FACULTY OF ENGINEERING
AND ARCHITECTURE

BRUNO: Exchangeable Deep Learning Models of Predictive Distributions

Iryna Korshunova

Doctoral dissertation submitted to obtain the academic degree of
Doctor of Computer Science Engineering

Supervisors
Prof. Joni Dambre, PhD* - Prof. Arthur Gretton, PhD**

* Department of Electronics and Information Systems
Faculty of Engineering and Architecture, Ghent University
** University College London, United Kingdom

November 2020

GHENT
UNIVERSITY

ISBN 978-94-6355-441-1
NUR 984
Wettelijk depot: D/2020/10.500/118

Members of the Examination Board

Chair

Prof. Em. Daniël De Zutter, PhD, Ghent University

Other members entitled to vote

Prof. Gert De Cooman, PhD, Ghent University
Prof. Ferenc Huszár, PhD, University of Cambridge, United Kingdom
Michiel Stock, PhD, Ghent University
Prof. Dougal Sutherland, PhD, Toyota Technological Institute at Chicago, USA

Supervisors

Prof. Joni Dambre, PhD, Ghent University
Prof. Arthur Gretton, PhD, University College London, United Kingdom

Acknowledgements

The five years of my PhD were so far the happiest in my life, and I am grateful to many people for making it so. Likewise, this thesis would not have existed without a lot of their help.

First of all, I would like to thank Joni for supporting me all the way from writing a scholarship application to the day I graduate. Especially, I am grateful for having a lot of freedom and, most importantly, your trust in whatever I was doing. Thank you for accepting my long absences, which made me a pretty useless PhD student to have around in the lab.

The main ideas presented in this thesis I owe to Ferenc. I was very lucky when you suggested me to look at the problem of exchangeable RNNs and then gave me some essential guidance. Without your advice, and I suspect my PhD would have ended much worse. I found a lot of pleasure in working on this topic and learnt many new things. I will never be able to repay for what you have done for me.

I am also grateful to Ferenc for introducing me to Arthur – the friendliest and smartest professor I have ever met. Arthur, I will not even try to describe how special and important you are as a mentor, and how much being able to stay at Gatsby has influenced me. I will always be thankful and forever admire you.

Another awesome person who deserves my infinite gratitude is Lucas. During my three internships at Twitter, it was amazing that I could always sit next to you and ask all sort of questions at all times. I have learnt so much! Sometimes I wish I could be your

full-time intern. Maybe it will still be the case in the future. Also, thank you for becoming one of my best friends!

Last but not least, or perhaps the non-leastest person who was essential for this PhD to end well, is Jonas. You are my biggest support and help. Thank you for all the research-related things, like discussing ideas, reviewing my papers, going together to conferences, proofreading this thesis, etc. But most importantly, thanks for your love and for being with me all the time.

Further, I want to thank Francis and Michiel for giving me a place to live during the first months of my PhD. Also, thank you, Francis, for making our lab such a friendly place, it would have been different without you.

A huge thanks go to Sander and Aaron. You gave me the best examples of the type of researcher I wanted to be. Thank you for everything I leant from you about deep learning, for introducing our lab to Kaggle competitions, and for letting me join you for ICML in 2015 which was my very first conference. While all members of the old Reslab are awesome people, I would like to especially thank Juan Pablo, Pieter and Pieter-Jan. PJ, thanks for making me write the epilepsy paper that convinced me to stay away from that field.

I am grateful to Lio, Jeroen, Matthias, Andreas, Frederic, Gabriel and Elias for the Kaggle competitions we did together. Also, thanks to other Reslab/IDlab members for interesting lunch stories, AoE evenings, lots of office plants (Olivier and Francis), a supply of Club-Mate (Dries) and maintaining our GPU farm (Lio, Len, Maxim). Moreover, I would like to thank Matthias for his practical knowledge of PhD defences and Olivier for sharing his LaTeX thesis template.

Visiting the Gatsby Unit was the highlight of my PhD. In this place of most brilliant researchers and nicest of people, I was utterly happy. Thus, huge thanks to all of you, and especially to Ricardo, Dougal, Heiko, Heishiro, Kevin, Wittawat, Sanjeevan, Barry, Michael and Mihaela.

