathbf{y}) = \int p(y^* \mid x^*, \mathbf{w}) p(\mathbf{w} \mid \mathbf{x}, \mathbf{y}) d\mathbf{w}. \tag{4.5}$$

We will therefore need to calculate the posterior over weights to infer the predictive distribution over y^*, which is our objective. There are two main paradigms for approximating the posterior over weights $p(\mathbf{w} \mid \mathbf{x}, \mathbf{y})$; the sampling method (introduced by Neal 1995), and the variational inference method (Hinton and Van Camp 1993 were the first to use the variational inference approach for BNN without explicitly using the name).

For the sampling paradigm a common approach is to use MCMC. We note that using the Bayes rule:

$$p(\mathbf{w} \mid \mathbf{x}, \mathbf{y}) \propto p(\mathbf{y} \mid \mathbf{x}, \mathbf{w}) p(\mathbf{w}). \tag{4.6}$$

If we can draw sample from the right-hand side of Equation 4.6 that are representative of our posterior distribution $p(\mathbf{w} \mid \mathbf{x}, \mathbf{y})$, we can then use these samples to approximate the predictive distribution $p(y^* \mid x^*, \mathbf{x}, \mathbf{y})$. Using MCMC, the Markov chain explores the the space of $\mathbf{w}$, providing us with the samples from our distribution of interest $p(\mathbf{w} \mid \mathbf{x}, \mathbf{y})$. The sampling methods however could be challenging in deep neural networks and computationally cumbersome (Papamarkou et al. 2019).

The variational inference methods provide an alternative solution to sampling, by introducing a variational distribution $q_\theta(\mathbf{w})$, which is dependent on the variational parameters θ. The aim is to optimise these parameters such that the Kullback–Leibler (KL) divergence (Kullback and Leibler 1951) between $q_\theta(\mathbf{w})$ and $p(\mathbf{w} \mid \mathbf{x}, \mathbf{y})$ is minimised. In their basic form the variational inference methods in BNN rely on analytical solutions to integrals and are therefore limited to models with shallow networks. The stochastic variational inference itroduced by Graves 2011 tries to overcome this deficiency and to allow inference in deeper networks, by using Monte Carlo (MC) estimates of the derivatives instead of analytical ones. The stochastic variational inference method however suffers from noisy MC estimates and does not perform too well in practice.

4.4.1 Monte Carlo Dropout

In the work of Gal and Ghahramani 2016, it is shown that using dropout (Hinton et al. 2012) and drawing a different mask for each forward pass during training is equivalent to approximating a GP model with variational inference. The method is referred to as Monte Carlo (MC) dropout and has become a widely used approximate inference method for BNNs. A brief summary of the approach, based on the work of Gal and Ghahramani 2016, is as follows.

We start with a GP model and drive a variational inference approximation to the model (this order is in contrast to Neal 1995 who starts with a BNN to show that a single layer neural network that is infinitely wide, converges to a GP). We assume a covariance function for a GP of the following form

$$k(\mathbf{x}, \mathbf{x}') = \int p(\mathbf{w})p(\mathbf{b})h(\mathbf{w}^\mathbf{T}\mathbf{x} + \mathbf{b})h(\mathbf{w}^\mathbf{T}\mathbf{x}' + \mathbf{b})d\mathbf{w}d\mathbf{b} \tag{4.7}$$

where $h(.)$ is a non-linear activation function, $p(w)$ is a normal distribution, and $p(b)$ is some distribution. We further assume we have only one layer, where W_1 is the wight matrix connecting the input to the hidden layer, W_2 is the wight matrix connecting the hidden layer to the output, and $\mathbf{b}$ is the vector of biases. A priori,

each row of matrix W_i has the same distribution as $p(\mathbf{w})$. We can approximate $k(\mathbf{x}, \mathbf{x}')$ with Monte Carlo integration using K terms:

$$\hat{k}(\mathbf{x}, \mathbf{x}') = \frac{1}{K} \sum_{k=1}^{K} h(\mathbf{w_k^T x} + \mathbf{b_k}) h(\mathbf{w_k^T x}' + \mathbf{b_k}) \tag{4.8}$$

where $\mathbf{w_k} \sim p(\mathbf{w})$ and $b_k \sim p(b)$. These K samples then correspond to the number of hidden units in the single layer. Using $\hat{k}$ instead of k, our GP model can be formulated as $\mathbf{f} \mid \mathbf{x}, \mathbf{W_1}, \mathbf{b} \sim \mathcal{N}(0, \hat{k}(\mathbf{x}, \mathbf{x}'))$, where $\mathbf{W_1} = [\mathbf{w_k}]_{k=1}^{K}$ and $\mathbf{b} = [\mathbf{b_k}]_{k=1}^{K}$. Following standard GP methodology and integrating with respect to $\mathbf{f}, \mathbf{W_1}, \mathbf{b}$, the marginal likelihood can be written as:

$$p(\mathbf{y} \mid \mathbf{x}) = \int p(\mathbf{y} \mid \mathbf{f}) p(\mathbf{f} \mid \mathbf{x}, \mathbf{W_1}, \mathbf{b}) p(\mathbf{W_1}) p(b) d\mathbf{f} d\mathbf{W_1} d\mathbf{b} \tag{4.9}$$

Gal and Ghahramani 2016 then integrate Equation 4.9 with respect to $\mathbf{f}$ and introduce a $K \times 1$ auxiliary random variable $\mathbf{w_d} \sim \mathcal{N}(0, \mathbf{I}_K)$ that corresponds to $\mathbf{W_2} = [\mathbf{W_d}]_{d=1}^{D}$ and retrieve the marginal likelihood as follows (we refer the reader to Gal and Ghahramani 2016's Appendix for details):

$$p(\mathbf{y} \mid \mathbf{x}) = \int p(\mathbf{y} \mid \mathbf{x}, \mathbf{W_1}, \mathbf{W_2}, \mathbf{b}) p(\mathbf{W_1}) p(\mathbf{W_2}) p(b) d\mathbf{W_1} d\mathbf{W_2} d\mathbf{b}. \tag{4.10}$$

Equation 4.10 is the GP model re-parametrised and marginalised over $\mathbf{W_1}$, $\mathbf{W_2}$, and $\mathbf{b}$. This is equivalent to the weighted basis function interpretation of GPs discussed in Section 2.2.1 in Chapter 2.

The integral 4.10 is however intractable. We can estimate this with variational inference using variational distribution $q(\mathbf{W_1}, \mathbf{W_2}, \mathbf{b}) = q(\mathbf{W_1}) q(\mathbf{W_2}) q(\mathbf{b})$. Our minimisation objective is the KL divergence between the approximate posterior using MC sampling realisations $\hat{\mathbf{W}}_\mathbf{1}$, $\hat{\mathbf{W}}_\mathbf{2}$, $\hat{\mathbf{b}}$, and the GP posterior

$$\begin{aligned} -\mathscr{L}_{GP-MC}(\theta) = &- \sum_{t=1}^{T} \log p(\mathbf{y} \mid \mathbf{x}, \hat{\mathbf{W}}_\mathbf{1}, \hat{\mathbf{W}}_\mathbf{2}, \hat{\mathbf{b}}) \\ &+ \mathrm{D}_{\mathrm{KL}}(q_\theta(\mathbf{W_1}, \mathbf{W_2}, \mathbf{b}) \| p(\mathbf{W_1}, \mathbf{W_2}, \mathbf{b})). \end{aligned} \tag{4.11}$$

The final step is to approximate this KL divergence. Gal and Ghahramani 2016 show that in networks with large number of weights, this objective is equivalent to

that of optimising a neural network and using dropout technique for regularisation. We refer the reader to Gal and Ghahramani 2016 for further details on this.

We use an alternative approach to evaluating uncertainty, where we estimate the multimodal posterior distribution $p(\mathbf{y} \mid \mathbf{x})$, using the Deep-MDN method. We explain the method in the following section and make rigorous comparisons to dropout, where we show that our method is computationally more efficient.

4.5 Deep Mixture Density Network Model

The Deep Mixture Density Network (Deep-MDN) model is an extension of the Mixture Density Network model discussed in Chapter 3, where the posterior distribution is given via a mixture model with m components. We follow the same steps and assumption in Equations 3.1 to 3.7 presented in Chapter 3. The model resembles the same dynamics as the RBFN model presented in Chapter 3.3.3 with the exception of the network's structure. We replace the radial basis function layer with deep networks, and experiment with two of the most commonly used neural networks architectures, namely the LSTM and the CNN, which here we provide with two convolutional layers (see, e.g. Goodfellow et al. 2016). Figure 4.3 illustrates the schematic of the Deep-MDN models we use in this chapter.

4.6 Experiments

We compare the Deep-MDN model with $m = 1$, noted as Deep-MDN-1, to a standard DL model noted as SDL, a standard DL model with dropout noted as SDL-D, and a Bayesian deep learning model where dropout is used to estimate the Bayesian posterior distribution noted as BDL.

Our SDL baseline models are LSTM and CNN structures with the same architectures as used in the Deep-MDN-1 model. For the SDL-D model, we introduce dropout with the same LSTM and CNN structures of the standard

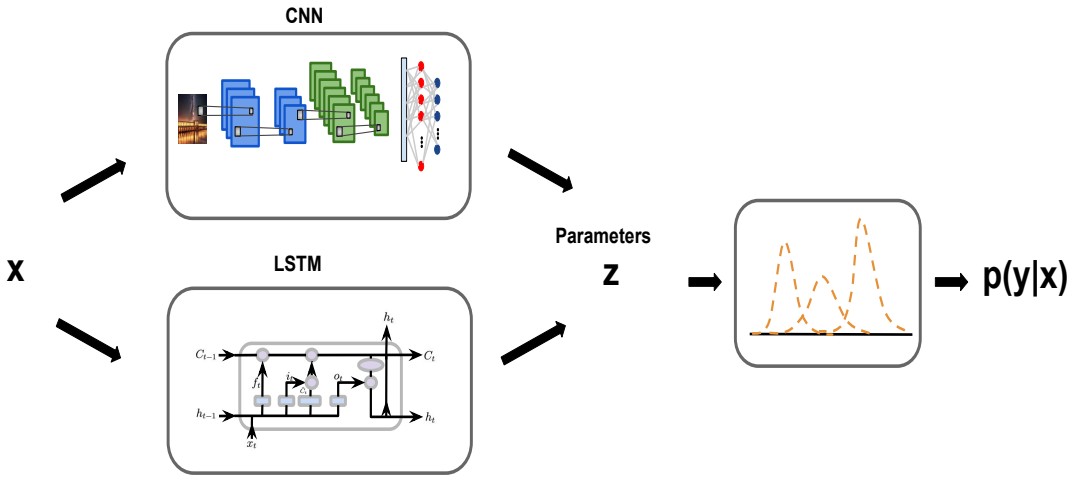

Figure 4.3. Schematic of the Deep-MDN models.

DL model. The dropout rate is set to 20% for all experiments[1]. We use the mean square error (MSE) as the loss function in training across all models, for comparison and consistency.

For the BDL we use the same models, loss function, and dropout rate as SDL-D, but we further use dropout to estimate the posterior distribution as proposed by Gal and Ghahramani 2016. We note that SDL and SDL-D make point estimate predictions while BDL and Deep-MDN-1 estimate posterior distributions at each time interval.

4.6.1 Datasets

We perform experiments on six datasets, namely the Mackey-Glass chaotic dataset, sunspot dataset (Clette 2015), air quality time series (CO levels at hourly intervals, De Vito et al. 2008), humidity levels in the kitchen time series (10 minutes intervals, Candanedo et al. 2017), US initial jobless claims (USIJC, weekly intervals, Bureau of Labor Statistics, United States Department of Labor 2018), and internet traffic dataset (hourly intervals, Cortez et al. 2012).

[1]We find results are not highly sensitive to the dropout rate.

All algorithms have as input the last k data points, where k is set to 2 for LSTM and 10 for CNN.

4.6.2 Comparison across Models

The plots in Figure 4.4 illustrate the results for the Deep-MDN-1 model in comparison with our baseline models for the Mackey-Glass time series[2]. All errors are computed over the test dataset which consists of 400 data points post the training set.

The error curves in the top row of Figure 4.4,are the average of the negative log likelihood for the test dataset at 200^{th} epoch intervals. One standard deviation standard errors are shown over 20 runs. We note that the Deep-MDN-1 model is significantly more efficient compared to other models, showing rapid convergence to lower errors in comparison. The boxplots in the second row of Figure 4.4 show the MSEs associated with the final epoch for a random run for each model. These plots illustrate not only that the average error is significantly lower in the Deep-MDN-1 model, but also that the errors are lower for all time intervals in the test dataset.

The bottom row of Figure 4.4 shows QQ plots for both the BDL and Deep-MDN-1 models (the only two models which estimate posterior distributions). These indicate that Deep-MDN-1's posterior distribution is better calibrated compared to the BDL model for both LSTM and CNN models.

We note that the results in Figure 4.4 are over a total run of 1200 epochs and that all models will *eventually* converge to low error levels, albeit after significantly more epochs compared to Deep-MDN-1. Table 4.1 details the comparative efficiency, in terms of timing ratios for all datasets. For fair comparison all experiments are conducted using the Keras library (Chollet et al. 2015).

[2]The results show the same trends in all other datasets.

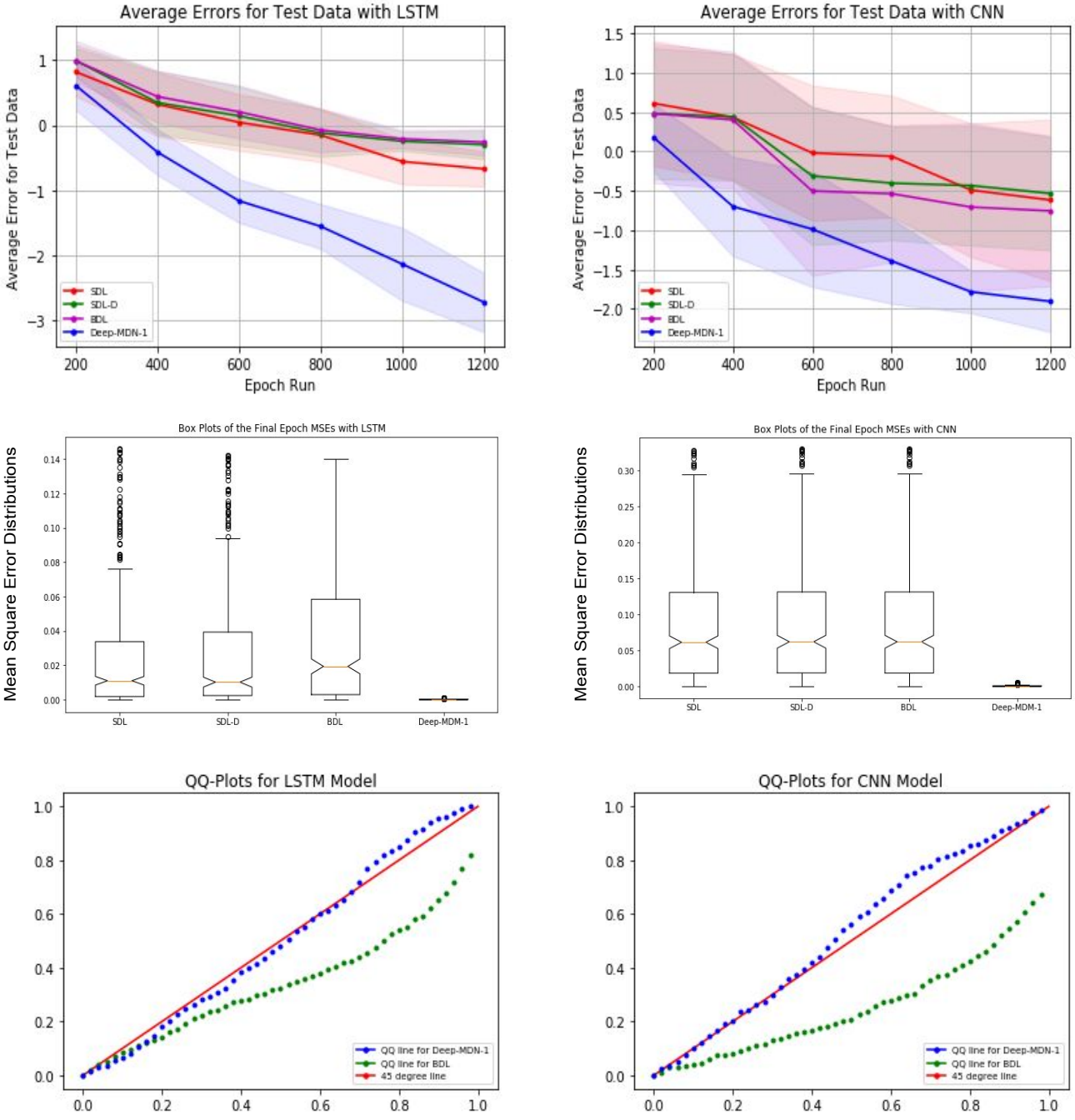

Figure 4.4. Top row: Average error curves for every 200$^{\text{th}}$ epoch. Middle row: Box plots of the MSEs for the final epoch. Bottom row: QQ-plots for Deep-MDN-1 and BDL models.

Table 4.1. Timing of models to that of Deep-MDN-1. These ratios represent the timing compared to Deep-MDN-1 for the same level of average error.

		SDL	SDL-D	BDL
Mackey-Glass	LSTM	2.9	4.0	4.0
	CNN	5.6	7.3	6.7
Sunspot	LSTM	1.1	2.2	2.3
	CNN	1.5	2.6	2.4
Air quality	LSTM	2.2	1.3	1.3
	CNN	3.2	1.6	1.6
Humidity levels	LSTM	1.3	3.5	3.2
	CNN	1.9	5.1	4.2
USIJC	LSTM	3.0	5.4	5.4
	CNN	1.4	2.0	1.7
Internet traffic	LSTM	3.1	6.7	6.9
	CNN	1.7	2.6	2.6

4.6.3 Comparison between Different Orders of Mixture Coefficient

In Section 4.6.2 our estimates are for a unimodal Deep-MDN model. This is to ease comparison across models. Deep-MDN is however capable of estimating multimodal datasets and can allow flexibility in the posterior distribution. In this section we compare the performance of the Deep-MDN model for different values of mixture coefficient m.

The plots in Figure 4.5 illustrate Deep-MDN models with mixture components, m, set to 1, 3 and 10 for the Mackey-Glass time series. We refer to the extra other models as the Deep-MDN-3 and the Deep-MDN-10 respectively. The plots indicate that increasing the mixture coefficient will further enhance convergence and accelerate computation, but note that there is a diminishing benefit to increasing m.

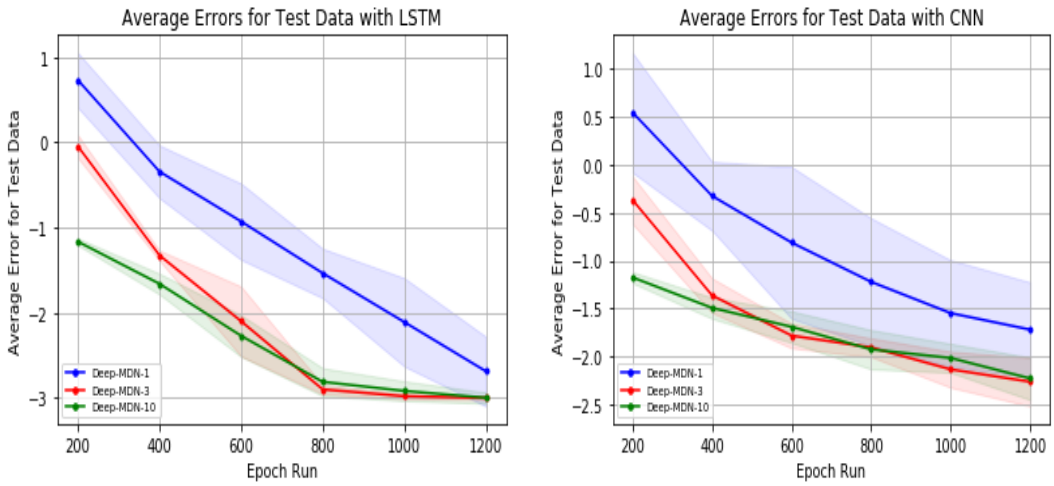

Figure 4.5. Test data negative log likelihood curves for Deep-MDN models with mixture components 1, 3, and 10.

4.7 Conclusion

In this chapter we present Deep-MDN models, which we show are capable of efficiently estimating posterior distributions. We compare our model to baseline DL models including Bayesian deep learning model, Monte Carlo dropout. Our approach significantly outperforms the baseline models both in terms of learning efficiency and in the calibration of posterior uncertainty.

The computational efficiency of many probabilistic models including BNNs are one of the main hindrances in choosing them. We have shown that our approach is significantly less costly in estimating predictive uncertainty. Further our approach has the capability to estimate a flexible posterior which goes beyond the capabilities of models such as MC dropout which assume a unimodal posterior.

In Chapter 5 we expand the Deep-MDN model further and present an extension of the model inspired by generative adversarial networks (Goodfellow et al. 2014).

5

Mixture Density Conditional Generative Adversarial Model

Contents

5.1 Introduction

Generative Adversarial Networks (GANs) have been one of the many breakthroughs in Deep Learning methods in recent years. Several different variations of the model

have been introduced since the method was first introduced by Goodfellow et al. 2014. One of the most popular variations of this work is that of the Conditional Generative Adversarial Network (CGAN) (Mirza and Osindero 2014) in which the generator and discriminator (we review the GAN process in Section 5.3) are both conditioned on some observed information. In the application of time series forecasting, future values are conditioned on information observed from the past, either from the time series itself, some set of associated exogenous data, or a combination of the two. This conditioning addition to the GAN formalism makes the CGAN approach particularly useful as a foundational model for time series prediction. Most applications of (C)GANs, however, have been within computer vision and, to a lesser extent, in natural language processing and simulation models (Wu et al. 2019; Hodge et al. 2019; Barth et al. 2019).

The literature on the application of any form of GAN model to problems associated with time series is, to date, limited. However, some recent literature shows the potential usefulness of the method. For example, the work reported by Esteban et al. 2017 applies a recurrent GAN to generate realistic, synthetic, medical data series, and, in Wiese et al. 2020, a (standard) GAN model is used to generate realistic financial asset prices and analyze their distributions.