I thank the Magic Pony team at Twitter for welcoming me for almost a year worth of internships. It was always fun and in-

tellectually stimulating to be among you. I doubt I will ever enjoy playing foosball as much as I did at Twitter. For that, I am especially grateful to Seb, Francisco, Casper, Jose, Aly, Ferenc, Lucas and Conrado, who played foosball badly but was keeping me a great company when everyone else went to Hawaii.

Throughout my PhD, I had a pleasure to collaborate with a number of amazing people, whom I have not mentioned yet. Yarin, Wenzhe, Hanchen, Mateusz, Thibault, Joao Felipe and Bob Sturm, thank you all!

A massive thank you to Daniël De Zutter, Gert De Cooman, Michiel Stock, Ferenc Huszár, Dougal Sutherland (and his cat Bug) for being on my thesis jury. I could not wish for more competent and friendlier examiners!

I am immensely thankful for the support of my friends in London and elsewhere, but above all to Lucas and Thu, Kai, Luba, Katia and Solomia, Priska, and friends in Ghent, among whom Pieter I., Pieter G. and Kevin will always have a special place in my heart.

For being a continuous source of warmth and love, my deepest gratitude goes to my family in Ukraine and Belgium, in particular to papa Tolia, grandpa, mama Anne and papa Dirk, Oma, uncle Sereja and aunt Tania, my godmother, (in-law) sisters and brothers.

I would like to dedicate this thesis to my grandpa (Дедушка), who got me interested in math when I was small.

Iryna Korshunova, November 26, 2020

Summary

The concept of a model as a predictive device is central to the field of machine learning. Common tasks such as classification, regression or generation, can be viewed as doing certain types of predictions using models that learnt relevant properties of the training data. To learn efficiently, the models must encode a correct set of assumptions about the nature of the data. One of the basic assumptions is whether the order of the inputs matters. For example, order-dependence would be a suitable assumption for text modelling since permuting the words might give sentences a different meaning and so we want our model to capture this change. On the other hand, the opposite assumption of order-invariance is appropriate when dealing with sets of items, where no inherent order is present.

In Bayesian statistics, the concept related to sets is that of exchangeable sequences of random variables. Exchangeability implies that the joint distribution of any finite subset of variables is permutation-invariant. This property holds not only when variables are independent and identically distributed, but also when they are positively correlated. The latter is what enables learning from previous observations and allows to reason about latent variables behind the data as shown by the celebrated de Finetti's theorem. In this way, exchangeability has become the cornerstone assumption of Bayesian modelling.

Meta-learning is one recent field of machine learning research where exchangeable models can find their profound use. For instance, in a few-shot concept learning, some tasks can be

formulated as learning to complete short exchangeable sequences. While the Bayesian approach is appropriate here, it has several practical issues. Firstly, doing exact inference is often infeasible, and secondly, we need a way of dealing with complex high-dimensional observations, e.g. images, since those are involved in the most interesting applications. To tackle these issues, it is therefore appealing to combine the principledness of Bayesian methods with an expressive power of deep neural networks, which is the topic of this dissertation.

This thesis

We propose a family of exchangeable architectures called BRUNO, which leverages deep learning tools to perform exact Bayesian inference on sets of high-dimensional inputs. BRUNO, when viewed as a recurrent neural network, is defined in terms of predictive distributions. In theory, these distributions can be highly complex as a result of having a powerful bijective neural mapping as a part of our model. However, these predictive distributions are tractable to evaluate and to sample from. Both operations have a constant memory and a linear time complexity in the number of data points we condition on. Unlike many other methods related to BRUNO, we use no variational approximations in our model.

The above-mentioned properties of BRUNO stem directly from building a probabilistic predictive model from basic principles. With the meta-learning application in mind, exchangeability naturally serves as a starting point. We discuss the implications of such design and the possibility of building more general exchangeable models.

We devise BRUNO models for both unconditional and conditional distribution over exchangeable sequences. Together, these models cover a broad range of applications that require generalisation from short observed sequences while also modelling sequence variability. In the unconditional case, an illustrative task is to generate new instances of a handwritten character given a few examples of this character written by different people. On the other hand, the conditional case considers a scenario where the

model additionally receives a vector of labels or tags associated with every input. For example, if images in the given sequence are rotated to some known degrees, then the task would be not only to generate new instances of this character but to do so conditionally on a desired angle of rotation. In our experiments, we show how BRUNO can achieve competitive results on such few-shot generation tasks, and with minor modifications to its training procedure, few-shot classification tasks as well.

While the construction of predictive distributions in BRUNO models is done without a reference to the posterior over some latent variables, it is possible to find the analytic form of this posterior via an alternative formulation. This allows us to extend the applications of our models even further. In particular, we use conditional BRUNO for the problem of task inference in meta reinforcement learning, where we justify the applicability of exchangeable models. Also, we work out the details on how to combine BRUNO with an off-policy reinforcement learning algorithm so to obtain sample-efficiency, short training times and the ability to quickly adapt a policy to new tasks.

Our design of BRUNO targets the modelling of high-dimensional observations, and so in practice, it needs to sacrifice some of its theoretical properties when used with low-dimensional inputs. For instance, this is the case in reinforcement learning, where we need to model a predictive distribution of scalar rewards. We show that despite the necessary changes, our models remain successfully functional.

In the final chapter of this thesis, we discuss additional interpretations of BRUNO by drawing analogies with deep learning models, memory models and stochastic processes. This allows to gain a deeper understanding of BRUNO and connect it with a larger body of literature. For future prospects, we outline several concrete ideas that could lead to useful extensions of our models. We also present our view on the current issues in meta-learning research and how we hope the field will evolve.

Samenvatting

Het concept van een model als voorspeller staat centraal in het domein van machinaal leren. Veelvoorkomende taken zoals classificatie, regressie of generatie kunnen worden gezien als het maken van bepaalde voorspellingen aan de hand van modellen die relevante eigenschappen van de trainingsgegevens hebben geleerd. Om efficiënt te leren, moeten de modellen een verzameling juiste aannames over de aard van de gegevens maken. Eén van die basisaannames is of de volgorde van invoer belangrijk is. Ordeafhankelijkheid zou bijvoorbeeld een geschikte aanname zijn voor tekstmodellering, omdat het permuteren van woorden zinnen een andere betekenis kan geven en daarom willen we dat ons model deze verandering merkt. Aan de andere kant is de tegenovergestelde aanname van ordeinvariantie gepast bij het omgaan met verzamelingen, waar geen inherente volgorde aanwezig is.

Het concept van uitwisselbare reeksen willekeurige variabelen in de Bayesiaanse statistiek, is de tegenhanger van de ordeinvariantie bij verzamelingen. Uitwisselbaarheid houdt in dat een gezamenlijke verdeling van een eindige deelverzameling van variabelen permutatie-invariant is. Deze uitspraak geldt niet alleen wanneer variabelen onafhankelijk en identiek verdeeld zijn, maar ook wanneer ze positief gecorreleerd zijn. Dit laatste maakt het mogelijk om te leren van eerdere waarnemingen en om te redeneren over de latente variabelen achter de gegevens, zoals blijkt uit de beroemde stelling van de Finetti. Op deze manier is uitwisselbaarheid de hoeksteen van de Bayesiaanse modellering geworden.

Meta-leren is een recent gebied van onderzoek naar machinaal leren waar uitwisselbare modellen hun nut kunnen vinden. Bij weinig-pogingen-leren kunnen sommige taken bijvoorbeeld worden geformuleerd als het leren om korte uitwisselbare reeksen te voltooien. Hoewel de Bayesiaanse benadering hier geschikt is, heeft deze verschillende praktische problemen. Ten eerste is een exacte gevolgtrekking vaak onhaalbaar, en ten tweede hebben we een manier nodig om met complexe hoogdimensionale waarnemingen om te gaan, b.v. afbeeldingen, aangezien deze van toepassing zijn bij de meest interessante toepassingen. Om deze problemen aan te pakken, is het daarom aantrekkelijk om het principe van Bayesiaanse methoden te combineren met de expressieve kracht van diepe neurale netwerken, wat het onderwerp is van dit proefschrift.

Deze thesis

We stellen een familie van uitwisselbare architecturen voor, genaamd BRUNO, die diepgaande leermiddelen gebruiken om exacte Bayesiaanse inferentie uit te voeren op een verzameling van hoogdimensionale invoer. Wanneer BRUNO wordt gezien als een terugkerend neuraal netwerk, wordt het gedefinieerd in termen van voorspellende kansverdelingen. Hun analytische vorm kan in theorie arbitrair complex zijn als gevolg van een krachtige bijectieve neurale afbeelding als onderdeel van ons model. Deze voorspellende kansverdelingen zijn echter handelbaar om te evalueren en om van te bemonsteren. Beide bewerkingen hebben slechts een constant geheugen nodig en een lineaire tijdcomplexiteit in het aantal gegevenspunten dat we als gegeven nemen. In tegenstelling tot veel andere methoden die verband houden met BRUNO, gebruiken we geen variationele benaderingen in ons model.