In Zhou et al. 2018, a GAN is used to forecast high-frequency stock data, and, in Luo et al. 2018, GANs are used to generate missing values in incomplete time series. We note that the GAN models used in all these applications make point estimates for the forecast. Although a perfectly valid approach, and one that has a long history in time series forecasting, as discussed in Chapter 1, we argue that probabilistic forecasts are a prerequisite in many application domains, in which knowledge of the predictive uncertainty is as vital as the prediction itself. In this chapter, we expand on the CGAN algorithm to allow a full predictive probability distribution, rather than a point value. To obtain richer predictive densities, we model the posterior distribution using a finite Gaussian Mixture model (GMM). Although we find only occasional evidence that such non-Gaussian predictions

offer significant benefits, we note that producing them is not much more costly than single Gaussian predictive distributions, and so present our approach as a more general multi-component model.

This chapter is set out as follows: first, we recast, via modifications of the GAN loss functions, the conditional GAN approach to provide (multimodal) probability densities over forecasts; second, we show, through a series of comparative experiments, that GAN methods—particularly our proposed approach—are able to make good forecasts, especially in situations in which noise is prevalent. We show how our proposed method is extremely robust to variable noise injections. Finally, we demonstrate how, for some time series forecasting problems, superior results are obtained by entertaining a more complex, multi-modal, forecast distribution.

5.2 Related Work

In this section we review recent literature which is close to our approach. We start by noting the work of Y. Yu and Zhou 2018, in which a mixture of GAN models is proposed as a data clustering model. Although clearly related, this is somewhat different to our approach, in which we use a single GAN generator, linked to the hyperparameters of the posterior mixture model, rather than a mixture of GAN models. Furthermore, our goal is forecasting rather than unsupervised data classification. The approaches advocated in Gurumurthy et al. 2017 and Ben-Yosef and Weinshall 2018 propose a latent space, used for sampling latent vectors in the GAN, formulated via a Gaussian mixture. The latter replaces, in these papers, the single Gaussian distribution used for such generation in standard GAN models. In both these papers, the generator and discriminator have a similar structure to a standard GAN. In Eghbal-zadeh et al. 2019, a mixture model is used, but for the discriminator alone, with the generator being that of a standard GAN model; we note the difference to our approach, in which the *generator* is a GMM.

Yoon et al. 2019 propose a model to best approximate the joint distribution over a set of static and temporal features, associated with a time series. The authors use a GAN to propagate these features so as to model the dynamics of the time series. We note that the outputs from the model are point estimates and that, in the case of a 1-d system, the method is the same as CGAN albeit with an added autoencoder. Finally, Richardson and Weiss 2018 compares (standard) GAN models to GMMs for image generation. The authors show that GANs are superior in their ability to generate sharp images, but note that mixture models offer more efficient inference. They propose a combination of the two and introduce a GAN model in which the GAN generator is a mixture model. However, the sample generator still makes point estimates from the multimodal distribution in order to retain a discriminator function the same as that of a standard GAN model. We offer discriminator extensions which allow the GAN process to operate on the full (multimodal) posterior distribution.

5.3 The GAN and CGAN Models

We start by formulating the definitions which we use in the (C)GAN models. We consider a time series, y_t. Our aim is to estimate the forecast of some $y^f_{t'>t}$, conditioned on a set of observations which we denote $\mathbf{x}_t$, the input to our model (in our experiments we use the lagged output variable as our input). Every (C)GAN model has two network units, a generator network and a discriminator network. In the GAN model the input to the generator network is $\mathbf{z_g}$, and in the CGAN model the inputs to the generator network are $\mathbf{x}_t$ and $\mathbf{z_g}$, where $\mathbf{z_g}$ is a collection of samples from a normal distribution, $p(z_g) = \mathcal{N}(0, \sigma^2_{\text{data}})$. During model training, the output from the generator, $y^f_{t'}$, as well as the true forecast sample $y_{t'}$, are fed to the discriminator, whose role is to discriminate between them i.e. to identify $y^f_{t'}$ as the 'fake' sample.

5.3.1 GAN Model

The goal of the GAN method is to estimate a generative model using an *adversarial* process (Goodfellow et al. 2014). This is achieved by simultaneously training two models. Firstly, a generative model G, that in the case of data forecasting learns past patterns in the data and infers the predictive values. Secondly, a discriminative model D, that determines how likely a sample was to originate from the 'true' training data, compared to being sampled from the generator. The generator is hence matched against an adversary, the discriminator (whose goal is to detect the difference between a true data sample and one created by the generator). Components of the model are then trained (via an optimization process) to maximize the probability of the discriminator being unable to distinguish true from generated data samples. Typically, including the approach we take here, the generator and the discriminator are both constructed as multilayer perceptrons, with stochastic gradient methods being employed to obtain optimization. Schematic of the GAN model is depicted in Figure 5.1a.

The value function for $V(G, D)$ for the discriminator and the generator is formulated as follows:

$$\begin{aligned}\min_G \max_D V(G, D) = &\mathbb{E}_{y_t \sim p_{data}(y_t)}[\log D(y_t)] \\ &+ \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z_g)))].\end{aligned} \tag{5.1}$$

In practice, we train the generator G, to maximise $\log D(G(z_g))$ instead of minimising $\log(1 - D(G(z_g)))$. This is due to the fact that early in training, the generator's performance is poor and maximising $\log D(G(z_g))$ provides stronger gradients during early stages of training (Goodfellow et al. 2014).

5.3.2 CGAN Model

In an unconditioned GAN model, there are no controls over the data that is generated. In the CGAN model, in contrast, by conditioning the model on

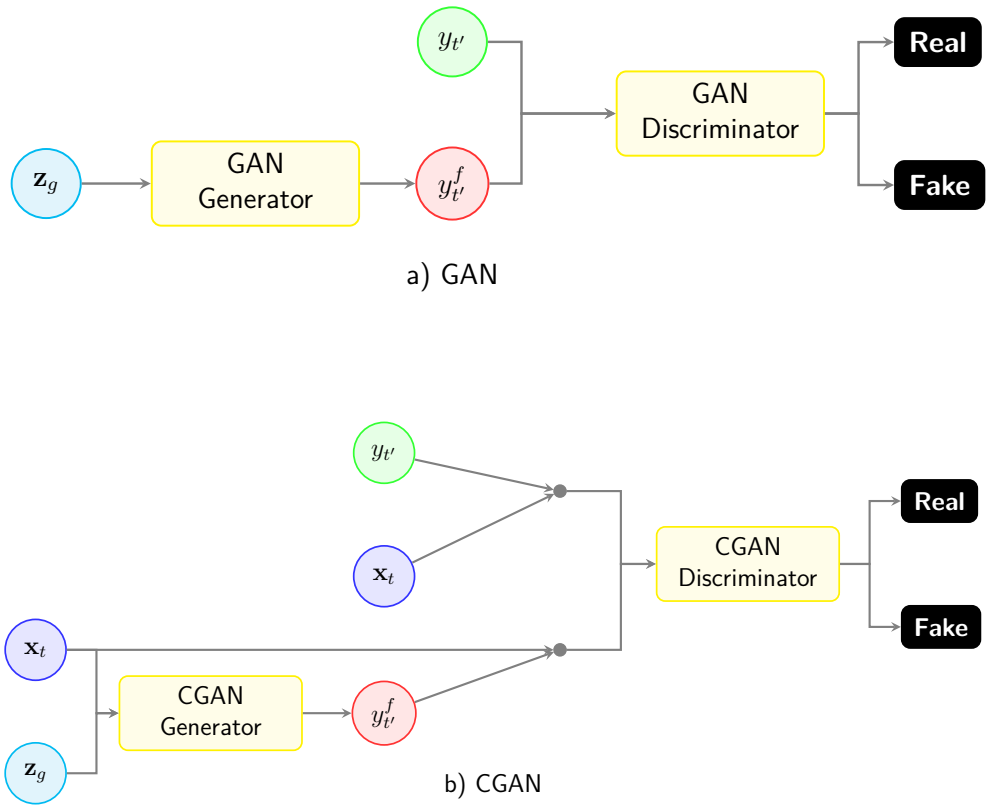

Figure 5.1. *Schematic of (a) GAN and (b) CGAN models, showing Generator and Discriminator components and associated variables.*

additional information, it is possible to direct the data generation process (Mirza and Osindero 2014). In the CGAN model the generator and the discriminator are both conditioned on further information. This makes CGANs an interesting framework for time series forecasting. Schematic of the CGAN model is depicted in Figure 5.1b, and the min-max objective function is the following:

$$\begin{aligned}\min_G \max_D V(G,D) = \mathbb{E}_{y_t \sim p_{data}(y_t)}[\log D(y_t \mid x_t)] \\ + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z_g \mid x_t)))].\end{aligned} \tag{5.2}$$

5.4 The MD-CGAN Model Framework

As with the GAN and CGAN methods, we consider a time series, y_t. Our aim now is to infer the *posterior distribution* over some $y_{t'>t}$, conditioned on the set

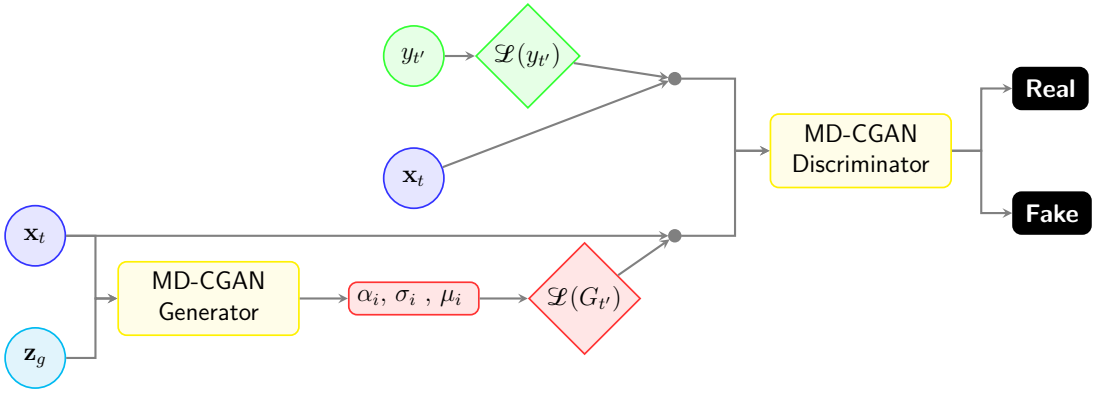

Figure 5.2. *Schematic of the MD-CGAN model. We note that the discriminator in the MD-CGAN model has a different loss function and structure to the GAN and CGAN models.*

of observations, $\mathbf{x}_t$. In order to form the posterior distribution we model the *full conditional density* $p(y_{t'}|\mathbf{x}_t)$ as an adversarial network. To achieve this we use a Mixture Density Network (MDN) model (Bishop 1995) for the generator G. The inputs to the generator network are as per the CGAN approach, $\mathbf{x}_t$ and $\mathbf{z_g}$, where $\mathbf{z_g}$ is, as before, a collection of samples from a normal distribution, $p(z_g) = \mathcal{N}(0, \sigma^2_{\text{data}})$. The outputs of $G_{t'}(\mathbf{x}_t, \mathbf{z}_g)$ are now, however, the parameters of the Gaussian mixture posterior over the forecast. This mixture has mixing coefficient, standard deviation and mean for the i-th component denoted as α_i, σ_i, and μ_i respectively. As proposed in Bishop 1995 and discussed extensively in Chapter 3, we achieve this by using latent variables $\mathbf{s} = \{\mathbf{s}_\alpha, \mathbf{s}_\sigma, \mathbf{s}_\mu\}$, conditioned on the inputs. The mapping from $[\mathbf{x}_t, \mathbf{z}_g] \mapsto \mathbf{s} \mapsto \{\alpha_i, \sigma_i, \mu_i\}$ is modelled via our network. As the mixings must satisfy $\sum_i \alpha_i = 1$, we map $\mathbf{s}_\alpha$ to α via the *softmax* function, where $\alpha_i = \frac{\exp(s_{\alpha,i})}{\sum_{i'} \exp(s_{\alpha,i'})}$. The elements of σ are strictly positive so we adopt, $\sigma_i = \exp(s_{\sigma,i})$. Finally the means can be mapped directly from the latent variables, hence $\mu_i = s_{\mu,i}$. Schematically, the MD-CGAN method is depicted in Figure 5.2.

The above formalism allows us to directly model the predictive likelihood conditioned on an input, and the likelihood of G, conditioned on the observations

$\mathbf{x}_t$ and samples $\mathbf{z}_g$ as:

$$\mathscr{L}(G_{t'}(\mathbf{x}_t, \mathbf{z}_g)) = \sum_{i=1}^{m} \alpha_i(\mathbf{x}_t, \mathbf{z}_g)\mathcal{N}_i(y_{t'}|\mu_i(\mathbf{x}_t, \mathbf{z}_g), \sigma_i(\mathbf{x}_t, \mathbf{z}_g)) \tag{5.3}$$

where m is the number of mixture components.

As in the CGAN model, the discriminator, D, is also conditioned on $\mathbf{x}_t$. The input to the discriminator model is, by design, $\mathbf{x}_t\sqrt{2\pi}\sigma_a\mathscr{L}(y_{t'})$, where σ_a is the standard deviation of the set of observed y_t [1]. For true values of $y_{t'}$, $\sqrt{2\pi}\sigma_a\mathscr{L}(y_{t'})$ is maximized. The generator tries to 'fool' the discriminator by generating $G_{t'}$ such that the $\sqrt{2\pi}\sigma_a\mathscr{L}(G_{t'})$ is maximized. The loss function for the generator, L_G is as in Equation 5.4, The discriminator network, on the other hand, attempts to differentiate between true $y_{t'}$ values and the pseudo-values created by the generator. The loss function for the discriminator, L_D is as in Equation 5.5.

$$L_G = \mathbb{E}_{z\sim P_z(z)}[-\mathscr{L}(G_{t'}(\mathbf{x}_t, \mathbf{z}_g))] \tag{5.4}$$

$$\begin{aligned} L_D =& \mathbb{E}_{y\sim P_{\text{data}}(y)}[\|\mathbf{x}_t\sqrt{2\pi}\sigma_a\mathscr{L}(y_{t'}) - \mathbf{x}_t\|^2] + \\ & \mathbb{E}_{z\sim P_z(z)}[\|\mathbf{x}_t\sqrt{2\pi}\sigma_a\mathscr{L}(G_{t'}(\mathbf{x}_t, \mathbf{z}_g))\|^2] \end{aligned} \tag{5.5}$$

We note that the lowest value of the discriminator loss is achieved when $\sqrt{2\pi}\sigma_a\mathscr{L}(y_{t'})$ is maximal (unity) and $\mathscr{L}(G_{t'}(\mathbf{x}_t, \mathbf{z}_g))$ is minimal (zero). Our algorithm, thus, follows the steps in Algorithm 1.

5.5 Experiments

5.5.1 Comparison with Other Learning Models

To provide a range of comparisons to methods related to this work, we compare our MD-CGAN model to the following baseline methods: the Mixture Density Network (MDN) model, chosen to baseline mixture density outputs against (Bishop 1995);

[1] We note that the model is not sensitive, within reason, to this value (it is, in effect, a constant in the update equations) and we discuss its setting later in the chapter.

Algorithm 1 MD-CGAN Algorithm

for number of training iterations **do**
  for j steps do **do**
    Sample N noise samples, $\{\mathbf{z}^1,...,\mathbf{z}^N\}$ from $p_g(\mathbf{z})$
    Sample N data points, $\{\mathbf{x}^1,...,\mathbf{x}^N\}$ from $p_{\text{data}}(\mathbf{x})$
    Update the discriminator by descending its stochastic gradient:

$$\nabla_{\theta_{\mathscr{L}}} \sum_{n=1}^{N} [\|\mathbf{x}^{(n)}\sqrt{2\pi}\sigma_a \mathscr{L}(y^{(n)}) - \mathbf{x}^{(n)}\|^2 + \|\mathbf{x}^{(n)}\sqrt{2\pi}\sigma_a \mathscr{L}(G(\mathbf{z}^{(n)}, \mathbf{x}^{(n)}))\|^2]$$

  end for
  Sample N noise samples, $\{\mathbf{z}^1,...,\mathbf{z}^N\}$ from $p_g(\mathbf{z})$
  Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \sum_{n=1}^{N} -\mathscr{L}(G(\mathbf{z}^{(n)}, \mathbf{x}^{(n)}))$$

end for

the CGAN model, chosen as a well-known GAN approach (Mirza and Osindero 2014); and a "standard" multilayer perceptron neural network (SNN) as a simple, yet effective, baseline. As a more traditional, parametric, benchmarking model, we use regular (linear-Gaussian) Autoregressive (AR) models (Papoulis 1984), with parameters obtained by standard least-squares maximum-likelihood estimation. To promote as fair comparison as possible for the nonlinear methods, we use a *common* neural network architecture across core components in the models, choosing the neural network structure commonly used for GAN models in recent literature (Mirza and Osindero 2014). We do not alter this structure throughout our experiments, and we keep to fixed hyperparameter settings (guided by those used in Mirza and Osindero 2014). Figure 5.3 provides a schematic of the structure used. We note that, whilst the lengths of the input and output vectors are dependent on the model, the structure of all networks remains constant.

5.5.2 Details of Implementation

All models were coded in the Python language, and we use the Keras library (Chollet et al. 2015) to build the neural networks.

The neural networks in Figure 5.3 follow the structure of the CGAN model detailed in Mirza and Osindero 2014. The hyperparameters in the models were set as follows: for the discriminator, both in CGAN and MD-CGAN, the dropout parameter is set to 0.4, and alpha for the leaky ReLU is set to 0.2. For the generator, the dropout rate is set to 0.5, and alpha for the leaky ReLU is, again, 0.2. The parameter governing the number of neurons, n, in the dense layers of the neural network modules is set to 40. The variance parameter, σ_a, in the MD-CGAN models is set to 0.2. During training, we follow the steps of algorithm 1 and set j to 1 for all datasets.

The number of training iterations is, however, specific to model and dataset. With the exception of the GAN models, we monitor the errors, until they reach saturation during training. For the GAN models, in which both the generator and the discriminator have a loss value per iteration, we monitor the average sample error from the training data and continue iteration until the errors reach saturation.

In all models, we optimize parameters using the Adam optimizer (Kingma and Ba 2015), with learning rate set to 0.001, exponential decay for the first momentum set to 0.9, exponential decay for the second momentum set to 0.999, and epsilon set to 10^{-7}.

5.5.3 Data

We perform experiments on twelve datasets with differing provenance; including Mackey-Glass chaotic dataset (Mackey and Glass 1977); the Sunspot dataset (Clette 2015); eight financial time series; and two other noisy datasets. We first look at the one-step forecasting and the issue of (test set) noise resilience, focusing on controlled experiments with the Mackey-Glass and Sunspot datasets (Subsection

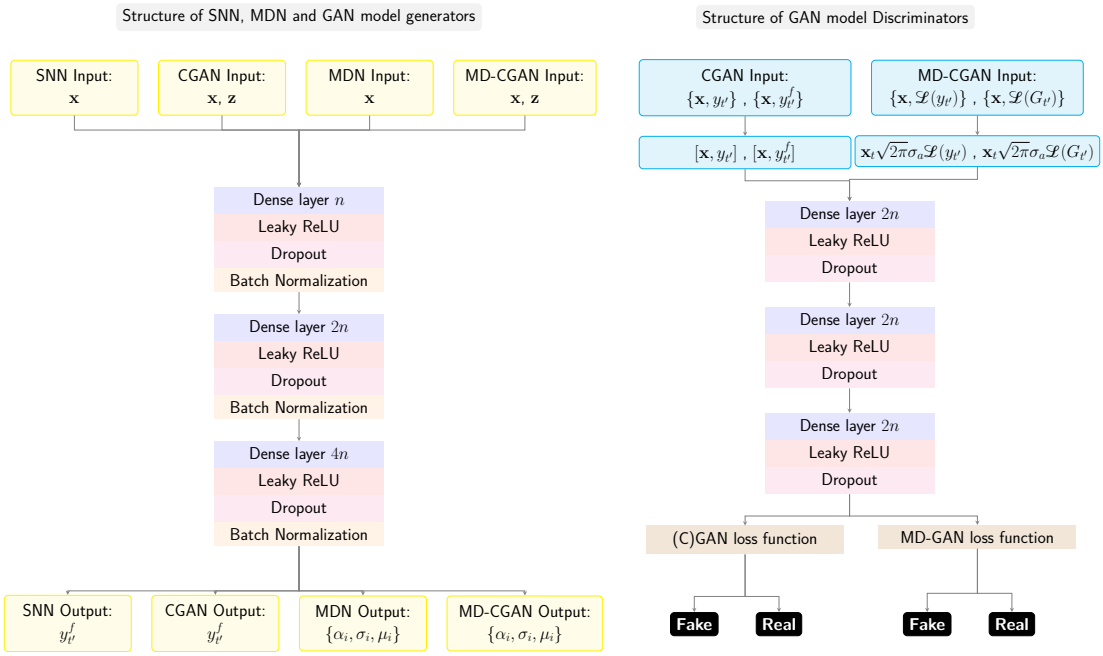

Figure 5.3. *Schematic of a common neural network structure used across all models.*

5.5.4). We then expand our experiments to consider the other ten datasets with an increased forecast horizon in Subsection 5.5.5.

For all datasets we use, the time series is split into training and out-of-sample test sets. The training datasets in all our experiments are comprised of 2000 samples, and all test sets consist of 400 sequential data points post the training set. All performance metrics are obtained from the test set only. To enable a simple comparison across all models and datasets, we keep a fixed lookback window of data as input. All models are, therefore, provided, as input, with the last k data points, set to $k = 5$ for the purpose of all our experiments. Of course, in practice, the lookback window (or other observation vector) would be optimized for a model and dataset, typically using shrinkage, model-complexity penalties or cross-validation methods (Bishop 2006). Our goal here is to showcase like-for-like performance when exactly the same observation vector is available across all models. We run both an AR(5) model (corresponding to the common $k = 5$ observation vector)

and, as a martingale baseline, the AR(0) model, in which the forecast is simply the previous observation. All datasets are pre-normalized to the [0,1] interval, again to allow for simpler comparison across data and methods. We further note that both the CGAN and SNN models make *point estimate* predictions, whilst MD-CGAN and MDN estimate posterior *distributions*. To enable a simple comparison across models, therefore, we report the mean-square error (MSE) for all methods. The number of mixture components, m, in both MD-CGAN and MDN is set to unity (we vary this in Section 5.5.6), to further ease comparison across models, some of which produce a probability distribution at output and others point values. The posterior mean of the predictive distribution is taken as the forecast value for both the MDN and MD-CGAN models for the purposes of point-prediction error reporting. Selecting $m > 1$ for the MD-CGAN and MDN methods would, in principle, be possible, but, as the posterior distribution is non-Gaussian (it is a mixture of Gaussians), the associated error metric is no longer a simple MSE, making like-for-like comparison to other approaches difficult.

5.5.4 One-step Forecasting

We start our experiments looking at one-step ahead forecasts on two well known datasets, the Mackey-Glass chaotic time series (Mackey and Glass 1977) and the Sunspot dataset (Clette 2015). We consider one-step forecast errors in the presence of increasing test set noise. We add 5% to 30% (by amplitude) normally distributed noise to the test data (from a GAN perspective, these input perturbations are, in effect, treated similarly to adversarial attacks). However, a true *adversarial* attack has an intelligently chosen perturbation to maximize malicious impact. We note that no noise (perturbation) is added to the training dataset. Mean Square Errors (MSE) are presented in Tables 5.1 and 5.2 and Figure 5.4 for all algorithms considered. For both datasets, we see that MD-CGAN has the best performance for noise levels of 10% and above. Indeed, we see that the GAN models (particularly MD-CGAN) perform consistently well (especially in the

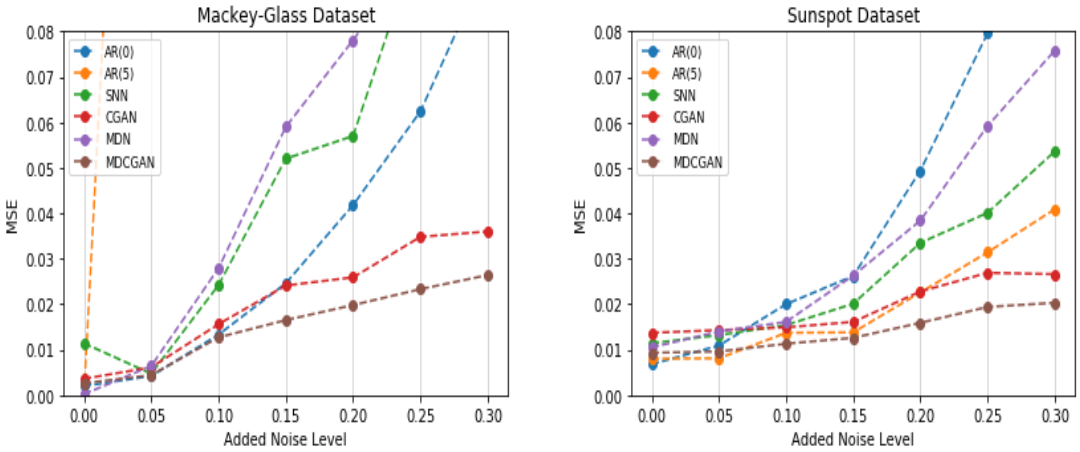

Figure 5.4. Comparative MSE plots with increasing test set noise perturbation. We note that the GAN-based methods, particularly MD-CGAN, perform consistently even as noise levels increase.

Mackey-Glass example) across multiple noise levels, indicating that the approach is particularly resilient to additive observation noise. This is to be expected, as GAN approaches treat the additive noise as adversarial perturbation to the input, against which they are designed to be robust.

In the next section, we investigate model performance at longer forecast horizons, mainly in the finance domain, which is known to contain variable amounts of stochasticity. Financial times series are often dominated by stochastics, and we expect GAN approaches to be well-suited to forecasting in these circumstances.

5.5.5 Forecasts over Longer Horizons

One-step forecasts were presented in Section 5.5.1. Here, we extend analysis over longer horizons. All models make estimates over a horizon of ten weeks for an extended set of financial time series, namely:

- U.S. initial jobless claims (USIJC, weekly intervals) (Bureau of Labor Statistics, United States Department of Labor 2018);

Table 5.1. Mackey-Glass data: MSE variation with added noise level, with standard deviations in brackets.

	AR(0)	AR(5)	SNN	CGAN	MDN	MD-CGAN
0% noise	0.0020	**4.1e-06**	0.0014	0.0036	0.0002	0.0026
	(0.0000)	(0.0000)	(0.0000)	(0.0002)	(0.0000)	(0.0001)
5% noise	**0.0042**	0.2794	0.0047	0.0061	0.0064	0.0044
	(0.0000)	(0.0000)	(0.0001)	(0.0004)	(0.0001)	(0.0002)
10% noise	0.0133	1.1558	0.0242	0.0155	0.0278	**0.0126**
	(0.0000)	(0.0000)	(0.0015)	(0.0009)	(0.0007)	(0.0002)
15% noise	0.0246	2.6581	0.0519	0.0240	0.0589	**0.0165**
	(0.0000)	(0.0000)	(0.0014)	(0.0009)	(0.0011)	(0.0004)
20% noise	0.0418	4.4868	0.0570	0.0259	0.0780	**0.0197**
	(0.0000)	(0.0000)	(0.0021)	(0.0013)	(0.0012)	(0.0007)
25% noise	0.0624	7.3152	0.1013	0.0347	0.0980	**0.0233**
	(0.0000)	(0.0000)	(0.0023)	(0.0012)	(0.0012)	(0.0008)
30% noise	0.0951	10.1527	0.1640	0.0360	0.1402	**0.0264**
	(0.0000)	(0.0000)	(0.0099)	(0.0012)	(0.0011)	(0.0006)

- EURUSD foreign exchange daily rates (EURUSD FX rate) (Investing.com 2018);
- WTI crude oil spot prices (WTI) (U.S. Energy Information Administration 2020);
- Henry Hub Natural Gas spot prices (NG) (U.S. Energy Information Administration 2020);
- CBOE Volatility Index (VIX) (Yahoo!Finance 2020a);
- New York Harbor No. 2 Heating Oil Spot Price (Heating Oil) (U.S. Energy Information Administration 2020);
- Invesco-DB U.S. Dollar Index Bullish Fund (USD Index) (Yahoo!Finance 2020b);
- iShares MSCI Brazil Small-Cap ETF (SC ETF) (Yahoo!Finance 2020c);

as well as two other time series:

- Number of daily births in Quebec (DB) (Hipel and McLeod 1994); and

Table 5.2. Sunspot data: MSE variation with added noise level, with standard deviations in brackets.

	AR(0)	AR(5)	SNN	CGAN	MDN	MD-CGAN
0% noise	0.0080	**0.0068**	0.0114	0.0137	0.0105	0.0093
	(0.0000)	(0.0000)	(0.0004)	(0.0002)	(0.0000)	(0.0001)
5% noise	0.0109	**0.0081**	0.0132	0.0143	0.0140	0.0096
	(0.0000)	(0.0000)	(0.0006)	(0.0008)	(0.0008)	(0.0002)
10% noise	0.0200	0.0137	0.0154	0.0149	0.0161	**0.0113**
	(0.0000)	(0.0000)	(0.0008)	(0.0012)	(0.0013)	(0.0002)
15% noise	0.0262	0.0138	0.0201	0.0161	0.0263	**0.0126**
	(0.0000)	(0.0000)	(0.0009)	(0.0010)	(0.0013)	(0.0002)
20% noise	0.0494	0.0226	0.0335	0.0228	0.0384	**0.0159**
	(0.0000)	(0.0000)	(0.0014)	(0.0022)	(0.0018)	(0.0003)
25% noise	0.0795	0.0314	0.0401	0.0269	0.0592	**0.0194**
	(0.0000)	(0.0000)	(0.0022)	(0.0021)	(0.0023)	(0.0006)
30% noise	0.0974	0.0409	0.0536	0.0266	0.0758	**0.0203**
	(0.0000)	(0.0000)	(0.0025)	(0.0023)	(0.0017)	(0.0007)

- minimum daily temperatures in the city of Caribou, Maine, USA (MDT) (National Centres for Environmental Information 2021).

We forecast over a ten-week horizon, representing a 50-step forecast for the daily datasets (FX, WTI, NG, VIX, Heating Oil, USD Index, SC ETF, DB, and MDT), and 10 steps for the weekly USIJC dataset. Our comparisons, as previously, include standard econometric linear models, namely the 5-th order autoregressive, AR(5), model and the martingale, or AR(0) model, in which the forecast is the last observed datum. Taking the martingale model as a baseline, we present in Table 5.3 the mean-square errors as the ratio to the martingale model error. We note that the MD-CGAN approach delivers ratios below unity and provides the lowest error of almost all models in this scenario.

5.5.6 Multimodal Posterior Predictions

Finally, we compare the performance of MD-CGAN over varying numbers of mixture components. In all the previous experiments, we set $m = 1$ (hence, the model produced a single predictive Gaussian posterior). Here, we present the

Table 5.3. Ratio of model MSE to martingale, AR(0), baseline model over 10-week forecast horizon.

	AR(5)	SNN	CGAN	MDN	MD-CGAN
USIJC	0.78	0.79	0.77	0.84	**0.73**
EURUSD FX rate	1.91	1.25	0.85	33.48	**0.76**
WTI	0.85	0.89	1.53	1.48	**0.80**
NG	1.01	0.94	1.07	1.13	**0.82**
VIX index	0.71	0.71	0.91	0.77	**0.66**
Heating Oil	0.82	0.93	**0.54**	0.89	0.59
USD Index	1.24	1.34	1.37	0.68	**0.54**
SC ETF	0.89	0.82	0.69	0.81	**0.65**
DB	0.56	0.41	0.66	0.45	**0.38**
MDT	0.65	0.64	0.69	0.62	**0.61**

results for the datasets from Section 5.5.5, with $m \in \{1, 2, 3\}$. We report (negative) log-likelihood measures (as we do not compare against point-value models in this section) and consider one-step forecasts on all the datasets. Table 5.4 presents the performance across datasets for varying numbers of mixture components in the posterior prediction. Figure 5.5 shows the predicted distribution for the test datasets for NG and VIX Index.

We note performance improvement for some datasets for $m > 1$. We do not attempt to infer m, though choosing its value based on performance on a set of cross-validation data would be an option, as would enforcing regularization over the mixture model posterior, through more extensive use of Bayesian inference.

5.6 Conclusion

In this chapter, we present the MD-CGAN model, which offers extensions to the CGAN (Mirza and Osindero 2014) methodology, particularly to allow for GAN inference of a (multimodal) posterior distribution over forecast values. Our approach uses a CGAN-like approach to infer the hyperparameters of a Gaussian Mixture model posterior, and we find that the MD-CGAN approach outperforms other methods on all datasets in which noise is prevalent, including all the financial

Table 5.4. Negative log-likelihood variation with m, with standard deviations in brackets.

	$m = 1$	$m = 2$	$m = 3$
USIJC	-1.01	-1.05	**-1.09**
	(0.02)	(0.04)	(0.02)
EURUSD FX rate	**-1.79**	-0.98	-0.67
	(0.08)	(0.05)	(0.04)
WTI	**-1.75**	-1.24	-1.33
	(0.09)	(0.02)	(0.06)
NG	-1.28	**-1.33**	-1.25
	(0.04)	(0.02)	(0.02)
VIX index	**-1.50**	-1.39	-1.48
	(0.05)	(0.04)	(0.03)
Heating Oil	**-1.63**	-1.37	-1.35
	(0.06)	(0.03)	(0.02)
USD Index	-0.65	-0.83	**-0.86**
	(0.05)	(0.05)	(0.07)
SC ETF	**-1.26**	-1.10	-1.09
	(0.05)	(0.03)	(0.05)
DB	-0.62	**-0.69**	-0.67
	(0.01)	(0.02)	(0.02)
MDT	-0.82	-0.85	**-0.91**
	(0.02)	(0.03)	(0.03)

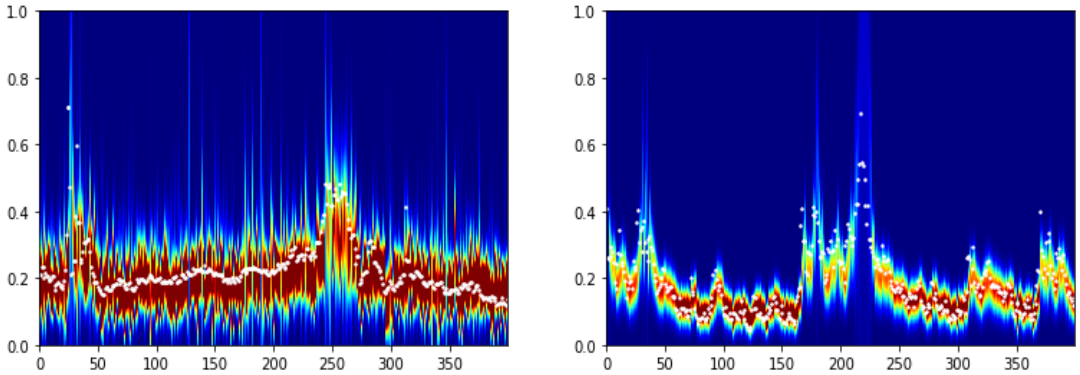

Figure 5.5. Estimated distributions for (left) NG with $m = 2$ and (right) VIX index with $m = 1$ over out-of-sample test datasets. True samples are shown as white dots. Red indicates high data likelihood and blue data low likelihood under the MD-CGAN model.

time series investigated, over long-term forecast horizons. Although the method performs well across all tested datasets, in cases in which strong correlations exist

in the time series and little noise is expected at test time, alternative forecasting methods (including neural networks and a range of parametric models) may perform equally well.

As a GAN model, our approach retains "adversarial" robustness to data perturbations when forecasting. We find this is most notable when noise is extensively present in data. Our method is, thus, particularly well suited to dealing with financial data. Furthermore, MD-CGAN can effectively estimate a flexible posterior distribution, in contrast to standard GAN models which (almost without exception) produce point value outputs.

In summary, the MD-CGAN model combines the advantageous features of both 'flexible probabilistic forecasting and GAN methods. We see this as a particularly useful approach for dealing with time series in which noise levels are variable and significant and for providing robust, long-term forecasts beyond simple point estimates.

Part II

Modelling Multimodality in Multi-Agent Systems

Part II Summary

In this part we focus on multimodality for cooperation in multi-agent systems. Training agents in cooperative settings offers the promise of AI agents able to interact effectively with humans (and other agents) in the real world. Multi-agent reinforcement learning (MARL) has the potential to achieve this goal, demonstrating success in a series of challenging problems. However, whilst these advances are significant, the vast majority of focus has been on the *self-play* paradigm. This often results in a *coordination problem*, caused by agents learning to make use of arbitrary conventions when playing with themselves. This means that even the strongest self-play agents may have very low *cross-play* with other agents, including other initializations of the same algorithm. In Chapter 6 we tackle this problem using a posterior belief over the other agents' strategy. Concretely, we consider the problem of selecting a strategy from a finite set of previously trained agents, to play with an unknown partner. We propose the classic statistical technique, Gibbs sampling, to update beliefs about other agents in a three-step card game.

In Chapter 7 we propose "On-the-fly strategy adaptation" algorithm to solve the ad-hoc coordination problem in cooperative multi-agent systems by adapting agent strategies *on the fly*, using a posterior belief over the other agents' strategy. We propose an extension of the Gibbs sampling method discussed in Chapter 6, to update beliefs about other agents and obtain close to optimal ad-hoc performance. Despite its simplicity, our method is able to achieve strong cross-play with unseen partners in the challenging card game of Hanabi, achieving successful ad-hoc coordination without knowledge of the partner's strategy a priori.

6

Inference in ad-hoc Cooperative Multi-Agent Systems

Contents

6.1 Introduction

In recent years there has been a surge of interest in deep multi-agent reinforcement learning (MARL) (Foerster et al. 2016; Foerster 2018), a paradigm whereby agents

learn, from interaction, to compete or cooperate with other agents. This part focuses on cooperative MARL, which not only offers the potential for agents to learn to interact with other machine learning based agents in the real world, but also provides a possible framework to coordinate with humans.

Many of the most prominent successes in MARL have come in the *self-play* paradigm (Tesauro 1994) where agents optimise their policies by playing a copies of themselves repeatedly without any direct supervision. The self-play setting has been used in both adversarial and cooperative MARL settings. There has been great success with a series of agents achieving breakthroughs on challenging environments such as Go (Silver et al. 2018; Silver et al. 2017b), Chess (Silver et al. 2018; Silver et al. 2017a), Poker (Moravcik et al. 2017; Brown and Sandholm 2018) and Hanabi (Bard et al. 2020; Hu and Foerster 2019; Foerster et al. 2019). While many agents can achieve impressive self-play performance, they often do so by developing arbitrary conventions which hinder their effectiveness when coordinating with others. A key challenge in cooperative multi-agent reinforcement learning is the ability to train robust agents that can play well with others, including humans.

It is shown that in a two-player game, a self-play strategy could converge to a Nash equilibrium (Nash 2016) and could even lead to super-human performance in certain domains (Zinkevich et al. 2007; Bai and Jin 2020; Crandall and Goodrich 2011; Conitzer and Sandholm 2007). However the self-play framework seems more beneficial in an adversarial MARL setting compared to a cooperative one. For one thing, *regret* is not easily defined in a self-play setting. In adversarial zero-sum games the strongest regret is considered by using a more tractable notion of *external regret* (Blum and Mansour 2007; Arora et al. 2012). Such relaxation however is weak in a cooperative setting (DiGiovanni and Zell 2021) and overall regret minimization is not clearly defined and understood in a non zero-sum-game or an n-player setting. In the cooperative MARL framework such as Hanabi for example, despite the strength of these agents when playing with copies of themselves, they often achieve this performance by making use of arbitrary

conventions. For example, in Hanabi, agents will play hint tokens in a specific manner, only interpretable to themselves (Hu et al. 2020). This means many state-of-the-art agents fail to coordinate with an unknown, arbitrary agent at test time (Bard et al. 2020), achieving significantly worse *cross-play* compared to their promising self-play performance. Furthermore, many prominent MARL agents fail to coordinate with independent runs of the same agent algorithm (Hu et al. 2020), suggesting that multiple initializations of the same agent converge to different strategies (described as the ad-hoc coordination problem). We discuss the Hanabi game in further detail in Chapter 7 and we refer the reader to Section 7.4.2 for a brief overview of the rules of the games.

In this chapter we consider the *ad-hoc coordination problem* (Stone et al. 2010), whereby an agent has to coordinate with an unknown partner in either an ad-hoc paradigm or with just a few successive trials. We focus on the problem of selecting a strategy from a finite set of previously trained agents and interacting with an agent whose policy is unknown. In this setting the agents need to actively interact and make inferences about other agents to achieve successful cooperation. Humans do this routinely in their interactions as suggested by the theory of mind (Baker et al. 2017; Premack and Woodruff 1978).

At the core of our approach we use Gibbs sampling (Geman and Geman 1984). Gibbs sampling is a simulation approach to estimate the joint probability distribution by sampling from the conditional probability distributions. By iteratively updating conditional probabilities, thus evaluating beliefs about the other agents' strategies and hidden information, agents are able to form an understanding of the other agents' policies in such a way that they can perform close to optimal performance, subject to their capabilities, even in an ad-hoc environment. Although Gibbs sampling is well-known, as far as we are aware it has not previously been considered for ad-hoc coordination in multi-agent settings. The method is prominently used for Bayesian inference, and has been applied in various areas of machine learning; some recent work includes Ludwig et al. 2020;

Ko et al. 2019; De Sa et al. 2018 in probabilistic graphical models and Alt et al. 2019; L. Yu et al. 2019 in reinforcement learning. The method forms the core of inference in many probabilistic programming languages such as OpenBUGS (Spiegelhalter et al. 2007), JAGS (Plummer et al. 2003), Church (Paige and Wood 2014), PyMC (Patil et al. 2010), and Turing (Ge et al. 2018).

To the best of our knowledge, this is the first work that considers Gibbs sampling in cooperative multi-agent settings.

6.2 Related Work

Bowling and McCracken 2005, and Stone et al. 2010, were among the first to propose the *ad-hoc* teamwork challenge, whereby agents need to coordinate with a previously unknown agent in a task where all agents are capable of individually contributing. More recently, the authors of the Hanabi challenge (Bard et al. 2020) explicitly propose *ad-hoc team play* as one of the primary benchmarks for future progress in cooperative MARL. Concretely, the challenge states that an agent must achieve a high score in the game when coordinating with an unknown partner, with just a few successive trials. The most related work to ours is Canaan et al. 2019, who use the MAP-Elites (Mouret and Clune 2015) algorithm to learn a diverse set of policies, and show it can be used to coordinate across seeds of the same algorithm by making use of an iterative Bayesian Optimization approach (Brochu et al. 2010). However, this approach requires meta-information about the policies in the set, which requires human knowledge and thus does not generalize to other problems.

In other related work in ad-hoc multi agent problems, the MARL agent has access to other agent's previous observed behaviour (Barrett et al. 2017; Peysakhovich and Lerer 2017; Lerer and Peysakhovich 2019; Carroll et al. 2019) or it is assumed that the agents will interact with the same agents interacted with during training (Lowe et al. 2017). It has also been shown it is possible to solve various MARL tasks by learning communication conventions between agents using

communication channels (Sukhbaatar et al. 2016; Mordatch and Abbeel 2018). In our setting the agents have no access to previous behaviour or observations and the are no channels to establish communication conventions.

6.3 Background and Setting

We consider fully-cooperative Markov games. We model this setting with a Decentralized Partially-Observable Markov Decision Process (Dec-POMDP) (Bernstein et al. 2002; Nair et al. 2003), discussed in Section 2.4.2.

The game is fully cooperative, thus agents share the reward $r_t = R(s_t, u_t)$, conditioned on the joint action and the state. The goal is hence to maximize the expected return, $J = \mathbb{E}_\tau R(\tau)$, where $R(\tau) = \sum_t \gamma^t r_t$ is calculated using a discount factor γ, and τ is the trajectory.