De bovengenoemde eigenschappen van BRUNO komen rechtstreeks voort uit het bouwen van een probabilistisch voorspellend model op basis van basisprincipes. Met het meta-leren als toepassing in gedachten dient uitwisselbaarheid als natuurlijk uitgangspunt. We bespreken de implicaties van een dergelijk

ontwerp en de mogelijkheid om meer algemene uitwisselbare modellen te bouwen.

We ontwerpen BRUNO-modellen voor zowel onvoorwaardelijke als voorwaardelijke kansverdelingen over uitwisselbare rijen. Samen omvatten deze modellen een breed scala aan toepassingen die de veralgemening van korte waargenomen rijen vereisen, terwijl ze ook de variabiliteit van rijen modelleren. In het onvoorwaardelijke geval is het een illustratieve taak om nieuwe instanties van een karakter te genereren aan de hand van enkele voorbeelden van dit karakter, geschreven door verschillende mensen. Aan de andere kant beschouwt het voorwaardelijke geval een scenario waarin het model bovendien een vector ontvangt met labels die bij iedere invoer horen. Als afbeeldingen in de gegeven reeks bijvoorbeeld worden geroteerd naar een bepaalde hoek, dan zou het niet alleen de taak zijn om nieuwe instanties van dit karakter te genereren, maar om dit voorwaardelijk te doen onder een gewenste rotatiehoek. In onze experimenten laten we zien hoe BRUNO competitieve resultaten kan behalen op zulke generatietaken met weinig pogingen, en mits kleine aanpassingen aan de trainingsprocedure, ook op classificatie taken met weinig pogingen.

Hoewel de constructie van voorspellende verdelingen in BRUNO-modellen wordt gedaan zonder verwijzing naar de posterior over enkele latente variabelen, is het mogelijk om de analytische vorm van deze posterior te vinden via een alternatieve formulering. Hierdoor kunnen we de toepassingen van onze modellen nog verder uitbreiden. In het bijzonder gebruiken we voorwaardelijke BRUNO voor het probleem van taakinferentie in meta-reinforcement leren, waar we de toepasbaarheid van uitwisselbare modellen rechtvaardigen. We werken ook de details uit over het combineren van BRUNO met een off-policy versterkend leeralgoritme om een betere sample-efficiëntie te verkrijgen, met bijbehorende korte trainingstijden en de mogelijkheid om een policy snel aan nieuwe taken aan te passen.

Ons ontwerp van BRUNO is gericht op het modelleren van hoogdimensionale waarnemingen, en daarom moet het in de praktijk enkele van zijn theoretische eigenschappen opofferen wanneer het wordt gebruikt met laagdimensionale inputs. Dit is bijvoorbeeld

het geval bij reinforcement leren, waar we een voorspellende kansverdeling over scalaire beloningen moeten modelleren. We laten zien dat ondanks de noodzakelijke veranderingen onze modellen succesvol blijven functioneren.

In het laatste hoofdstuk van dit proefschrift bespreken we aanvullende interpretaties van BRUNO door analogieën te trekken met diepe modellen, geheugenmodellen en stochastische processen. Dit maakt het mogelijk om een dieper inzicht in BRUNO te krijgen en het te verbinden met een grotere hoeveelheid literatuur. Voor toekomstige vooruitzichten schetsen we verschillende concrete ideeën die kunnen leiden tot nuttige uitbreidingen van onze modellen. Ten slotte presenteren we onze visie op de huidige problemen binnen het onderzoek naar meta-leren en hoe we hopen dat het gebied verder zal evolueren.