As in Foerster et al. 2019, we consider a setting in which the Markov state, s_t, consists of discrete features, f^{s_t}, which are themselves composed of public features, f^{pub,s_t}, and private features, f^{pri,s_t}. The public features are known by all agents and the private features are known by at least one, but not all, the agents. We denote f^{i,s_t} as representing the private features only observed by agent i. In addition, we assume two types of agents in the system; simple agents, A^S, that play a fixed policy, π^S, and complex agents, A^C, which coordinate with the set of simple agents and further have access to the set of policies, $\Pi = \{\pi^1, ..., \pi^n\}$. This provides groups of agents with the ability to reason over the intention of others, similar to the *theory of mind* popularized for understanding human action taking (Baker et al. 2017; Premack and Woodruff 1978).

Similar to the approach in Bard et al. 2013, the portfolio of the policies, Π, is constructed offline. The portfolio consists of a set of policies and for each policy $\pi^i \in \Pi$ played by the other agent, the optimal response policy is $B(\pi^i)$. In the experiments in Part II, the policy set Π provided to our agent, is not optimised. This could be an area of focus for future work that we discuss further in Chapter

8. Further, the function $B(\pi^i)$ maps $\pi^i \in \Pi$ to $\pi^j \in \Pi$. In a more complex setting this assumption could be relaxed, such that instead of Π there is a partner policy set and a response policy set that do not overlap.

6.4 Gibbs Sampling

Gibbs sampling facilitates the estimation of a joint distribution by sampling from the conditional probabilities of that distribution. Gibbs sampling is a useful method when the full joint distribution is intractable or cannot be evaluated or sampled from directly, but the conditional distribution of each variable (or clique of variables) can be evaluated and hence sampled from.

We consider a distribution $p(\mathbf{z}) = p(z_1, ..., z_n)$ that we want to estimate and sample from. In Gibbs sampling we replace the variables at each step with the sample drawn from the distribution of that variable, conditioned on the remaining variables. Therefore, we draw z_i from the distribution $p(z_i \mid \mathbf{z}_{\setminus i})$ and use that sample as the value of z_i for the next iteration, in effect forming a sample-based successive iteration algorithm. Algorithm 2 elaborates the steps of a standard Gibbs sampling method (Bishop 2006).

Algorithm 2 Standard Gibbs Sampling

1: **Initialize:**
$\{z_i : i = 1, ..., n\}$
2: **for** $t = 0, ..., T$ **do**
- Sample $z_1^{t+1} \sim p(z_1 \mid z_2^t, z_3^t, ..., z_n^t)$
- Sample $z_2^{t+1} \sim p(z_2 \mid z_1^{t+1}, z_3^t, ..., z_n^t)$

$\vdots$

- Sample $z_l^{t+1} \sim p(z_l \mid z_1^{t+1}, ..., z_{l-1}^{t+1}, z_{l+1}^t, ..., z_n^t)$

$\vdots$

- Sample $z_n^{t+1} \sim p(z_n \mid z_1^{t+1}, z_2^{t+1}, ..., z_{n-1}^{t+1})$

3: **end for**

As discussed in Chapter 2, Gibbs sampling may be seen as a special case of the Metropolis-Hastings (MH) algorithm. To show this we consider a step of the MH algorithm, for the variable z_i, where the other variables $\mathbf{z}_{\setminus i}$ are fixed. The transition probability from $\mathbf{z}$ to $\mathbf{z}^*$ is $q_i(\mathbf{z}^* \mid \mathbf{z}) = p(z_i^* \mid \mathbf{z}_{\setminus i})$. Given the fact that the other components in the set are fixed and will not change within the step, ${\mathbf{z}^*}_{\setminus i} = \mathbf{z}_{\setminus i}$. Furthermore, $p(\mathbf{z}) = p(z_i \mid \mathbf{z}_{\setminus i})p(\mathbf{z}_{\setminus i})$. The acceptance probability of the MH algorithm in this case will be as follows:

$$\begin{aligned} A(\mathbf{z}^*, \mathbf{z}) &= \frac{p(\mathbf{z}^*)q_i(\mathbf{z} \mid \mathbf{z}^*)}{p(\mathbf{z})q_i(\mathbf{z}^* \mid \mathbf{z})} \\ &= \frac{p(z_i^* \mid {\mathbf{z}^*}_{\setminus i})p(\mathbf{z}^*_{\setminus i})p(z_i \mid {\mathbf{z}^*}_{\setminus i})}{p(z_i \mid \mathbf{z}_{\setminus i})p(\mathbf{z}_{\setminus i})p(z_i^* \mid \mathbf{z}_{\setminus i})} \\ &= 1. \end{aligned} \tag{6.1}$$

We can thus see Gibbs sampling as a special case of the MH algorithm, in which the probability of acceptance is always unity.

6.4.1 Gibbs Sampling for ad-hoc Coordination

In line with the *theory of mind*, one possible approach for coordination in multi-agent ad-hoc problems is the formation of beliefs regarding strategies that are played by the other agents in the system. If agents can accurately identify and understand the strategies that are played by other agents, they can respond with a policy that could achieve the best results. In a partially-observable, multi-agent problem space however, agents need to estimate the joint probability distribution of the other agents' policies and the hidden information, which, by definition, is not available to them. This joint probability distribution is more often than not difficult to infer and thence sample from.

In our ad-hoc coordination setting, the complex agents, A^C, need to estimate the joint probability distribution $P(\pi^S, f^{S,s_t} \mid u_t^S, s_t)$ at every step of the game, where π^S is the simple agent policy, and $f^{S,s_t} \in F^{S,s_t} = \{f^{1,s_t}, ..., f^{n,s_t}\}$. Here, F^{S,s_t} denotes a set of features that A^S could be privy to but are hidden from A^C in state

s_t. For example, in a typical card game, F^{S,s_t} could represent a combination of cards at each step that only A^S can observe and are not openly discarded or on display.

Algorithm 3 Gibbs Sampling for ad-hoc Coordination

1: **Initialize:**
   $\pi_0^S \leftarrow \pi^i$, $\pi^i \in \Pi = \{\pi^1, \ldots, \pi^n\}$
2: **for** $t = 0, ..., T$ **do**
3: **if** it is A^C turn to play **then**
4: Sample $f_{t+1}^{S,s_t} \sim P(f^{S,s_t} \mid \pi_t^S, u_t^S, s_t)$
5: Sample $\pi_{t+1}^S \sim P(\pi^S \mid f_{t+1}^{S,s_t}, u_t^S, s_t)$
6: **end if**
7: **end for**

In order to estimate this joint distribution we utilize Gibbs sampling algorithm in this chapter. We introduce an extended version of this algorithm in chapter 7. The core of the two-step Gibbs sampling algorithm we use here is shown in Algorithm 3.

Noting that all agents observe u_t^S at every step, we may use Bayes' theorem to estimate the distributions in steps 4 and 5 of Algorithm 3 as follows:

$$P(f^{S,s_t}|\pi_t^S, u_t^S, s_t) = \frac{P(u_t^S|f^{S,s_t}, \pi_t^S, s_t)P(f^{S,s_t} \mid \pi_t^S, s_t)}{P(u_t^S|\pi_t^S, s_t)} \tag{6.2}$$

$$= \frac{P(u_t^S|f^{S,s_t}, \pi_t^S, s_t)P(f^{S,s_t} \mid \pi_t^S, s_t)}{\sum_{f^{i,s_t}} P(u_t^S|f^{i,s_t}, \pi_t^S, s_t)P(f^{i,s_t} \mid \pi_t^S, s_t)} \tag{6.3}$$

$$P(\pi^S|f_{t+1}^{S,s_t}, u_t^S, s_t) = \frac{P(u_t^S|\pi^S, f_{t+1}^{S,s_t}, s_t)P(\pi^S \mid f_{t+1}^{S,s_t}, s_t)}{P(u_t^S|f_{t+1}^{S,s_t}, s_t)} \tag{6.4}$$

$$= \frac{P(u_t^S|\pi^S, f_{t+1}^{S,s_t}, s_t)P(\pi^S \mid f_{t+1}^{S,s_t}, s_t)}{\sum_{\pi^i} P(u_t^S|\pi^i, f_{t+1}^{S,s_t}, s_t)P(\pi^i \mid f_{t+1}^{S,s_t}, s_t)} \tag{6.5}$$

6.5 Experiment

The experiment in this chapter is, by design, simple and should be viewed as an exemplar proof of concept. We extend this in Chapter 7, in which we perform more complex experiments.

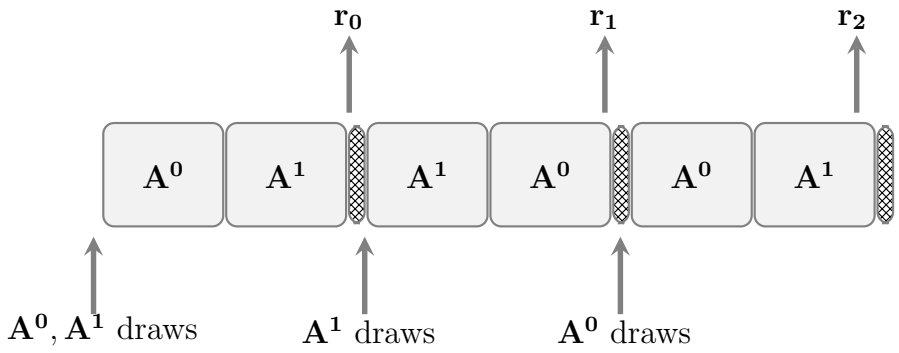

Figure 6.1. *Schematic of the three-step matrix card game. Each square presents the action by each agent (A^0 and A^1). Every time-step consists of one action from each agent to receive a joint reward r_t.*

6.5.1 Three Time-Step Matrix Game for ad-hoc Play

We construct a two-player three-step card game. There are two agents playing the game, A^0 and A^1. Each agent holds only one card at each time-step and the cards have a value of 0 or 1 with a probability of 0.5. Therefore

$$P(\text{Card}_t^i = m) = 0.5,\ m \in \{1, 2\} \quad \forall i \wedge \forall t$$

where Card_t^i is the card value for agent i at time t.

Each agent A^i takes an action u_t^i per time-step. At the initial time-step both agents draw a card followed by agent A^0 taking the first action u_0^0 and agent A^1 taking action u_0^1. After this, agent A^1 draws a new card and makes the next move followed by agent A^0. We continue in this manner for n steps, where n is set to 3 for our experiment. Figure 6.1 illustrates the game.

6.5.2 Reward Matrix Structure

The agents receive a shared reward at each time-step after they both take an action. The shared reward r_t^1 is conditional on the cards and the actions taken by the agents and has the same structure for each time-step. Figure 6.2 shows the reward matrix M^1 that determines r_t^1. It is evaluated at each time-step as follows:

$$r_t^1 = M^1[\text{Card}_t^1][\text{Card}_t^2][u_t^1][u_t^2]$$

The second shared reward per time-step, r_t^2, is conditional on the actions taken by the agents only, and changes at each time-step. Figure 6.3 shows the reward matrix M_t^2, for evaluating r_t^2. The numbers in the reward matrix in Figure 6.3 are generated randomly from a uniform distribution $\mathcal{U}(-15, 15)$. r_t^2 is determined as follows:

$$r_t^2 = M_t^2[u_t^1][u_t^2]$$

Total shared reward per time-step is the sum of the two rewards:

$$r_t = r_t^1 + r_t^2 \tag{6.6}$$

With this structure, the reward is a function of the latest state and joint actions.

6.6 Agent Policies

In order to ensure compatibility and resemblance with real-world ad-hoc play settings, we consider two main attributes for our selection of policies, diversity in the type of policies and diversity in the capability of the agents. To ensure diversity in the type of policies, we aim for a set of diversely trained policies that our agents can use as their strategy. In order to achieve this we use two different deep multi-agent reinforcement learning (MARL) methods, along with two bots with simple strategies. To ensure diversity in capabilities across the agents, we aim for strategies with different *self-play* average rewards. We note that the optimal solution to the ad-hoc cooperative problem, might not generate the absolute best reward for the task, but the best reward conditional on partners' abilities.

Once the strategies are trained, the complex agent has a portfolio composed of four strategies. The simple agent plays one of the four strategies at random. The complex and the simple agent then play together, without any prior knowledge of each other's policies.

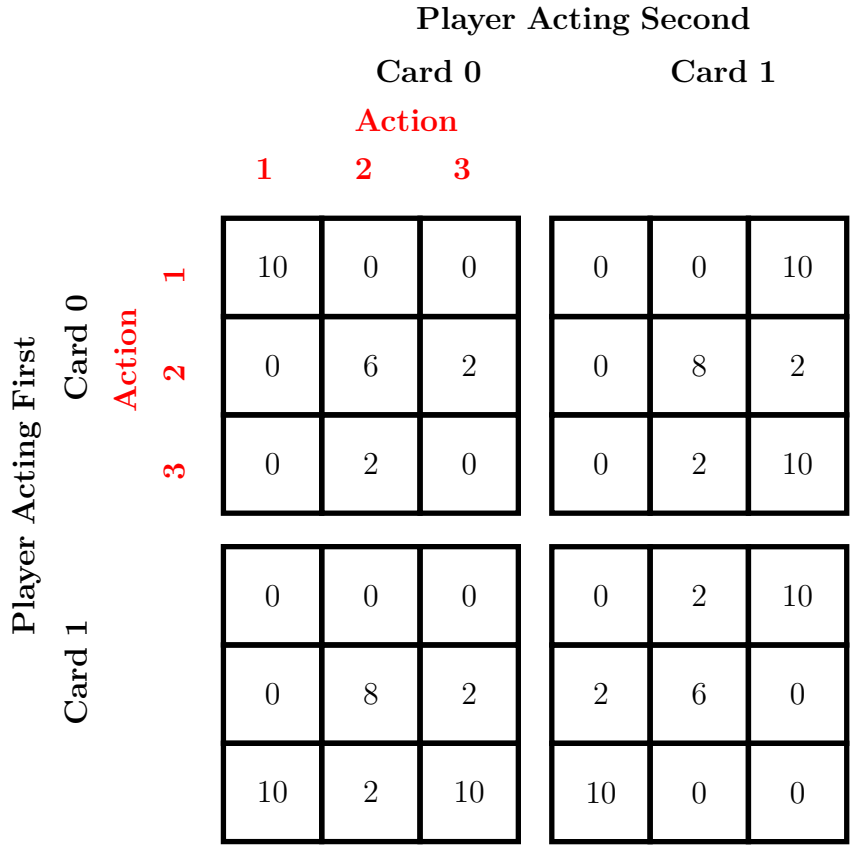

Figure 6.2. Reward matrix M^1, conditional on joint actions and card values for each agent, to evaluate r_t^1.

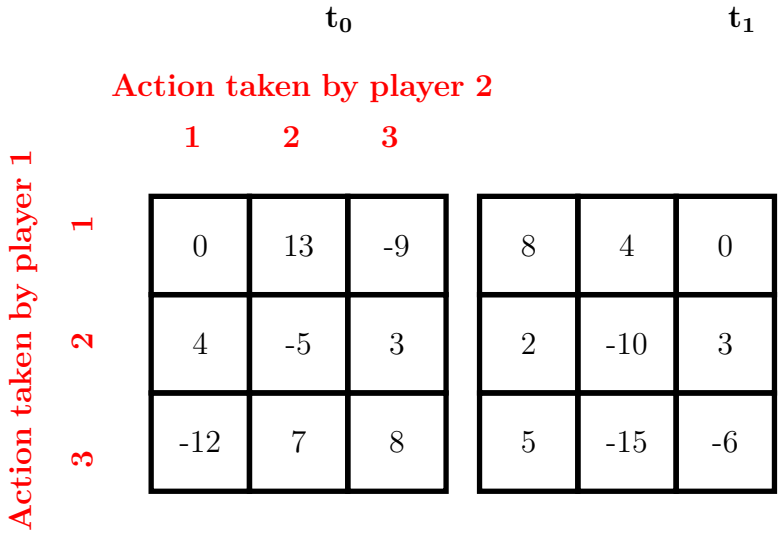

Figure 6.3. Reward matrix M_t^2,conditional on joint actions to evaluate r_t^2 for each time-step.

6.6.1 Deep Multi-Agent Reinforcement Learning Policies

We start by training the two MARL policies in self-play mode . Our chosen MARL policies are a BAD policy and a Standard-MARL Policy discussed in below sections.

BAD Policy

For our first MARL strategies, we train a policy using a method similar to the Bayesian Action Decoder (BAD) introduced by Foerster et al. 2019. The BAD learning method uses an approximate Bayesian update to obtain a public belief conditioned on the actions taken by other agents, in addition to sampled deterministic policies.

The method relies on the proposal from Nayyar et al. 2013, to construct a *public agent* who can generate optimal behaviour conditioned on the public observation f^{pub_t}, and public belief β_t. The public agent's policy π_{BAD} does not observe the private features, but it can instruct the agents what action to take for private observations they receive. At each time step the public agent selects a partial policy $\hat{\pi}$ conditional on public beliefs and public features. Each agent A^i then takes action $u_t^i = \hat{\pi}(f^{i,s_t})$. The public agent then observe action u_t^i and updates its belief.

Similar to the BAD framework, we construct a public agent whose policy π_{BAD}, conditions on the public observations and the public belief. π_{BAD} does not observe the private state, but it provides the agents with the action to take conditional on their private observation.

The agents in our setting pursue a defined known strategy $S_t = \mathbb{E}(\hat{\pi}(f_t^i))$. This is a divergence from the BAD method, where $\hat{\pi}(f_t^i)$ is known by the agents. Our public belief updater is therefore as follows:

$$\begin{aligned} P(f_t^i \mid u_t^i, \beta_t, f^{\text{pub}}, S_t) &= \frac{P(u_t^i \mid f_t^i, \beta_t, f^{\text{pub}}, S_t).P(f_t^i \mid \beta_t, f^{\text{pub}})}{P(u_t^i \mid \beta_t, f^{\text{pub}})} \\ &\propto \mathbb{1}(S_t(f_t^i), u_t^i).P(f_t^i \mid \beta_t, f^{\text{pub}}) \end{aligned}$$

We note that the assumption for all agents to share the exact knowledge of the partial policies $\hat{\pi}(f_t^i)$ is unrealistic for practical applications. We therefore relax this assumption by using S_t as the shared knowledge amongst the agents.

For our BAD policy, the agents assume that the partner's action is optimal with respect to the total reward r_t at each step. We can therefore formulate S_t for our BAD policy as follows:

$$\overline{R}_t(u^i) \mid f_t^i = \mathbb{E}_{f_t^j}(\max_{u^j}(r_t(u^j) \mid u^i, f_t^i, f_t^j))$$
$$S_t(f_t^i) = \arg\max_{u^i}(\overline{R}_t(u^i) \mid f_t^i)$$

We update the belief matrix β_t as follows. Assuming agent i to be the first player and agent j to be the second player, the non-normalised belief matrix for agent j at time t is $\widetilde{\beta}_t^j$, a 3×2 matrix (number of actions $\times$ number of card values) for our experiment. Each row of this matrix $\widetilde{b}^j_{t,u^i,\text{Card}^i}$, represents different plays by agent i at time t, where $u^i \in \{1, 2, 3\}$. Each column represents the potential card held by agent i, where $\text{Card}^i \in \{0, 1\}$. $\widetilde{\beta}_t^j$ is updated as follows:

$$\overline{R}_t(u^i) \mid \text{Card}^i = \mathbb{E}_{\text{Card}^j}(\max_{u^j}(r_t(u^j) \mid u^i, \text{Card}^i, \text{Card}^j))$$
$$\widetilde{b}^j_{t,u^i,\text{Card}^i} = \mathbb{1}(\arg\max_{u^i}(\overline{R}_t(u^i) \mid \text{Card}^i), u^i)$$

We row-normalise matrix $\widetilde{\beta}_t^j$ to compute β_t^j.

Similar to the BAD method, we parameterise $\pi^\theta_{BAD}(u^i \mid \beta_t, f^{\text{pub}}, f^i)$ using deep neural networks, and an actor-critic model to train the policy.

Standard-MARL Policy

For the second MARL policy for our experiment, we train a standard actor-critic framework in self-play mode, using the vanilla gradient policy optimisation method. We refer to this policy as the "Standard-MARL" policy. Unlike the BAD policy the agents do not form any beliefs in the standard model. Foerster et al. 2019 show that this policy performs worse compared to their method. We have similar results for the MARL policies, where the Standard-MARL policy has lower self-play

Table 6.1. Action schedule for Bot 1 and Bot 2

	Bot 1	Bot 2
A^0	3	3
A^1	3	3
A^1	3	1
A^0	1	1
A^0	1	3
A^1	3	3

performance compared to the BAD policy. Our MARL policies therefore have different self-play performance and satisfy the diversity in capability requirement for our intended ad-hoc experiment.

6.6.2 Bot Policies

In addition to the MARL policies, we construct two bot polices. The bots for our card experiment follow action rules that are conditional on the step. For our experiment with two agents there are two actions per time-step, and therefore a total of six actions. Figure 6.1 illustrates the order of actions by each agent, and the action schedule is illustrated in Table 6.1.

6.7 Ad-hoc Game Structure

Our aim is to show coordination in a partially observable ad-hoc environment using Gibbs sampling. In our card game, agents do not see each other's hand and therefore we have a partially observable environment. The agents can play any of the four constructed policies and therefore we have an ad-hoc setting (we assume though that the simple agent plays a random fixed policy, while the complex agent can play a mixed strategy of the policies in the portfolio). In addition, for our experiment we assume that the agents have no prior knowledge of each other's policies, when they initiate the game.

We use Algorithm 2 for an initial run of our experiment, with a batch of thirty two parallel games. For this run, the Gibbs sampler provides us with an estimation of the simple agent's policy per time-step, for each game in the batch, noted as $\hat{\pi}_t^{S,G_n}$, where G_n is the n^{th} game in the batch. We define dominant belief of the simple agent's policy, $\hat{\pi}^S$, as the most frequent policy across all steps and games of a batch. Therefore,

$$\hat{\pi}^S = \text{Mode}(\hat{\pi}_0^{S,G_1}, \hat{\pi}_1^{S,G_1}, \hat{\pi}_2^{S,G_1}, ..., \hat{\pi}_0^{S,G_{32}}, \hat{\pi}_1^{S,G_{32}}, \hat{\pi}_2^{S,G_{32}}) \tag{6.7}$$

The complex agent assumes dominant policy belief $\hat{\pi}^S$, as an estimation of the simple agent's policy π^S. The complex agent then adjusts its policy $\pi^C_{\text{adjusted}} = B(\hat{\pi}^S)$, where $B(.)$ is the best response policy function. We use self-play as a best response policy for this experiment, therefore $\pi^C_{\text{adjusted}} = \hat{\pi}^S$. We note that self-play is not the best response for many problems. However, given that we have a two-player setting, this is a reasonable assumption on this occasion. We discuss best response policy further in Chapter 7. The complex agent then plays with the simple agent using the updated policy, and we report the result for our experiment for an average of 1000 games.

6.8 Results

The results for our experiment are shown in table 6.3. The rows are the complex agent's initial assumption π_0^C, for the simple agent's policy π^S; and the columns are the simple agent's policy. We compare these results to a benchmark, where the complex agent plays a fixed policy π^C, selected at random from the portfolio without Gibbs sampling.

We note that self-play values for each policy, are on the diagonal in the tables. Using Gibbs sampling we can achieve close to self-play performance for all values of π^S, and this is not the case when the method is not used. In particular, the values in bold show how using Gibbs sampling has significantly improved the

Table 6.2. Rewards without Gibbs sampling, where both agents play a fixed game at random (standard errors in brackets).

		π^S			
		[i]	[ii]	[iii]	[iv]
π^C	[i] BAD	27.15 (0.07)	21.27 (0.06)	**17.69** (0.07)	**22.81** (0.10)
	[ii] Standard-MARL	27.58 (0.09)	21.19 (0.07)	**14.90** (0.09)	**21.20** (0.05)
	[iii] Bot 1	27.38 (0.14)	**14.35** (0.07)	27.25 (0.07)	29.83 (0.04)
	[iv] Bot 2	26.97 (0.10)	**14.56** (0.03)	28.21 (0.07)	29.64 (0.06)

Table 6.3. Rewards with Gibbs sampling and for different policy initialisations π_0^C (standard errors in brackets).

		π^S			
		[i]	[ii]	[iii]	[iv]
π_0^C	[i] BAD	27.46 (0.08)	21.89 (0.07)	**26.63** (0.18)	**29.22** (0.08)
	[ii] Standard-MARL	27.69 (0.04)	21.35 (0.07)	**26.89** (0.16)	**28.58** (0.12)
	[iii] Bot 1	28.02 (0.04)	**20.92** (0.10)	27.83 (0.05)	30.35 (0.05)
	[iv] Bot 2	28.01 (0.05)	**20.62** (0.09)	27.90 (0.06)	29.11 (0.10)

performance for those scenarios. In terms of accuracy the complex agent can identify the simple agent accurately, in 74% of all games.

6.9 Discussion and Conclusion

In this chapter, we show how Gibbs sampling is helpful in identifying other agents' policies in a cooperative task. We note that the experiment presented in this chapter is simple by design, with the aim of understanding the efficacy of the Gibbs sampling approach in a cooperative multi-agent setting. Our solution has two components to it, forming a belief about other agents' strategies and

playing a best response policy conditional to this belief. We show that, using the Gibbs sampler, the complex agent is able to improve its performance and achieve close to self-play reward levels.

The Gibbs sampling algorithm facilitates a per-step evaluation of π^S for the complex agent which may vary for each time-step. Using the most frequent per-step evaluation we approximate the simple agent's policy. We incorporate this step in our algorithm in Chapter 7.

Finally in the experiment presented in this chapter, the feature set F^{S,s_t} is small and remains the same throughout the game. This, however, is not the case in many CMAS problems, where F^{S,s_t} varies at each time step. The length of the game is also fixed and remains the same irrespective of actions. This assumption is also unrealistic in many applications, where certain actions can terminate the game early. Further, the complex agent A^C, has full access to all the policies that the simple agent A^S plays. In most real-world scenarios the agents do not know or have access to the exact policy of the other agent. We relax all these assumptions in Chapter 7.

7

On-the-fly Strategy Adaptation Algorithm

Contents

7.1 Introduction

In this chapter we consider the *ad-hoc coordination problem* (Stone et al. 2010), whereby an agent has to coordinate with an unknown partner in an ad-hoc paradigm, either in a single-shot or with just a few successive trials. We achieve this by introducing *on-the-fly strategy adaptation* (OSA) algorithm, which is an

extension of the Gibbs algorithm introduced in Chapter 6. Despite its simplicity, our method is able to achieve strong cross-play with unseen partners in the Slime Volleyball game (Ha 2020) and the challenging card game of Hanabi, achieving successful ad-hoc coordination.

We start this chapter by introducing the OSA algorithm. We then evaluate the performance of the algorithm in our toy problem, the slime volleyball game. We then focus on the multi-player card game of Hanabi (Bard et al. 2020), being one of the most prominent testbeds for cooperative MARL (Foerster et al. 2019; Lerer et al. 2020; Hu and Foerster 2019; Hu et al. 2020; Nekoei et al. 2021). Hanabi is a partially-observable, fully cooperative multi-agent game that consists of 2-5 players. What makes it such a fascinating testbed for MARL is the rich volume of potential strategies. For instance, agents may use completely different conventions, thus two agents may infer different beliefs around the same hint or action taken. The game therefore represents a multi-challenge problem where one can employ the theory of mind. One of the prominent challenges in the Hanabi environment is *ad-hoc team play*, the ability to play well with an arbitrary player never seen before. As of yet, very few agents have been able to achieve strong performance in this setting, and those that have rely on significant domain knowledge (Hu et al. 2021).

We show that, using an extension of the Gibbs algorithm, we are able to achieve close to self-play performance in cross-play situations with ad-hoc players. While there have been many works that consider coordination amongst ad-hoc agents, most of these are limited to agents that are trained by the same model, or are not constrained to a single-shot ad-hoc setting. Our method makes comparisons across a set of diversified models and therefore a diverse set of ad-hoc agents. Finally, since there are no restrictions on the policies that can be included, our method offers scope to be applied to future, stronger base policies.

7.2 Related Work

Related to our challenge, is the recently introduced *zero-shot coordination* (ZSC) problem (Hu et al. 2020). ZSC specifies that an algorithm must be run independently and produce agents with high cross-play performance. Methods for achieving this include making use of the symmetries of the problem (Hu et al. 2020; Parker-Holder et al. 2020a), or making assumptions about prior actions (Hu et al. 2021). ZSC differs from the ad-hoc setting in that it requires agents to train from scratch and produce reproducible policies *within the same algorithm*, but it does not place any requirements on these subsequent agents to play well with unknown agents such as humans. Nonetheless, methods for ZSC have produced more general agents that have successfully transferred to the ad-hoc teamplay setting (Hu et al. 2021).

7.3 On-the-fly Strategy Adaptation

We discussed Gibbs sampling Chapter 6 and detail its selection as a credible method for strategy selection in ad-hoc agent settings.

In a partially-observable, multi-agent problem space however, agents need to estimate the joint probability distribution of the other agents' policies and the hidden information, which, by definition, is not available to them. This joint probability distribution is more often than not difficult to infer and thence sample from.

As discussed in Chapter 6, in a partially-observable, multi-agent problem, agents need to estimate the joint probability distribution of the other agents' policies and the hidden information, which, is more often than not intractable. In order to estimate this joint distribution we utilize an extension of the Gibbs sampling algorithm. However, Gibbs sampling in its standard form can be slow to converge and can only provide per-step evaluations in our problem domain. We therefore develop an extended version of the Gibbs approach which is further discussed in Section 7.3.1. The core of the two-step Gibbs sampling algorithm we use here is shown in Algorithm 3 in Chapter 6.

7.3.1 The OSA Algorithm

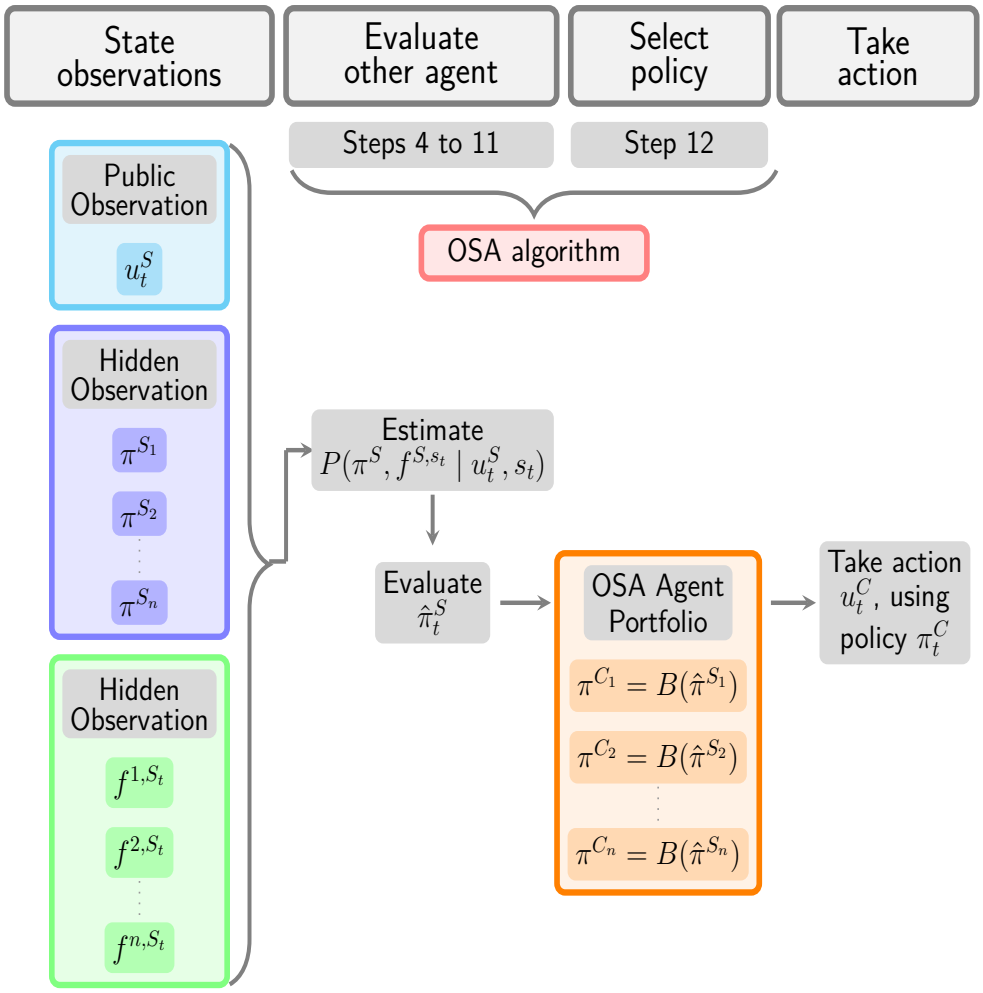

Figure 7.1. *Schematic of the steps of the OSA algorithm at each time step*

We introduce On-the-fly Strategy Adaptation (OSA) as an extension to Algorithm 3. In the OSA approach, the set of A^C perform the Gibbs update steps of Algorithm 3 to evaluate π_t^S, providing per-step evaluation of π^S. Each agent in A^C then uses the most-likely policy evaluation, $\hat{\pi}_t^S$, as its policy belief over A^S at time t. We note therefore, that $\hat{\pi}_t^S = \text{Mode}(\pi_1^S, ..., \pi_{t+1}^S)$. Agents A^C then set their policy π_t^C to $B(\hat{\pi}_t^S)$, where $B(\hat{\pi}_t^S)$ is the optimal response policy for $\hat{\pi}_t^S$.

In addition to A^C evaluating each policy in Π based on the observed action u_t^S, we augment the approach to remove redundant policies from the set. For every

policy $\pi^i \in \Pi$, if $P(u_t^S \mid \pi^i, s_t) \approx 0$, we remove that policy from the policy set (Algorithm 4, steps 6-8). This improves accuracy when Π is a large set and reduces the computational overheads of the approach by removing poor policies early on.

The full OSA procedure is shown in Algorithm 4. We use Equations 6.3 and 6.5 to evaluate steps 4 and 10 in the algorithm. Figure 7.1 presents the schematic of the algorithm.

Algorithm 4 OSA algorithm

1: **Initialize:**
   $\hat{\pi}_0^S \leftarrow \pi^i$, $\pi^i \in \Pi = \{\pi^1, ..., \pi^n\}$
2: **while** the game is ongoing **do**
3: **if** it is A^C turn to play **then**
4: $f_{t+1}^{S,s_t} \sim P(f^{S,s_t} \mid \hat{\pi}_t^S, u_t^S, s_t)$
5: **for** $\pi^i \in \{\pi^1, \ldots, \pi^n\}$ **do**
6: **if** $P(u_t^S \mid \pi^i, s_t) \approx 0$ **then**
7: $\Pi = \Pi \setminus \{\pi^i\}$ ▷ Remove redundant policies
8: **end if**
9: **end for**
10: $\pi_{t+1}^S \sim P(\pi^S \mid f_{t+1}^{S,s_t}, u_t^S, s_t)$
11: $\hat{\pi}_{t+1}^S = \text{Mode}(\pi_1^S, ..., \pi_{t+1}^S)$ ▷ $\hat{\pi}_t^S$ is the most frequent π_t^S
12: $\pi_{t+1}^C = B(\hat{\pi}_{t+1}^S)$ ▷ B: optimal response policy function
13: **end if**
14: **end while**

7.4 Experiments

In this section we test the ability of OSA to find effective ad-hoc coordination policies. To begin, we consider a toy setting, where agents must coordinate in a simple game of volleyball. We then consider the challenging card game of Hanabi, where we seek to achieve coordination between a set of MARL and hand coded agents. We follow the exact same setting discussed in Section 6.3 for our experiments' environments in this chapter.

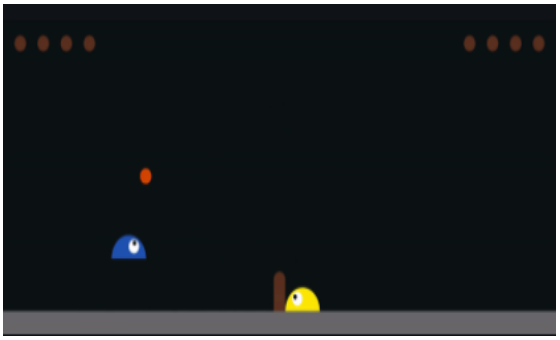

Figure 7.2. Frame-shot of two agents playing *Slime Volleyball*.

In our experiments, we assume that all policies in Π are equally likely for A^C to interact with and therefore A^C assumes a uniform prior distribution over $\pi^i \in \Pi$. We note that this assumption is not necessary for the algorithm and can be relaxed. Further in our first experiment, the Slime Volleyball game F^{S,s_t} is a small set and therefore we perform evaluations of Equations 6.3 and 6.5 for the whole set at each step of the game. In our second experiment, the Hanabi game, F^{S,s_t}, which is the set of all possible hands that A^S can be privy to, is large and we sample from it at each step of the game.

We further assume that self-play is the optimal response policy in our experiments and therefore $B(\hat{\pi}^S_t) = \hat{\pi}^S_t$. The reason we choose self-play as an optimal response policy is that a player's best response has to be a strategy that maximizes its expected reward and ideally will result in a Nash equilibrium, where no other player has an incentive to play a different strategy. As discussed in Chapter 6, in a two-player setting where regret is minimised, although not guaranteed a self-play strategy could still converge to a Nash equilibrium and perform very well in non zero-sum-games or n-player settings (Gibson 2013). We discuss this further in Section 7.6.

7.4.1 Slime Volleyball

We begin by considering the game of volleyball. We modified the recently introduced *Slime Volleyball* (slimevolleygym) (Ha 2020) such that the two "slime" agents play together in a cooperative fashion, seeking to keep the ball in the air for as long as possible (see Fig. 7.2). In addition, we alter the agent observation to make it partially observable, inducing a coordination problem as the agents cannot see the horizontal ball speed when the ball is in their side of the court, thus must infer this information along with the strategy of the other agent.

We consider three types of agents, one trained in self-play mode with deep reinforcement learning (PPO) (Schulman et al. 2017), one trained in self-play mode with a genetic algorithm (GA) (Baeck et al. 2000), and finally a scripted bot (Bot) which takes an action based on simple set of rules.

We assess the performance using two indicators of value, namely the average duration of each game (time) and the total number of passes between the players. Higher values for these indicators imply the agents are coordinating effectively.

In Table 7.1 we show the cross-play matrix for the three agents in our set. For example, the PPO agent can play for on average 26.79 passes, and the GA agent can play for 32.23 passes - yet when they are paired together the numbers are significantly lower (15.22 and 13.37, with numbers differing due to the ordering of the agents). When we use OSA to adapt the policy from the entire set, we are able to achieve a significantly better return (Table 7.2). Not only that OSA outperforms the cross-play matrix, but also for each π^S, the OSA is able to achieve the self-play performance levels.

For the number of passes the results achieved by OSA are higher in all cases and statistically significant using the standard errors.

For the average time, OSA achieves levels that are higher and statistically significant as well, with the exception of π^S playing the PPO policy and the π_0^C the Bot policy. The number of passes are still higher in this case when using

Slime volleyball cross-play with and without OSA. Each cell shows the mean ± standard error for time and number of passes for 1000 games. The first row in each cell is the duration time and the second the number of passes. Larger values imply improved coordination performance.

Table 7.1. Without OSA

		π^S		
		Bot	GA	PPO
π^C	Bot	2654 ± 15	2285 ± 19	2200 ± 19
		24.44 ± 0.19	21.24 ± 0.22	23.58 ± 0.22
	GA	2268 ± 20	2724 ± 14	1259 ± 12
		22.82 ± 0.21	32.23 ± 0.21	13.37 ± 0.19
	PPO	2222 ± 19	1254 ± 12	1979 ± 18
		23.99 ± 0.25	15.22 ± 0.15	26.79 ± 0.32

Table 7.2. With OSA

		π^S		
		Bot	GA	PPO
π_0^C	Bot	2642 ± 16	2730 ± 13	2000 ± 18
		24.41 ± 0.19	32.08 ± 0.20	27.44 ± 0.34
	GA	2677 ± 15	2737 ± 13	1950 ± 19
		24.84 ± 0.19	32.47 ± 0.20	26.35 ± 0.35
	PPO	2645 ± 16	2704 ± 14	1965 ± 18
		24.68 ± 0.19	32.20 ± 0.21	26.72 ± 0.34

OSA. The reason for the higher average time for the cross-play compared to OSA is that the Bot policy is a much slower strategy compared to the PPO and the Bot agent keeps the ball in his court for a longer duration compared to the PPO agent. So while the average time is higher for the cross-play without OSA, the agents cooperation in passing the ball to each other is less.

7.4.2 Hanabi Experiments

We now consider ad-hoc coordination in the challenging Hanabi learning environment (Bard et al. 2020), where we consider a wide range of agents. We consider coordination amongst the following: Rainbow (Hessel et al. 2018), hand coded bots from the Hanabi Open Agent Dataset (HOAD) (Sarmasi et al.

2021), Valuebot, Holmesbot, Iggi and Piers, and Multi Agent Proximal Policy Optimization (MAPPO) (C. Yu et al. 2021). Notably, this includes both on policy (MAPPO) and off policy (Rainbow) deep reinforcement learning approaches, as well as scripted bots, providing a diverse range of agents. We train MAPPO agents using a multi-agent extension of the PPO; and for training the Rainbow policy we use the DQN algorithm. For the MAPPO model, we use two separate trained policies with different seeds.

With regards to the bot strategies the Valuebot plays cards that it has enough knowledge to know that they are playable, and gives hints about the cards that are valuable. It also prefers to discard the oldest cards first. Holmesbot is similar to Valuebot, but uses the life tokens and additional inference to strategize (Sarmasi et al. 2021). Iggi plays cards that it has enough information about as well and prefers number hints over color hints. Unlike the Valuebot it prefers discarding unplayable cards first and before the oldest card. Piers is similar to Iggi but also uses the life tokens and additional rules in order to strategize and avoid discarding the cards that are valuable (Sarmasi et al. 2021; Canaan et al. 2020). Therefore there is a positive correlation between Valuebot and Holmesbot strategies on one hand, and Iggi and Piers strategies on the other hand.

The Hanabi Game

A brief description of the game and rules are as follows. There are five cards in each player's hand. There are five colors; for each color, card values are as follows: one 5, three 1s, and two cards of ranks 2,3 & 4, providing 10 cards per color and a total of 50 cards in a deck. The players can observe the hands for all other players but not their own. The players need to cooperate by exchanging information using hints. The latter can help players make a decision as to which card to play. The goal of the game is to make a 'firework' for each color, consisting of all the card ranks of that color. Five fireworks can hence be made and the maximum score for each game is 25. There are 20 actions possible for each player

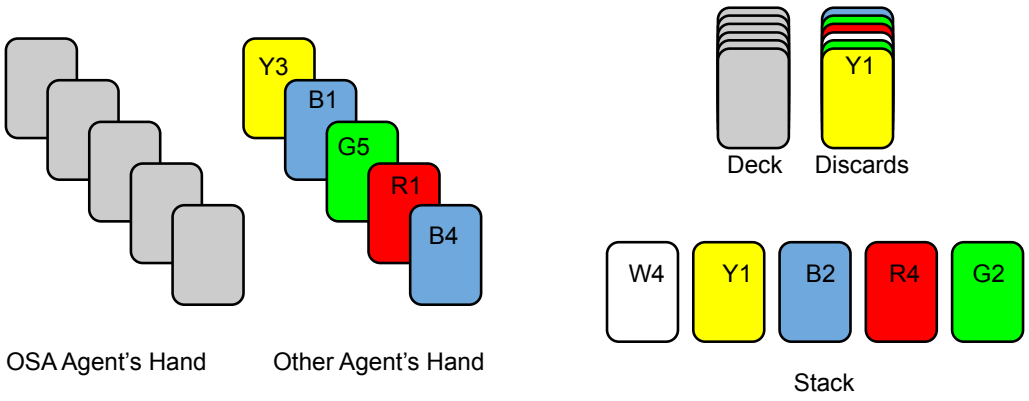

Figure 7.3. Example of a two-player Hanabi game.

at each turn; a player can either play or discard one of the 5 cards they are holding or they can hint to the rank or the color of the other players' holdings. The game begins with eight available information tokens and three life tokens. The game ends immediately if all life tokens are gone. The game can also come to an end if all fireworks have been successfully made, or after a player draws the last card from the deck and every player has taken one final turn. An example of a two-player Hanabi game is shown in Figure 7.3.