Contents

List of Abbreviations

AGI	Artificial general intelligence
AMLE	Augmented maximum likelihood estimation
BAMDP	Bayes-adaptive Markov decision process
BNN	Bayesian neural network
CNN	Convolutional neural networks
CS	Compound symmetry
DNN	Deep neural network
ELBO	Evidence lower bound
GP	Gaussian process
GQN	Generative query networks
IDF	Integer discrete flow
LSTM	Long short-term memory
MAML	Model-agnostic meta-learning
MDP	Markov decision process
ML-PIP	Meta-learning approximate probabilistic inference for prediction
MLE	Maximum likelihood estimation
MLP	Multi-layer perceptron
MSE	Mean squared error
NP	Neural process
POMDP	Partially observable Markov decision process
Real NVP	Real-valued non-volume preserving transformation
RL	Reinforcement learning
RNN	Recurrent neural network

SAC	Soft actor-critic
SGD	Stochastic gradient descent
TP	Student-t process
VAE	Variational autoencoder

1

Introduction

1.1 The Bayesian Paradigm

Bayesian statistics interprets the concept of probability as the quantification of subjective beliefs and uses probability distributions to describe all the relevant unknown quantities , such as statistical hypotheses , as well as the data. The requirement to specify prior beliefs is one of the identifying features of Bayesian statistics. The application of Bayes' theorem and basic rules of probability give us a way to update the beliefs upon the available evidence, i.e. learning from the data.

To illustrate these concepts and the basic method of parametric Bayesian inference, let us consider the following example. We wish to find the bias of a coin, which we can toss n times, thus collecting a set of observations, i.e. the realization of $D = \{x_i\}_{i=1}^{n}$, where every x_i can be either heads or tails, conventionally denoted as 1 and 0. More formally, we need to formulate a belief over the hypotheses regarding the underlying data-generating process given a set of observations D from interacting with the system.

We can describe the repeated coin flips by the Bernoulli process parameterized by θ. This process is defined as a sequence of independent random variables $x_1, x_2, x_3, \ldots$ with every $x_i \sim \text{Bernoulli}(\theta)$ for some value of θ. For example, in the event of $\theta = 0.3$, $p(x_i = 1) = 0.3$ and $p(x_i = 0) = 0.7$ for every i. Here, and throughout the thesis, we will follow the notation of Bernardo and

Smith [6], and use the same lowercase symbol for both random variables and their values, since the interpretation should always be clear from the context.

Given the observations, we wish to form a belief over a set of statistical hypotheses $M = \{h_\theta : \theta \in \Theta\}$, where Θ is a space of parameter values. In our case, the coin's bias corresponds to θ, thus $h_\theta = \theta$ and $\Theta = [0, 1]$. The probability we need is $p(\theta|D)$ – a *posterior* belief over θ that combines our prior beliefs with the information from the observed data. As a candidate for the *prior* $p(\theta)$, we can choose $p(\theta) = \mathcal{U}(0, 1)$ – a uniform distribution on a $[0, 1]$ interval. The Bayes rule then tells us how to obtain $p(\theta|D)$ by combining the prior and the *likelihood* $p(D|\theta)$:

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} \tag{1.1}$$

In case of independent Bernoulli random variables, the likelihood is $p(D|\theta) = \theta^H(1-\theta)^T$, where H and T is the number of heads and tails among n tosses. The denominator $p(D)$ is the *marginal likelihood* computed as $p(D) = \int_0^1 p(D|\theta)p(\theta)d\theta$, which is often intractable.

Once we have the posterior distribution, we can answer any inferential query by computing an expectation of an appropriate function with respect to the posterior. Most often, we are interested in doing predictions. For example, in case of the coin, predict the outcome of the next toss x_{n+1} given previous n tosses, i.e. finding the *posterior predictive* distribution $p(x_{n+1}|x_{1:n})$ that can be computed as:

$$p(x_{n+1}|x_{1:n}) = \int p(x_{n+1}|\theta)p(\theta|x_{1:n})d\theta, \tag{1.2}$$

where we use the notation $x_{1:n}$ to denote $x_1, \ldots x_n$ for $n \geq 1$.

The main problem with the integral above is that for many relevant problems it is intractable, and much of the previous research was dedicated to approximate methods, such as numerical integration and variational inference [7, 81].

What we presented in this section is a common starting point for explaining Bayesian learning in many machine learning course

notes. Here, we took for granted the existence of a mathematical model with some unknown variables, in our case θ, that we wish to infer. However, one might argue that observable quantities $x_1, x_2, x_3, \ldots$ are the only things that truly exist, and so our main concern should be a predictive probability model – a specification of the joint probability distribution for any subset of $x_1, x_2, x_3, \ldots$ [6]. In the next section, we explain an important assumption that connects the two perspectives.