Using OSA for Hanabi

We focus on two-player version of Hanabi, consisting of agents A^C and A^S. The hidden information from A^C at each step is its own hand H_t^C, which is known by A^S. Therefore A^C attempts to estimate the joint probability distribution $P(\pi^S, H_t^C \mid u_t^S, s_t)$ at each step using OSA (Algorithm 4), where $f^{S,s_t} = H^{C,s_t}$. We now evaluate the performance of agents in a variety of settings, with and without OSA. All evaluations are averaged over 1000 games. Further details on conducting the experiments are in Appendix A.

Ad-hoc team play

As with the slime volleyball setting, we begin by considering the cross-play matrix of the agents in our set, which we show in Table 7.3. As can be seen, compatibility is non-trivial, for example Rainbow achieves 22.01 in self play, but less than one point with Holmesbot or Valuebot. In addition, even the same algorithm may not repeatedly discover compatible strategies, the MAPPO agents both perform drastically worse together than when partnered with a copy of themselves.

To test OSA, we conduct two sets of Hanabi experiments. In the first Hanabi experiment agent A^S's policy is included in the A^C's policy set, therefore $\pi^S \in \Pi$. In the first experiment we have a fixed set policies for all experiments. The results are shown in Table 7.4. As we see, when the agent is included in the set, OSA is able to recover performance close to self-play. Even in challenging settings, for example being initialized with a Holmesbot agent and playing with Rainbow, the algorithm still achieves a score of 19.54.

Next we consider a more challenging setting, where A^C does not have A^S's policy, $\pi^S \notin \Pi$. In this second experiment, for each evaluation we exclude π^S from our set. The results are shown in Table 7.6. As we see, despite not containing the optimal policy, A^C still successfully plays with A^S using the policy in its set that is most correlated with π^S for majority of the games. Comparisons for both experiments, which both contain statistically significant gains are shown in Table 7.7.

k-shot coordination

All experiments performed in the previous section are single-shot ad-hoc coordination games in which an agent A^C has no knowledge of agent A^S's policy, and has not played with A^S previously. In this section we define a k-shot ad-hoc game whereby A^C plays with A^S a total of k times in the second Hanabi game ($\pi^S \notin \Pi$). We note that, in the first Hanabi experiment, the agents already achieve close to self-play performance levels in a single-shot ad-hoc setting.

Mean rewards for ad-hoc play between agents A^C and A^S, when the simple agent's policy is part of the complex agent's policy set. Standard Errors are in Table 7.7.

Table 7.3. Without OSA

		π^S						
		[i]	[ii]	[iii]	[iv]	[v]	[vi]	[vii]
π^C	[i] MAPPO-1	23.99	15.22	0.00	3.91	8.87	9.45	0.08
	[ii] MAPPO-2	16.41	23.41	0.00	8.16	8.05	16.3	0.06
	[iii] Holmesbot	0.00	0.02	15.76	1.96	0.32	0.19	17.44
	[iv] Iggi	4.50	7.96	2.06	16.97	12.75	5.17	4.21
	[v] Piers	9.01	8.41	0.37	13.24	17.17	8.64	1.17
	[vi] Rainbow	8.90	15.66	0.25	4.56	8.58	22.01	0.63
	[vii] Valuebot	0.04	0.06	17.49	3.58	0.99	0.39	18.5

Table 7.4. With OSA

		π^S						
		[i]	[ii]	[iii]	[iv]	[v]	[vi]	[vii]
π_0^C	[i] MAPPO-1	23.21	21.72	14.06	15.99	16.47	20.08	16.13
	[ii] MAPPO-2	23.07	22.07	13.60	15.96	16.51	20.12	15.71
	[iii] Holmesbot	22.72	21.05	14.00	16.02	16.29	19.54	16.51
	[iv] Iggi	23.05	21.89	13.35	16.14	16.43	20.20	16.12
	[v] Piers	23.03	22.03	13.62	16.30	16.30	19.92	15.92
	[vi] Rainbow	22.97	21.93	13.24	16.16	16.24	20.16	15.90
	[vii] Valuebot	22.70	21.47	14.67	16.04	16.23	19.83	16.43

For the k-shot games, agent A^C begins the first game by using OSA and continues until a positive reward is achieved. If the agents achieve a positive reward in the m^{th} game of the k games, A^C uses $\pi_T^{C,m}$ for the subsequent games, where $\pi_T^{C,m}$ is A^C's last step policy of the m^{th} game using the OSA method. In short A^C plays as follows:

$$\pi^{C,m} = \begin{cases} \pi_T^{C,m-n}, & \text{if } (R^{m-n} > 0) \wedge (m > n \geq 0) \\ \pi_t^{C,m}, & \text{otherwise} \end{cases} \tag{7.1}$$

where R^{m-n} is the reward for game $m-n$. Table 7.8 shows the results for k-shot games, where $k \in \{0, 1, 4\}$, in the second experiment, in which $\pi^S \notin \Pi$. These results are compared with the maximum cross-play scores of A^C policies

*Mean rewards for ad-hoc play between agents A^C and A^S, when the simple agent's policy **is excluded** from the complex agent's policy set. Standard Errors are in Table 7.7.*

Table 7.5. Without OSA

		π^S						
		[i]	[ii]	[iii]	[iv]	[v]	[vi]	[vii]
π^C	[i] MAPPO-1		15.22	0.00	3.91	8.87	9.45	0.08
	[ii] MAPPO-2	16.41		0.00	8.16	8.05	16.3	0.06
	[iii] Holmesbot	0.00	0.02		1.96	0.32	0.19	17.44
	[iv] Iggi	4.50	7.96	2.06		12.75	5.17	4.21
	[v] Piers	9.01	8.41	0.37	13.24		8.64	1.17
	[vi] Rainbow	8.90	15.66	0.25	4.56	8.58		0.63
	[vii] Valuebot	0.04	0.06	17.49	3.58	0.99	0.39	

Table 7.6. With OSA

		π^S						
		[i]	[ii]	[iii]	[iv]	[v]	[vi]	[vii]
π_0^C	[i] MAPPO-1		12.37	13.11	9.83	9.52	11.28	13.85
	[ii] MAPPO-2	10.74		13.31	10.18	9.90	11.44	14.02
	[iii] Holmesbot	10.20	10.80		10.01	9.62	11.07	14.58
	[iv] Iggi	10.94	12.45	13.19		10.02	11.88	14.52
	[v] Piers	10.26	12.12	13.33	9.92		11.54	14.33
	[vi] Rainbow	10.57	12.35	13.26	10.38	10.16		14.36
	[vii] Valuebot	9.95	11.29	13.65	9.87	10.28	10.94	

against π^S. In the 4-shot game, for all policies, the agents achieve rewards close to this maximum level.

Comparison with a Common Best Response to a Population Policy

In this section we compare the performance of the OSA algorithm with a common best response (CBR) to a population of agents policy. The CBR policy trains on all seven strategies using the MAPPO optimisation method. Table 7.9 shows the performance of this policy compared to the OSA algorithm in a single-shot game for both experiments. Both OSA experiments perform significantly better compared to the CBR Policy.

Table 7.7. Mean ± standard error of rewards for A^C playing each of the 7 policies in the two Hanabi experiments.

	$\mathbb{E}_\tau R(\tau)$			
	$\pi^{\mathbf{S}} \in \mathbf{\Pi}$		$\pi^{\mathbf{S}} \notin \mathbf{\Pi}$	
	Without OSA	With OSA	Without OSA	With OSA
MAPPO-1	8.98 ± 0.12	22.96 ± 0.05	6.47 ± 0.11	10.44 ± 0.12
MAPPO-2	10.15 ± 0.12	21.74 ± 0.06	7.94 ± 0.12	11.90 ± 0.13
Holmesbot	5.13 ± 0.10	13.79 ± 0.10	3.36 ± 0.09	13.31 ± 0.10
Iggi	7.48 ± 0.09	16.09 ± 0.04	5.90 ± 0.09	10.03 ± 0.10
Piers	8.10 ± 0.10	16.35 ± 0.04	6.59 ± 0.10	9.92 ± 0.11
Rainbow	8.87 ± 0.11	19.98 ± 0.07	6.69 ± 0.10	11.36 ± 0.11
Valuebot	6.01 ± 0.10	16.10 ± 0.07	3.93 ± 0.09	14.28 ± 0.09

Table 7.8. Mean ± standard error of rewards in k-shot games. A^C plays each of the 7 policies using OSA and $\pi^S \notin \Pi$.

	$\mathbb{E}_\tau(R(\tau) \mid k)$			$\max_{i=0}^{n}(\mathbb{E}_\tau R_i)$
	$k=0$	$k=1$	$k=4$	
MAPPO-1	10.44 ± 0.12	12.84 ± 0.12	14.52 ± 0.11	16.41 ± 0.11
MAPPO-2	11.90 ± 0.13	13.76 ± 0.12	14.86 ± 0.11	15.66 ± 0.11
Holmesbot	13.31 ± 0.10	16.45 ± 0.08	17.33 ± 0.07	17.49 ± 0.07
Iggi	10.03 ± 0.10	10.76 ± 0.10	11.05 ± 0.10	13.24 ± 0.09
Piers	9.92 ± 0.11	10.86 ± 0.11	11.43 ± 0.10	12.75 ± 0.09
Rainbow	11.36 ± 0.11	13.45 ± 0.11	14.21 ± 0.10	16.30 ± 0.10
Valuebot	14.28 ± 0.09	16.72 ± 0.08	17.24 ± 0.07	17.44 ± 0.07

7.5 Hanabi Experiment Analysis

7.5.1 Complex Agent Policy Distribution

In order to better understand the strength and weakness of the method, we assess the distribution of the policies that A^C plays in coordination with A^S for each Hanabi experiment. Figure 7.4 depicts the distribution for π_T^C in the first Hanabi experiment ($\pi^S \in \Pi$) in which π_T^C is agent A^C's policy at the end of each game.

Table 7.9. Mean ± standard error of rewards for the CBR Policy and the two OSA experiments

	CBR Policy	OSA ($\pi^S \in \Pi$)	OSA ($\pi^S \notin \Pi$)
MAPPO-1	5.36 ± 0.12	22.96 ± 0.05	10.44 ± 0.12
MAPPO-2	6.03 ± 0.13	21.74 ± 0.06	11.90 ± 0.013
Holmesbot	1.11 ± 0.07	13.79 ± 0.10	13.31 ± 0.10
Iggi	2.38 ± 0.08	16.09 ± 0.04	10.03 ± 0.10
Piers	3.67 ± 0.11	16.35 ± 0.04	9.92 ± 0.11
Rainbow	4.28 ± 0.12	19.98 ± 0.07	11.36 ± 0.11
Valuebot	2.20 ± 0.06	16.10 ± 0.07	14.28 ± 0.09

The blue bars indicate the distribution over all games, irrespective of reward values at the end of each game. The red bars indicate the distribution only for the games in which rewards were greater than zero. In the first experiment, with the exception of the Holmesbot and Valuebot strategies, agent A^C has close to perfect accuracy in identifying π^S and playing the policy with A^S. In the case of Holmesbot and Valuebot strategies, the two policies have a high correlation and excellent cross-play scores. Whilst OSA identifies π^S accurately in majority of the games, at times it identifies Holmesbot as Valuebot and vice versa. We note, however, that for these two policies the games that are wrongly identified as the other type still yield a positive reward with very few exceptions.

Figure 7.5 depicts the distribution of π_T^C in the second Hanabi experiment, in which $\pi^S \notin \Pi$. In this experiment, across all seven policies, agent A^C plays the policy with the highest cross-play score with π^S for the majority of games. In this experiment we note that A^C also plays, on occasion, other policies that have a correlation with π^S. We further note that in many games that achieved zero scores, π_T^C was still the policy with the closest cross-play score to π^S.

In summary, in the first Hanabi experiment, π^S is the dominant policy selected by A^C to play A^S, and in the second Hanabi experiment, where $\pi^S \notin \Pi$, the

policy with the highest cross-play score with π^S is the dominant strategy for A^C to play with A^S.

7.5.2 Complex Agent Time Distribution of Final Policy

To get a better understanding of the convergence of the OSA algorithm, we define the convergence step t^*, as follows:

$$t^* = \inf\{t' \mid \pi^C_{t'+n} = \pi^C_T,\ \forall t' \leq n \leq T \wedge n \in \mathbb{N}_0\}. \tag{7.2}$$

To put it simply, t^* is the time step after which π^c_t remains the same and equal to π^C_T.

Figures 7.6 and 7.7 show the distribution of t^* for the two Hanabi experiments. For both experiments for most encounters the algorithm converges in less than ten time-steps. For the Holmesbot and Valuebot in the first experience, the convergence is slower compared to the other policies. This is expected as these two policies are the most correlated amongst the set. However, given that $\pi^S \notin \Pi$ in the second experiment, this late convergence does not occur for these policies in this case.

7.5.3 Ablation Experiments

We perform ablation experiments on the OSA algorithm to better understand the impact of the additional steps to the Gibbs sampling method. Tables 7.10 and 7.11 show the results for the ablation experiments. Our first additional step, is the insertion of the Mode function in step 11 of the algorithm. This improves the accuracy of the evaluations of the partner's policy $\hat{\pi}^S$, as per-step evaluations of the Gibbs sampling algorithm are not always accurate. The first two columns in Tables 7.10 and 7.11 show the improvement in performance by adding this step.

Our second addition to the Gibbs sampling algorithm, is to remove redundant policies early on in the game. Steps 6 and 7 in the algorithm achieve this. This serves two purpose: First performance is improved, given that the removal of these polices early on, lowers the chance for the algorithm to choose them in the later stages of the game; and second we improve computational efficiency by decreasing

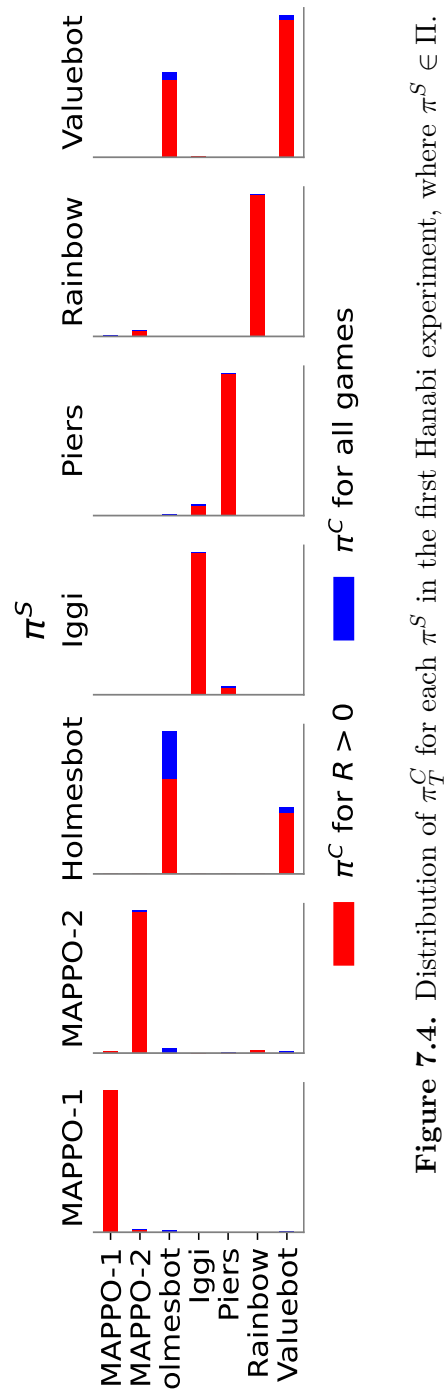

Figure 7.4. Distribution of π_T^C for each π^S in the first Hanabi experiment, where $\pi^S \in \Pi$.

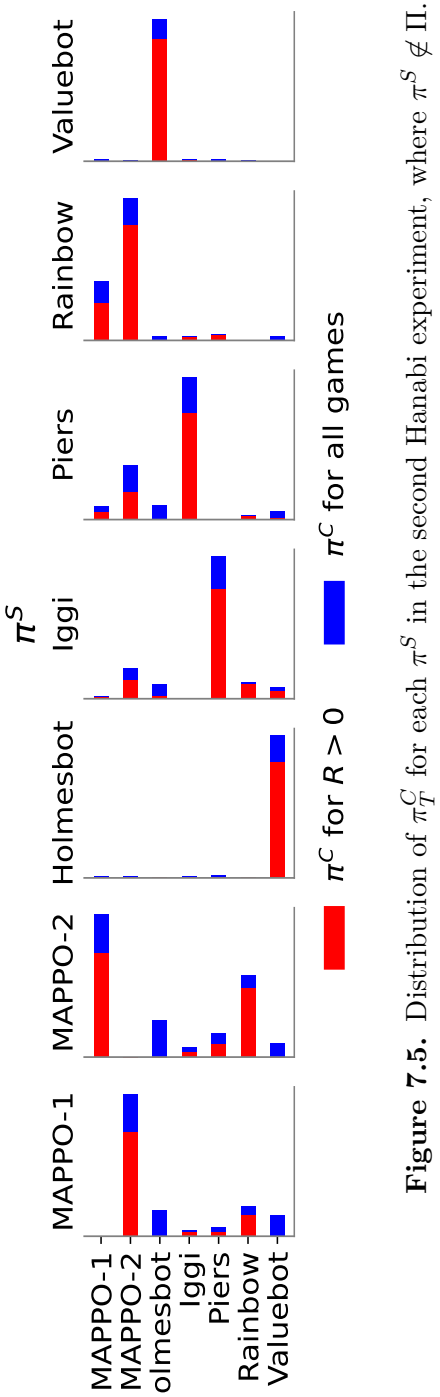

Figure 7.5. Distribution of π_T^C for each π^S in the second Hanabi experiment, where $\pi^S \notin \Pi$.

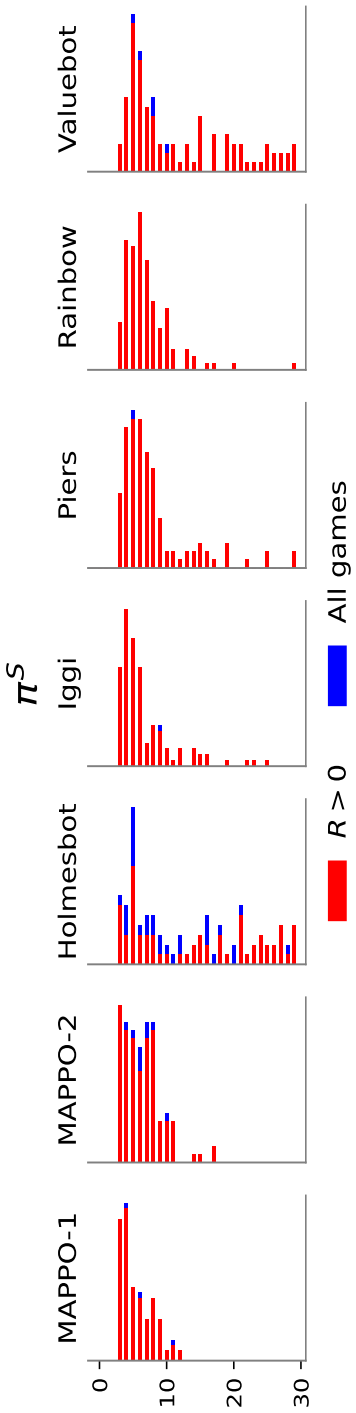

Figure 7.6. Time distribution of final Policy in the first Hanabi experiment, where $\pi^S \in \Pi$.

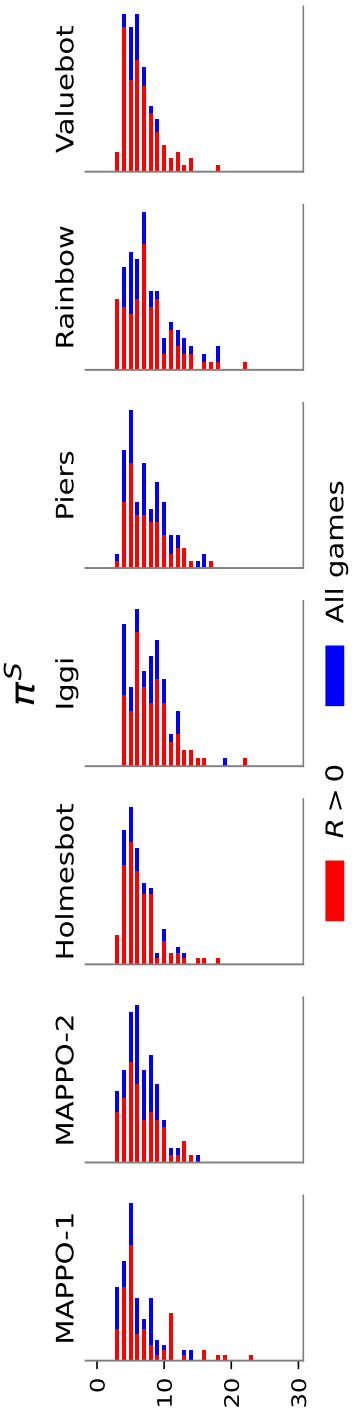

Figure 7.7. Time distribution of final Policy in the second Hanabi experiment, where $\pi^S \notin \Pi$.

Table 7.10. Mean ± standard error of rewards for the OSA policy excluding "removing redundant policies steps" (6, 7), and "mode step" (11), for the first Hanabi experiment, where $\pi^S \in \Pi$

	OSA Excl. Steps 6,7, 11	OSA Excl. Steps 6,7	OSA
MAPPO-1	11.54 ± 0.14	19.64 ± 0.12	22.96 ± 0.05
MAPPO-2	16.10 ± 0.12	19.00 ± 0.11	21.74 ± 0.06
Holmesbot	3.91 ± 0.11	9.46 ± 0.14	13.79 ± 0.10
Iggi	12.20 ± 0.10	13.81 ± 0.09	16.09 ± 0.04
Piers	11.82 ± 0.11	12.88 ± 0.11	16.35 ± 0.04
Rainbow	11.83 ± 0.11	15.54 ± 0.12	19.98 ± 0.07
Valuebot	7.84 ± 0.19	11.88 ± 0.12	16.10 ± 0.07

Table 7.11. Mean ± standard error of rewards for the OSA policy excluding "removing redundant policies steps" (6, 7), and "mode step" (11), for the second Hanabi experiment, where $\pi^S \notin \Pi$

	OSA Excl. Steps 6,7, 11	OSA Excl. Steps 6,7	OSA
MAPPO-1	2.15 ± 0.08	6.21 ± 0.12	10.44 ± 0.12
MAPPO-2	3.99 ± 0.11	6.44 ± 0.13	11.90 ± 0.13
Holmesbot	1.45 ± 0.07	3.92 ± 0.10	13.31 ± 0.10
Iggi	7.60 ± 0.10	7.68 ± 0.11	10.03 ± 0.10
Piers	6.39 ± 0.12	7.12 ± 0.11	9.92 ± 0.11
Rainbow	3.18 ± 0.08	5.95 ± 0.11	11.36 ± 0.11
Valuebot	1.05 ± 0.05	6.63 ± 0.12	14.28 ± 0.09

the size of the portfolio policy early on in the game. The second and third columns in Tables 7.10 and 7.11 show the improvement in performance by adding this part.

7.6 Discussion

In our Hanabi experiments we have included a set of policies from a diversified set of models. We note that agent A^C's policy set, Π, is not restrictive and can include other models. For example the strategies presented in Hu et al. 2020 and Hu et al.

2021 are shown to play the Hanabi game well with humans in ZSC settings; these policies could be added to agent A^C's policy set. The OSA algorithm is a method that is inclusive of all strategies and models and not in direct comparison with each. It may also be possible to achieve strong ad-hoc coordination by *learning* a set of diverse strategies and then using OSA at test time.

Further due to the policy-removal addition to the Gibbs sampling algorithm (steps 5 to 9 in Algorithm 4), the method is able to focus the policy set to the most relevant subset early on in the game, thereby scaling well as the number of policies in Π are increased. In addition, performance is further improved in k-shot games with the OSA method and agents can achieve performance close to the best cross-play in the policy set, when π^S is not in Π. We note that this improvement can not be achieved for strategies that play a fixed policy in every play of the k-shot game.

We further note that, in the current format, the method treats all policies in the policy set Π equally and there are no portfolio optimization steps for policy selection. We see this as a possible extension.

Regarding the limitations of OSA, A^C's performance is dependant on the policies in Π and π^S. If none of the policies in Π are positively correlated with π^S, OSA will not perform well. A well diversified portfolio of policies will mitigate this issue to some extend. This further implies that if π^S is a random play, the algorithm will not have any added value.

Further we have not investigated in detail the best response policy in this work. The OSA algorithm performs two main steps. In the first step the algorithm evaluates the other agent by estimating the joint probability, $P(\pi^S, f^{S,s_t} \mid u_t^S, s_t)$, discussed in Section 6.4.1. The second step is to determine the best response policy after this evaluation. We chose self-play in our experiments. However it is possible that self-play is not the optimal response policy for certain MARL settings. While there is a lot of empirical evidence for the success of self-play strategies, more theoretical work is needed to better understand their effectiveness specifically in non zero-sum-games. Even though the two-player self-play game

could converge to a Nash equilibrium this is not a guaranteed outcome in non zero-sum-games specially in games with partial observability. We also note that finding a Nash equilibrium itself is a minimal requirement for a satisfactory result in a multi-agent system. While it could be the only optimal solution in some games, it is possible to have multiple Nash equilibria and in that case for a good performance it is reasonable to require the game to converge to a Pareto optimal Nash equilibrium (Conitzer and Sandholm 2007), as in this case simply achieving a Nash equilibrium could be sub-optimal. Further in our experiments the games are symmetric with respect to the agents and no agent has an advantage compared to others. Choosing self-play as a best response policy in an environment where agents have non-symmetric resources could lead to an undesirable outcome.

7.7 Conclusion

We present On-the-fly Strategy Adaptation (OSA), a novel approach for coordination between ad-hoc agents across a set of diverse models. Despite its simplicity, the algorithm achieves impressive performance while scaling gracefully. We have further provided average rewards for both ad-hoc and k-shot ad-hoc games and shown that performance improves in k-shot games. We believe OSA could be particularly impactful in partially observable cooperative settings, which feature in many real world settings.

We note that the fully cooperative ad-hoc coordination problem is a challenging one that has attracted a lot of attention in recent years and the Hanabi game has been gaining momentum as its benchmark for testing. Bard et al. 2020 discusses the problem of ad-hoc play across strategies and their poor performance. To the best of our knowledge no good solutions have been offered to date for this particular problem where agents have no knowledge or previous experience of playing together in the game of Hanabi. We have shown that despite its simplicity, OSA performs very well in this setting. Due to shortage of solutions for this

particular coordination problem there are currently no clearly defined benchmarks, but if we consider "best response to a population" policy as a best response benchmark policy, we show that OSA significantly outperforms this benchmark.

8

Conclusions and Future Work

Contents

8.1 Conclusions & Summary

In this thesis we have introduced putative solutions to use multimodality across a variety of prediction and coordination problems in machine learning. Our preferred approach uses extensions of finite mixture models, which we show provide significant benefits without loss of computational tractability. We have argued throughout that *uncertainty* is a critical, yet often overlooked, component of model value. We showed that uncertainty estimates, using mixture model methods, far outperform well-known alternatives (such as Monte Carlo dropout) - not just in terms of raw uncertainty calibration, but also in data efficiency, speed to convergence and test-time errors. In an era of big data and big models, measurement of model

uncertainty is not nearly prevalent enough. In most application domains, well-calibrated measures of accuracy, risk and model confidence are critical if we are to make rational, evidence-based decisions with our algorithms.

In Part I, we focus on multimodality in prediction of time series data sets, and propose a set of probabilistic models that can estimate multimodal posterior distributions using Gaussian mixture models. In Chapter 3 we introduce the MiDGaP model, an extension of standard Gaussian processes within a mixture-model framework. We compare our approach to other probabilistic methods, in particular the mixture of experts model, and show that MiDGaP outperforms these approaches. In Chapter 4, we introduced the Deep-MDN model, an extension of the MDN method originally proposed in Bishop 1995, whereby we estimate the parameters of the mixture density posterior using both CNN and LSTM networks. We illustrate how our approach is computationally more efficient, compared to the prevalent paradigm of Monte Carlo dropout. In Chapter 5, we introduced MD-CGAN, a further extension of the Deep-MDN model, which uses a CGAN-like approach to infer the hyperparameters of a multimodel model posterior. We show how the method is particularly valuable when deployed in high-noise data.

In Part II, we tackled the problem of coordination in cooperative multi-agent systems (CMAS). In Chapter 6, we use a Gibbs sampling paradigm to formulate a posterior belief over agent strategies. In Chapter 7 we introduced the OSA algorithm, an extension of the Gibbs sampling method for CMAS. We test OSA on the challenging game of Hanabi and show how our method can achieve strong cross-play even with unseen partners, so achieving successful ad-hoc coordination.

We conclude that, for most problems tackled by machine learning, uncertainty estimation is not just a good thing, but imperative. Further, measures of uncertainty need not be restricted to a single measurement of variance and a unimodal target distribution. We need to accept, and work with, the heterogeneity present in most challenges that are representative of real-world settings. We believe that entertaining the possibility of multiple credible solution modes and

the use of multimodal components in our models can add to the performance and resilience of regression, decision making and coordination methods. Probability theory is the pervasive mechanism of choice for such representations of uncertainty and, despite its apparent extra complications, allows for excellent calibration against empirical test data.

In a world of big data, it is vital to propose model extensions which do not add to computational overload. There is thus a balance to be struck between accuracy, complexity of solution and computational overheads. For practical approaches most suitable for real-world settings, we argue that mixture-style models offer an effective sweet spot, balancing effective training, low deployment overheads and significantly embellished performance.

8.2 Future Work

8.2.1 Part I

We concluded Part I with the presentation of the MD-CGAN model in Chapter 5. In the experiments in that chapter, "adversarial" perturbations of the observed test data are due to stochastics in the data, rather than carefully planned malicious attacks. Assessing the performance of the model in data with optimized adversarial attacks is an interesting avenue of future research that might be of value in finance, where "spoofing" is often prevalent.

Further as discussed we do not attempt to infer m, though choosing its value based on performance on a set of cross-validation data would be an option, as would enforcing regularization over the mixture model posterior, through more extensive use of Bayesian inference. We leave these extensions for future research.

Finally in the experiments performed in Chapter 5, we do not optimise the look-back window nor the structure of the MD-CGAN model for each data set. Optimisation of look-back parameters and performing neural architecture search

remains an interesting avenue for future research, likely to improve performance on a data set by data set basis.

8.2.2 Part II

We concluded Part II with the presentation of the OSA algorithm. Considering potential extensions to improve performance, we note that the complex agent's choice of its policy set, Π, can be optimised. In our current setting, policies are treated equally and no analysis is made of the models that generate these policies, nor their policy cross-correlations. We note that there is such cross-correlation amongst policies in most games. A potential extension to OSA could consider moving beyond simply removing policies and allowing policy creation and evolution, similar to reversible jump Markov chain Monte Carlo methods (Roberts et al. 2001). This may allow for a dynamic policy set at each step of the game.

Other potential extensions to the algorithm could re-evaluate the approach for the best response policy and to consider policies that are not self-play. Further work can consider using the OSA approach for strategy selection alongside a method for generating diverse strategies (Canaan et al. 2019; Yang et al. 2020; Lupu et al. 2021; Parker-Holder et al. 2020b; Conti et al. 2018). Furthermore, it may even be possible to directly optimise agents to achieve strong ad-hoc coordination via our approach. For example, we could train agents in such a fashion that they provide useful contributions to the policy set. Other extensions of the work offers relaxation of the assumption that agents A^S play a fixed policy throughout the game. It is reasonable to assume that some agents in the system could play a mixed strategy during the game.

Furthermore, our experiments are restricted to two agents. Although it is not uncommon to differentiate between the two-player and n-player solutions within the literature, We see the n-player problem as a natural extension. We consider three distinct directions:

- OSA plays a set of identical simple agents

- OSA plays a mix of ad-hoc simple agents
- OSA plays a mix of OSA and other ad-hoc simple agents

For the first case good scaling is expected, as the OSA agent independently updates the steps for each of the simple agents in turn each time step. The second and third create more complexity. In the former the performance is dependent on the combination and interaction of other policies. However, evaluating other agents (steps 4 to 11 of the algorithm) remains the same, though the best response policy is conditional on the policies played by the other agents. The 3rd case can be seen as an extension of case 2, with multiple OSA agents and the fixed policy assumption relaxed.

Finally, we note that in our experiments in Part II the rules of the game are fixed. In real-world problems, rules themselves are a variable within the system and subject to change. Since OSA uses the conditional probabilities to estimate the joint probability, in theory it is possible to adapt to evolving rules, though this is a particularly challenging problem.

Appendices

A

Conducting the Hanabi Experiments

Contents

A.1 Training of Policies for the Hanabi Experiments

For the Hanabi experiments we create a new environment, OSA Hanabi Learning Environment (OSA-HLE) which is based on the Hanabi Learning Environment (HLE, Bard et al. 2020) with small alterations to allow for MAPPO (C. Yu et al. 2021) and Hanabi Open Agent Dataset (HOAD, Sarmasi et al. 2021) policies to interact with each other.

Each policy in Π set is trained separately. Policies MAPPO-1 and MAPPO-2 are trained using the MAPPO model with different seeds. With the exception of the seed number we make no changes to any of the parameters used in MAPPO repository to train Hanabi MAPPO agents.

For the polices from the HOAD set we use the same method detailed in Hanabi Open Agent Dataset repository to create imitator agents in OSA-HLE environment.

A.2 Per-step Evaluations and Hyperparameters

For the Hanabi game, F^{S,s_t}, which is the set of all possible hands that the simple agent A^S can be privy to at each step is large and therefore we sample from F^{S,s_t}. The sample number is fixed and set to 100 for all experiments in Hanabi. We note that sampling higher number of hands from F^{S,s_t} could improve accuracy but it will slow down the speed of the computations. Further in order to make comparisons between experiments that use OSA and cross-play experiments that do not use OSA, we run all experiments on a set of randomly chosen fixed seeds. We use 10 fixed seeds ranging from 40 to 50040 with step size of 5000. We evaluate the performance of 100 games for each seed for a total of 1000 games for each experiment. We note that for each seed, we reset the environment for each individual game and therefore the order of the deck is different for each of the 100 games evaluated for each seed. Finally none of the hyperparameters mentioned are optimized or tuned.

References

Achiam, J. (2020). *Spinning up documentation, release*. OpenAI.

Alt, B., A. Šošić, and H. Koeppl (2019). "Correlation priors for reinforcement learning". In: *Advances in Neural Information Processing Systems* 32, pages 14155–14165.

Amato, C., G. Chowdhary, A. Geramifard, N. K. Üre, and M. J. Kochenderfer (2013). "Decentralized control of partially observable Markov decision processes". In: *52nd IEEE Conference on Decision and Control*. IEEE, pages 2398–2405.

Arora, R., O. Dekel, and A. Tewari (2012). "Online bandit learning against an adaptive adversary: from regret to policy regret". In: *arXiv preprint arXiv:1206.6400*.

Baeck, T., D. Fogel, and Z. Michalewicz (2000). *Evolutionary Computation 1: Basic Algorithms and Operators*. CRC Press.

Bai, Y. and C. Jin (2020). "Provable self-play algorithms for competitive reinforcement learning". In: *International Conference on Machine Learning*. PMLR, pages 551–560.

Baker, C. L., J. Jara-Ettinger, R. Saxe, and J. B. Tenenbaum (2017). "Rational quantitative attribution of beliefs, desires and percepts in human mentalizing". In: *Nature Human Behaviour* 1.4, pages 1–10.

Bard, N., J. N. Foerster, S. Chandar, N. Burch, M. Lanctot, H. Song, E. Parisotto, V. Dumoulin, S. Moitra, E. Hughes, I. Dunning, S. Mourad, H. Larochelle, M. Bellemare, and M. Bowling (2020). "The Hanabi challenge: A new frontier for AI research". In: *Artificial Intelligence* 280, page 103216.

Bard, N., M. Johanson, N. Burch, and M. Bowling (2013). "Online implicit agent modelling". In: *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 255–262.

Barnett, G., R. Kohn, and S. Sheather (1996). "Bayesian estimation of an autoregressive model using Markov chain Monte Carlo". In: *Journal of Econometrics* 74.2, pages 237–254.

Barrett, S., A. Rosenfeld, S. Kraus, and P. Stone (2017). "Making friends on the fly: Cooperating with new teammates". In: *Artificial Intelligence* 242, pages 132–171.

Barth, C. B., P. Assem, T. Foulkes, W. H. Chung, T. Modeer, Y. Lei, and R. C. Pilawa-Podgurski (2019). "Design and control of a GAN-based, 13-level, flying capacitor multilevel inverter". In: *IEEE Journal of Emerging and Selected Topics in Power Electronics*.

Barto, A. G., R. S. Sutton, and C. W. Anderson (1983). "Neuronlike adaptive elements that can solve difficult learning control problems". In: *IEEE transactions on systems, man, and cybernetics* 5, pages 834–846.

Bell, A. J. and T. J. Sejnowski (1995). "An information-maximization approach to blind separation and blind deconvolution". In: *Neural computation* 7.6, pages 1129–1159.

Ben-Yosef, M. and D. Weinshall (2018). "Gaussian mixture generative adversarial networks for diverse datasets, and the unsupervised clustering of images". In: *arXiv preprint arXiv:1808.10356*.

Bengio, Y., P. Lamblin, D. Popovici, and H. Larochelle (2007). "Greedy layer-wise training of deep networks". In: *Advances in neural information processing systems*, pages 153–160.

Bernstein, D. S., R. Givan, N. Immerman, and S. Zilberstein (2002). "The complexity of decentralized control of Markov decision processes". In: *Mathematics of operations research* 27.4, pages 819–840.

Beynier, A., F. Charpillet, D. Szer, and A.-I. Mouaddib (2013). "DEC-MDP/POMDP". In: *Markov Decision Processes in Artificial Intelligence*, pages 277–318.

Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford university press.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Blum, A. and Y. Mansour (2007). "From external to internal regret." In: *Journal of Machine Learning Research* 8.6.

Bo, L., C. Sminchisescu, A. Kanaujia, and D. Metaxas (2008). "Fast algorithms for large scale conditional 3D prediction". In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, pages 1–8.

Bowling, M. and P. McCracken (2005). "Coordination and adaptation in impromptu teams". In: *AAAI*. Volume 5, pages 53–58.

Box, G. E. and G. C. Tiao (2011). *Bayesian inference in statistical analysis*. Volume 40. John Wiley & Sons.

Brochu, E., V. M. Cora, and N. de Freitas (2010). "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning". In: *CoRR* abs/1012.2599.

Brown, N. and T. Sandholm (2018). "Superhuman AI for heads-up no-limit poker: Libratus beats top professionals". In: *Science* 359.6374, pages 418–424.

Bureau of Labor Statistics, United States Department of Labor (2018). URL: `https://www.bls.gov/`.

Canaan, R., X. Gao, Y. Chung, J. Togelius, A. Nealen, and S. Menzel (2020). "Evaluating RL agents in Hanabi with unseen partners". In: *AAAI'20 Reinforcement Learning in Games Workshop*.

Canaan, R., J. Togelius, A. Nealen, and S. Menzel (2019). "Diverse Agents for Ad-Hoc Cooperation in Hanabi". In: *2019 IEEE Conference on Games (CoG)*, pages 1–8. DOI: 10.1109/CIG.2019.8847944.

Candanedo, L. M., V. Feldheim, and D. Deramaix (2017). "Data driven prediction models of energy use of appliances in a low-energy house". In: *Energy and buildings* 140, pages 81–97.

Carroll, M., R. Shah, M. K. Ho, T. Griffiths, S. A. Seshia, P. Abbeel, and A. D. Dragan (2019). "On the Utility of Learning about Humans for Human-AI Coordination". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Edited by H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, pages 5175–5186. URL: https://proceedings.neurips.cc/paper/2019/hash/f5b1b89d98b7286673128a5fb112cb9a-Abstract.html.

Cassandra, A. R. (1998). "A survey of POMDP applications". In: *Working notes of AAAI 1998 fall symposium on planning with partially observable Markov decision processes*. Volume 1724.

Chollet, F. et al. (2015). *Keras*. URL: https://keras.io.

Choudrey, R. A. and S. J. Roberts (2003). "Variational mixture of Bayesian independent component analyzers". In: *Neural computation* 15.1, pages 213–252.

Clette, F. (2015). *WDC-SILSO*. URL: http://www.sidc.be/silso/.

Conitzer, V. and T. Sandholm (2007). "AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents". In: *Machine Learning* 67.1-2, pages 23–43.

Conti, E., V. Madhavan, F. P. Such, J. Lehman, K. O. Stanley, and J. Clune (2018). "Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-seeking Agents". In: *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*. Montréal, Canada: Curran Associates Inc., pages 5032–5043. URL: http://dl.acm.org/citation.cfm?id=3327345.3327410.

Cortez, P., M. Rio, M. Rocha, and P. Sousa (2012). "Multi-scale Internet traffic forecasting using neural networks and time series methods". In: *Expert Systems* 29.2, pages 143–155.

Crandall, J. W. and M. A. Goodrich (2011). "Learning to compete, coordinate, and cooperate in repeated games using reinforcement learning". In: *Machine Learning* 82.3, pages 281–314.

Dafoe, A., E. Hughes, Y. Bachrach, T. Collins, K. R. McKee, J. Z. Leibo, K. Larson, and T. Graepel (2020). "Open problems in cooperative AI". In: *arXiv preprint arXiv:2012.08630*.

De Sa, C., V. Chen, and W. Wong (2018). "Minibatch Gibbs sampling on large graphical models". In: *International Conference on Machine Learning*, pages 1173–1181.

De Vito, S., E. Massera, M. Piga, L. Martinotto, and G. Di Francia (2008). "On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario". In: *Sensors and Actuators B: Chemical* 129.2, pages 750–757.

Denker, J. and Y. LeCun (1990). "Transforming neural-net output levels to probability distributions". In: *Advances in neural information processing systems* 3.

DiGiovanni, A. and E. C. Zell (2021). "Survey of Self-Play in Reinforcement Learning". In: *arXiv preprint arXiv:2107.02850*.

Duchi, J., E. Hazan, and Y. Singer (2011). "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research* 12.7.

Eghbal-zadeh, H., W. Zellinger, and G. Widmer (2019). "Mixture density generative adversarial networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5820–5829.

Esteban, C., S. L. Hyland, and G. Rätsch (2017). "Real-valued (Medical) Time Series Generation with Recurrent Conditional GANs". In: *arXiv preprint arXiv:1706.02633*.

Everson, R. and S. J. Roberts (1999). "Independent component analysis: A flexible nonlinearity and decorrelating manifold approach". In: *Neural computation* 11.8, pages 1957–1983.

Foerster, J. N. (2018). "Deep Multi-Agent Reinforcement Learning". In: *University of Oxford*.

Foerster, J. N., Y. M. Assael, N. De Freitas, and S. Whiteson (2016). "Learning to Communicate with Deep Multi-Agent Reinforcement Learning". In: *Advances in Neural Information Processing Systems 29*. Edited by D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, pages 2137–2145.

Foerster, J. N., F. Song, E. Hughes, N. Burch, I. Dunning, S. Whiteson, M. Botvinick, and M. Bowling (2019). "Bayesian action decoder for deep multi-agent reinforcement learning". In: *International Conference on Machine Learning*. PMLR, pages 1942–1951.

Foster, L., A. Waagen, N. Aijaz, M. Hurley, A. Luis, J. Rinsky, C. Satyavolu, M. J. Way, P. Gazis, and A. Srivastava (2009). "Stable and Efficient Gaussian Process Calculations." In: *Journal of Machine Learning Research* 10.4.

Freedman, D. A. (2009). *Statistical models: theory and practice*. cambridge university press.

Fuller, W. A. (2009). *Introduction to statistical time series*. Volume 428. John Wiley & Sons.

Gal, Y. and Z. Ghahramani (2016). "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning". In: *international conference on machine learning*. PMLR, pages 1050–1059.

Gardner, M. W. and S. Dorling (1998). "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences". In: *Atmospheric environment* 32.14-15, pages 2627–2636.

Ge, H., K. Xu, and Z. Ghahramani (2018). "Turing: a language for flexible probabilistic inference". In: *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*, pages 1682–1690. URL: http://proceedings.mlr.press/v84/ge18b.html.

Gelfand, A. E. and A. F. Smith (1990). "Sampling-based approaches to calculating marginal densities". In: *Journal of the American statistical association* 85.410, pages 398–409.

Geman, S. and D. Geman (1984). "Stochastic Relaxation, Gibbs Distributions and Bayesian Restoration of Images". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-6.6, pages 721–741. DOI: 10.1109/TPAMI.1984.4767596.

Geyer, C. J. (1992). "Practical Markov chain Monte Carlo". In: *Statistical science*, pages 473–483.

Ghosh, S., J. Yao, and F. Doshi-Velez (2018). "Structured variational learning of Bayesian neural networks with horseshoe priors". In: *International Conference on Machine Learning*. PMLR, pages 1744–1753.

Gibbs, M. and D. J. MacKay (1997). *Efficient implementation of Gaussian processes*.

Gibson, R. (2013). "Regret minimization in non-zero-sum games with applications to building champion multiplayer computer poker agents". In: *arXiv preprint arXiv:1305.0034*.

Goldberg, D. E. and K. Deb (1991). "A comparative analysis of selection schemes used in genetic algorithms". In: *Foundations of genetic algorithms*. Volume 1. Elsevier, pages 69–93.

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep learning*. MIT press.

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014). "Generative adversarial nets". In: *Advances in neural information processing systems*, pages 2672–2680.

Graves, A. (2011). "Practical variational inference for neural networks". In: *Advances in neural information processing systems* 24.

Gurumurthy, S., R. K. Sarvadevabhatla, and R. V. Babu (2017). "Deligan: Generative adversarial networks for diverse and limited data". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 166–174.

Ha, D. (2020). *Slime Volleyball Gym Environment*. URL: `https://github.com/hardmaru/slimevolleygym`.

Hastings, W. K. (1970). *Monte Carlo sampling methods using Markov chains and their applications*.

Hessel, M., J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. G. Azar, and D. Silver (2018). "Rainbow: Combining Improvements in Deep Reinforcement Learning". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*. Edited by S. A. McIlraith and K. Q. Weinberger. AAAI Press, pages 3215–3222.

Hinton, G. E. (1984). *Distributed representations*.

Hinton, G. E. and R. R. Salakhutdinov (2006). "Reducing the dimensionality of data with neural networks". In: *science* 313.5786, pages 504–507.

Hinton, G. E., N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov (2012). "Improving neural networks by preventing co-adaptation of feature detectors". In: *arXiv preprint arXiv:1207.0580*.

Hinton, G. E. and D. Van Camp (1993). "Keeping the neural networks simple by minimizing the description length of the weights". In: *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13.

Hipel, K. W. and A. McLeod (1994). *Number of daily births in Quebec, Jan. 01, 1977 to Dec. 31, 1990*. `https://datamarket.com/data/set/235j`.

Hochreiter, S. and J. Schmidhuber (Nov. 1997). "Long Short-Term Memory". In: *Neural Computation* 9.8, pages 1735–1780. ISSN: 0899-7667. DOI: `10.1162/neco.1997.9.8.1735`. URL: `http://dx.doi.org/10.1162/neco.1997.9.8.1735`.

Hodge, J. A., K. V. Mishra, and A. I. Zaghloul (2019). "Joint multi-layer GAN-based design of tensorial RF metasurfaces". In: *2019 IEEE 29th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, pages 1–6.

Hu, H. and J. N. Foerster (2019). "Simplified action decoder for deep multi-agent reinforcement learning". In: *arXiv preprint arXiv:1912.02288*.

Hu, H., A. Lerer, B. Cui, L. Pineda, D. Wu, N. Brown, and J. N. Foerster (2021). "Off-Belief Learning". In: *The International Conference on Machine Learning*.

Hu, H., A. Peysakhovich, A. Lerer, and J. N. Foerster (2020). ""Other-Play"for Zero-Shot Coordination". In: *Proceedings of Machine Learning and Systems 2020*, pages 9396–9407.

Investing.com (2018). *Euro US Dollar Daily Price*. URL: `https://www.investing.com/currencies/eur-usd-historical-data`.

Jacobs, R. A., M. I. Jordan, S. J. Nowlan, and G. E. Hinton (1991). "Adaptive mixtures of local experts". In: *Neural computation* 3.1, pages 79–87.

Jordan, M. I. and R. A. Jacobs (1994). "Hierarchical mixtures of experts and the EM algorithm". In: *Neural computation* 6.2, pages 181–214.

Kendall, A. and Y. Gal (2017). "What uncertainties do we need in bayesian deep learning for computer vision?" In: *arXiv preprint arXiv:1703.04977*.

Kingma, D. P. and J. Ba (2015). "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Edited by Y. Bengio and Y. LeCun. URL: `http://arxiv.org/abs/1412.6980`.

Kingma, D., S. Mohamed, D. Rezende, and M. Welling (2014). "Semi-supervised learning with deep generative models". In: *Advances in neural information processing systems*, pages 3581–3589.

Ko, G. G., Y. Chai, R. A. Rutenbar, D. Brooks, and G.-Y. Wei (2019). "Accelerating Bayesian inference on structured graphs using parallel Gibbs sampling". In: *2019*

29th International Conference on Field Programmable Logic and Applications (FPL). IEEE, pages 159–165.

Konda, V. R. and J. N. Tsitsiklis (2000). "Actor-critic algorithms". In: *Advances in neural information processing systems*, pages 1008–1014.

Kullback, S. and R. A. Leibler (1951). "On information and sufficiency". In: *The annals of mathematical statistics* 22.1, pages 79–86.

Kwon, Y., J.-H. Won, B. J. Kim, and M. C. Paik (2020). "Uncertainty quantification using Bayesian neural networks in classification: Application to biomedical image segmentation". In: *Computational Statistics & Data Analysis* 142, page 106816.

LeCun, Y. (1989). "Generalization and network design strategies". In: *Connectionism in perspective* 19, pages 143–155.

Lerer, A., H. Hu, J. Foerster, and N. Brown (2020). "Improving Policies via Search in Cooperative Partially Observable Games". In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI*, pages 7187–7194.

Lerer, A. and A. Peysakhovich (2019). "Learning Existing Social Conventions via Observationally Augmented Self-Play". In: *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pages 107–114.

Lindsay, B. G. (1995). "Mixture models: theory, geometry and applications". In: *NSF-CBMS regional conference series in probability and statistics*. JSTOR, pages i–163.

Lowe, R., Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch (2017). "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Edited by I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, pages 6379–6390.

Ludwig, M., L.-F. Nothias, K. Dührkop, I. Koester, M. Fleischauer, M. A. Hoffmann, D. Petras, F. Vargas, M. Morsy, L. Aluwihare, et al. (2020). "Database-

independent molecular formula annotation using Gibbs sampling through ZODIAC". In: *Nature Machine Intelligence* 2.10, pages 629–641.

Luo, Y., X. Cai, Y. Zhang, J. Xu, and Y. Xiaojie (2018). "Multivariate time series imputation with generative adversarial networks". In: *Advances in Neural Information Processing Systems*, pages 1596–1607.

Lupu, A., H. Hu, and J. N. Foerster (2021). "Trajectory Diversity for Zero-Shot Coordination". In: *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1593–1595.

MacKay, D. J. (1992). "A practical Bayesian framework for backpropagation networks". In: *Neural computation* 4.3, pages 448–472.

MacKay, D. J. (1996a). "Hyperparameters: optimize, or integrate out?" In: *Maximum entropy and bayesian methods*. Springer, pages 43–59.

MacKay, D. J. (1996b). *Maximum likelihood and covariant algorithms for independent component analysis*.

Mackey, M. C. and L. Glass (1977). "Oscillation and chaos in physiological control systems". In: *Science* 197.4300, pages 287–289.

Manski, C. F. (1991). "Regression". In: *Journal of Economic Literature* 29.1, pages 34–50.

Marquis, P., O. Papini, and H. Prade (2020). *A Guided Tour of Artificial Intelligence Research: Volume I: Knowledge Representation, Reasoning and Learning*. Springer Nature.

Meeds, E. and S. Osindero (2006). "An alternative infinite mixture of Gaussian process experts". In: *Advances in neural information processing systems* 18, page 883.

Melis, A. P. and D. Semmann (2010). "How is human cooperation different?" In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 365.1553, pages 2663–2674.

Metropolis, N., A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller (1953). "Equation of state calculations by fast computing machines". In: *The journal of chemical physics* 21.6, pages 1087–1092.

Minnotte, M. C. (1997). "Nonparametric testing of the existence of modes". In: *The Annals of Statistics*, pages 1646–1660.

Mirjalili, S. (2019). "Genetic algorithm". In: *Evolutionary algorithms and neural networks*. Springer, pages 43–55.

Mirza, M. and S. Osindero (2014). "Conditional generative adversarial nets". In: *arXiv preprint arXiv:1411.1784*.

Mnih, V., A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu (2016). "Asynchronous methods for deep reinforcement learning". In: *International conference on machine learning*. PMLR, pages 1928–1937.

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. (2015). "Human-level control through deep reinforcement learning". In: *nature* 518.7540, pages 529–533.

Moravcik, M., M. Schmid, N. Burch, V. Lisy, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling (2017). "Deepstack: Expert-level artificial intelligence in heads-up no-limit poker". In: *Science* 356.6337, pages 508–513.

Mordatch, I. and P. Abbeel (2018). "Emergence of grounded compositional language in multi-agent populations". In: *Thirty-second AAAI conference on artificial intelligence*.

Mouret, J.-B. and J. Clune (2015). "Illuminating search spaces by mapping elites". In: *ArXiv* abs/1504.04909.

Nair, R., M. Tambe, M. Yokoo, D. Pynadath, and S. Marsella (2002). "Towards computing optimal policies for decentralized POMDPs". In: *Notes of the 2002 AAAI Workshop on Game Theoretic and Decision Theoretic Agents*.

Nair, R., M. Tambe, M. Yokoo, D. Pynadath, and S. Marsella (2003). "Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings". In: *IJCAI*. Volume 3, pages 705–711.

Nash, J. F. (2016). "4. The Bargaining Problem". In: *The Essential John Nash*. Princeton University Press, pages 37–48.

National Centres for Environmental Information (2021). *Minimum daily temperatures in Caribou, Maine, USA, 1940-1950*. `https://www.ncdc.noaa.gov/cdo-web/datasets`.

Nayyar, A., A. Mahajan, and D. Teneketzis (2013). "Decentralized stochastic control with partial history sharing: A common information approach". In: *IEEE Transactions on Automatic Control* 58.7, pages 1644–1658.

Neal, R. M. (1995). *Bayesian learning for neural networks*. PhD thesis, University of Toronto.

Nekoei, H., A. Badrinaaraayanan, A. Courville, and S. Chandar (2021). *Continuous Coordination As a Realistic Scenario for Lifelong Learning*. arXiv: `2103.03216 [cs.LG]`.

Ober, S. W. and L. Aitchison (2021). "Global inducing point variational posteriors for bayesian neural networks and deep gaussian processes". In: *International Conference on Machine Learning*. PMLR, pages 8248–8259.

Oden, T., R. Moser, and O. Ghattas (2010). "Computer predictions with quantified uncertainty, part I". In: *SIAM News* 43.9, pages 1–3.

Oliehoek, F. A. and C. Amato (2016). *A concise introduction to decentralized POMDPs*. Springer.

Oliehoek, F. A., M. T. Spaan, and N. Vlassis (2008). "Optimal and approximate Q-value functions for decentralized POMDPs". In: *Journal of Artificial Intelligence Research* 32, pages 289–353.

Orozco, B. P., G. Abbati, and S. Roberts (2018). "Mordred: Memory-based ordinal regression deep neural networks for time series forecasting". In: *arXiv preprint arXiv:1803.09704*.

Paige, B. and F. Wood (2014). "A compilation target for probabilistic programming languages". In: *International Conference on Machine Learning*. PMLR, pages 1935–1943.

Panait, L. and S. Luke (2005). "Cooperative multi-agent learning: The state of the art". In: *Autonomous agents and multi-agent systems* 11.3, pages 387–434.

Papamarkou, T., J. Hinkle, M. T. Young, and D. Womble (2019). "Challenges in Markov chain Monte Carlo for Bayesian neural networks". In: *arXiv preprint arXiv:1910.06539*.

Papoulis, A. (1984). *Probability, Random Variables, and Stochastic Processes*. McGraw–Hill.

Park, J. and I. W. Sandberg (1991). "Universal approximation using radial-basis-function networks". In: *Neural computation* 3.2, pages 246–257.

Parker-Holder, J., L. Metz, C. Resnick, H. Hu, A. Lerer, A. Letcher, A. Peysakhovich, A. Pacchiano, and J. N. Foerster (2020a). "Ridge Rider: Finding Diverse Solutions by Following Eigenvectors of the Hessian". In: *Advances in Neural Information Processing Systems*. Volume 33. Curran Associates, Inc., pages 753–765.

Parker-Holder, J., A. Pacchiano, K. M. Choromanski, and S. J. Roberts (2020b). "Effective Diversity in Population Based Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. Volume 33. Curran Associates, Inc., pages 18050–18062.

Parzen, E. (1962). "On estimation of a probability density function and mode". In: *The annals of mathematical statistics* 33.3, pages 1065–1076.

Patil, A., D. Huard, and C. J. Fonnesbeck (2010). "PyMC: Bayesian stochastic modelling in Python". In: *Journal of statistical software* 35.4, page 1.

Peysakhovich, A. and A. Lerer (2017). "Prosocial learning agents solve generalized stag hunts better than selfish ones". In: *arXiv preprint arXiv:1709.02865*.

Plummer, M. et al. (2003). "JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling". In: *Proceedings of the 3rd international workshop on distributed statistical computing*. Volume 124. 125.10. Vienna, Austria., pages 1–10.

Powell, M. (2001). "Radial basis function methods for interpolation to functions of many variables". In: *HERCMA*. Citeseer, pages 2–24.

Premack, D. and G. Woodruff (1978). "Does the chimpanzee have a theory of mind?" In: *Behavioral and brain sciences* 1.4, pages 515–526.

Quinonero-Candela, J. and C. E. Rasmussen (2005). "A unifying view of sparse approximate Gaussian process regression". In: *The Journal of Machine Learning Research* 6, pages 1939–1959.

Rasmussen, C. E. and Z. Ghahramani (2002). "Infinite mixtures of Gaussian process experts". In: *Advances in neural information processing systems* 2, pages 881–888.

Rasmussen, C. E. and C. Williams (2005). *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning series. MIT Press. ISBN: 9780262182539. URL: `https://books.google.com/books?id=Tr34DwAAQBAJ`.

Richardson, E. and Y. Weiss (2018). "On GANs and GMMs". In: *Advances in Neural Information Processing Systems*, pages 5847–5858.

Roberts, S. J., C. Holmes, and D. Denison (2001). "Minimum-entropy data partitioning using reversible jump Markov chain Monte Carlo". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23.8, pages 909–914.

Roberts, S. J., M. Osborne, M. Ebden, S. Reece, N. Gibson, and S. Aigrain (2013). "Gaussian processes for time-series modelling". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 371.1984, page 20110550.

Roberts, S. J. and W. D. Penny (2002). "Variational Bayes for generalized autoregressive models". In: *IEEE Transactions on Signal Processing* 50.9, pages 2245–2257.

Rosenblatt, F. (1958). "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6, page 386.

Rosenblatt, F. (1961). *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. Technical report. Cornell Aeronautical Lab Inc Buffalo NY.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1985). *Learning internal representations by error propagation*. Technical report. California Univ San Diego La Jolla Inst for Cognitive Science.

Sarmasi, A., T. Zhang, C.-H. Cheng, H. Pham, X. Zhou, D. Nguyen, S. Shekdar, and J. McCoy (2021). "HOAD: The Hanabi Open Agent Dataset". In: *AAMAS*.

Schulman, J., S. Levine, P. Abbeel, M. Jordan, and P. Moritz (2015). "Trust region policy optimization". In: *International conference on machine learning*. PMLR, pages 1889–1897.

Schulman, J., F. Wolski, P. Dhariwal, A. Radford, and O. Klimov (2017). "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347*.

Schwenker, F., H. A. Kestler, and G. Palm (2002). "Unsupervised and supervised learning in radial-basis-function networks". In: *Self-Organizing Neural Networks*. Springer, pages 217–243.

Shi, J. Q., R. Murray-Smith, and D. Titterington (2005). "Hierarchical Gaussian process mixtures for regression". In: *Statistics and computing* 15.1, pages 31–41.

Shumway, R. H. and D. S. Stoffer (2000). *Time series analysis and its applications*. Volume 3. Springer.

Silver, D., T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis (2017a). "Mastering chess and shogi by self-play with a general reinforcement learning algorithm". In: *arXiv preprint arXiv:1712.01815*.

Silver, D., T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis (2018). "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419, pages 1140–1144.

Silver, D., J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. (2017b). "Mastering the game of Go without human knowledge". In: *Nature* 550.7676, pages 354–359.

Somani, A., N. Ye, D. Hsu, and W. S. Lee (2013). "DESPOT: Online POMDP planning with regularization". In: *Advances in neural information processing systems* 26, pages 1772–1780.

Spiegelhalter, D., A. Thomas, N. Best, and D. Lunn (2007). "OpenBUGS user manual". In: *Version* 3.2, page 2007.

Stein, M. L. (1999). *Interpolation of spatial data: some theory for kriging*. Springer Science & Business Media.

Stigler, S. M. (2005). "PS Laplace, Théorie analytique des probabilités, (1812); Essai philosophique sur les probabilités, (1814)". In: *Landmark Writings in Western Mathematics 1640-1940*. Elsevier, pages 329–340.

Stone, P., G. A. Kaminka, S. Kraus, and J. S. Rosenschein (2010). "Ad Hoc Autonomous Agent Teams: Collaboration without Pre-Coordination". In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, pages 1504–1509.

Sukhbaatar, S., A. Szlam, and R. Fergus (2016). "Learning Multiagent Communication with Backpropagation". In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. Edited by D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, pages 2244–2252.

Sun, S. (2013). "Infinite mixtures of multivariate Gaussian processes". In: *Machine Learning and Cybernetics (ICMLC), 2013 International Conference on*. Volume 3. IEEE, pages 1011–1016.

Sutton, R. S. and A. G. Barto (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., D. A. McAllester, S. P. Singh, and Y. Mansour (2000). "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in neural information processing systems*, pages 1057–1063.

Tesauro, G. (1994). "TD-Gammon, a self-teaching backgammon program, achieves master-level play". In: *Neural computation* 6.2, pages 215–219.

Titterington, D., A. Smith, and H. Makov (1985). *Statistical analysis of finite mixture distributions*. Volume 198. John Wiley & Sons Incorporated.

Tresp, V. (2001). "Mixtures of Gaussian processes". In: *Advances in neural information processing systems*, pages 654–660.

Tsiligkaridis, T. and A. O. Hero (2013). "Covariance estimation in high dimensions via Kronecker product expansions". In: *IEEE Transactions on Signal Processing* 61.21, pages 5347–5360.

Tucker, H. G. (2013). *A graduate course in probability*. Courier Corporation.

U.S. Energy Information Administration (2020). URL: `https://www.eia.gov/`.

Weaver, L. and N. Tao (2013). "The optimal reward baseline for gradient-based reinforcement learning". In: *arXiv preprint arXiv:1301.2315*.

Widrow, B., N. K. Gupta, and S. Maitra (1973). "Punish/reward: Learning with a critic in adaptive threshold systems". In: *IEEE Transactions on Systems, Man, and Cybernetics* 5, pages 455–465.

Wiese, M., R. Knobloch, R. Korn, and P. Kretschmer (2020). "Quant GANs: Deep Generation of Financial Time Series". In: *Quantitative Finance*, pages 1–22.

Wu, H., B. Gu, X. Wang, V. Pickert, and B. Ji (2019). "Design and control of a bidirectional wireless charging system using GAN devices". In: *2019*

IEEE Applied Power Electronics Conference and Exposition (APEC). IEEE, pages 864–869.

Yahoo!Finance (2020a). *CBOE Volatility Index Historical Data*. URL: `https://finance.yahoo.com/quote/%5EVIX/history?p=%5EVIX`.

Yahoo!Finance (2020b). *Invesco DB US Dollar Index Bullish Fund Historical Data*. URL: `https://finance.yahoo.com/quote/UUP/history?p=UUP`.

Yahoo!Finance (2020c). *iShares MSCI Brazil Small-Cap ETF Historical Data*. URL: `https://finance.yahoo.com/quote/EWZS/history?p=EWZS`.

Yang, Y., Y. Wen, J. Wang, L. Chen, K. Shao, D. Mguni, and W. Zhang (2020). "Multi-Agent Determinantal Q-Learning". In: *Proceedings of the 37th International Conference on Machine Learning*. Volume 119, pages 10757–10766.

Yoon, J., D. Jarrett, and M. van der Schaar (2019). "Time-series Generative Adversarial Networks". In: *Advances in Neural Information Processing Systems*. Volume 32. Curran Associates, Inc.

Yu, C., A. Velu, E. Vinitsky, Y. Wang, A. M. Bayen, and Y. Wu (2021). "The Surprising Effectiveness of MAPPO in Cooperative, Multi-Agent Games". In: *CoRR* abs/2103.01955.

Yu, L., J. Song, and S. Ermon (2019). "Multi-agent adversarial inverse reinforcement learning". In: *International Conference on Machine Learning*. PMLR, pages 7194–7201.

Yu, Y. and W. J. Zhou (2018). "Mixture of GANs for Clustering." In: *IJCAI*, pages 3047–3053.

Zhang, Y., W. E. Leithead, and D. J. Leith (2005). "Time-series Gaussian process regression based on Toeplitz computation of O (N 2) operations and O (N)-level storage". In: *Proceedings of the 44th IEEE Conference on Decision and Control*. IEEE, pages 3711–3716.

Zhou, X., Z. Pan, G. Hu, S. Tang, and C. Zhao (2018). "Stock market prediction on high-frequency data using generative adversarial nets". In: *Mathematical Problems in Engineering* 2018.

Zinkevich, M., M. Johanson, M. Bowling, and C. Piccione (2007). "Regret minimization in games with incomplete information". In: *Advances in neural information processing systems* 20, pages 1729–1736.

Zou, F., L. Shen, Z. Jie, J. Sun, and W. Liu (2018). "Weighted AdaGrad with unified momentum". In: *arXiv preprint arXiv:1808.03408*.