1.2 Exchangeability and de Finetti's Theorem

Let us return to the coin example and take a closer look at the sequence of random variables $x_1, x_2, x_3, \ldots$, where every x_i represents a coin flip on the i-th step. If we forget about the data-generating process and do not hypothesize that every x_i is sampled from Bernoulli(θ) with some θ, can we still assume these random variables to be independent? If we do, then $p(x_{n+1}|x_{1:n}) = p(x_{n+1})$ meaning that the past experience tells us nothing about the future. For example, a-priori we believe that the coin is fair, but we observe 100 tosses from which 90 landed heads. The i. i. d. assumption would force us to adhere to our prior, and thus assign equal probabilities to heads and tails in the next toss. Common sense, on the other hand, tells us that heads are more likely. Thus, in the Bayesian framework, to enable learning from past observations, we cannot assume *marginal independence*. Instead, the right assumption here is that of *conditional independence*. Indeed, in the Bernoulli process, by assuming that $x_i \sim$ Bernoulli(θ) for some value of θ, we made x_i's conditionally independent and identically distributed.

The Representation theorem of de Finetti [33], which is central to motivating parametric models in Bayesian statistics, ties the notion of conditional independence to the concept of *exchangeability*.

Formally, a stochastic process, or an infinite sequence of random variables $x_1, x_2, x_3, \ldots$, is said to be exchangeable if for all finite

$n \geq 1$ and all permutations π of $\{1, \ldots, n\}$

$$p(x_1, \ldots, x_n) = p\left(x_{\pi(1)}, \ldots, x_{\pi(n)}\right), \tag{1.3}$$

i.e. the joint probability remains the same under any permutation of the sequence.

Alternatively, the joint distribution $p(x_1, \ldots, x_n)$ can be viewed as a multivariate distribution in $\mathbb{R}^n$ of a random vector with coordinates corresponding to the variables $x_1, \ldots, x_n$. By adopting this perspective, we can think of a stochastic process as a consistent assignment of finite-dimensional distributions. In this context, by consistency we mean that for every n, a distribution defined on $\mathbb{R}^n$ can be obtained from a distribution on $\mathbb{R}^{n+1}$ by marginalizing out a certain coordinate. If these distributions are invariant under the re-ordering of coordinates, then the process is exchangeable [82].

For i. i. d. random variables, the definition in Eq. 1.3 holds since $p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p(x_i)$. Thus the i. i. d. assumption is a special case of exchangeability. However, exchangeability for infinite sequences, also covers a case of positively correlated random variables [1]. In the coin flip example, we ruled out the possibility of marginal independence. At the same time, there is no inherent order in the sequence of flips, so their joint probability satisfies the definition of exchangeability in Eq. 1.3.

Another example of an exchangeable sequence is a simplest form of a Gaussian process, where for every finite n, $(x_1, \ldots, x_n)$ jointly have a multivariate normal distribution $\mathcal{N}_n(\mathbf{0}, \mathbf{\Sigma})$ with $\mathbf{\Sigma}$ having a compound symmetry (CS) structure:

$$\mathbf{\Sigma}_{ij} = \begin{cases} v, & \text{if } i = j \\ \rho, & \text{if } i \neq j \end{cases}, \tag{1.4}$$

where we further need a restriction $\rho \in [0, v)$ to make sure that $\mathbf{\Sigma}$ is a valid covariance matrix for any n. Alternatively, the non-negativity of ρ can be derived from the properties of infinitely exchangeable sequences [1, 89]. The sequence is exchangeable since permuting the dimensions of this Gaussian does not change the joint $p(x_1, \ldots, x_n)$. This example will be the basis for models

presented in this thesis and we will later analyse it in greater detail.

The Representation theorem of de Finetti [33] states that every exchangeable process is a mixture of i. i. d. processes:

$$p(x_1, \ldots, x_n) = \int \prod_{i=1}^{n} p(x_i|\theta)p(\theta)d\theta, \tag{1.5}$$

where θ is some parameter (finite or possibly infinite dimensional) conditioned on which, the random variables are i. i. d. [1]. Here, we omitted some technicalities present in a general formulation of this theorem. For instance, we assumed that the distribution over θ has a density, so we could write $p(\theta)d\theta$ instead of the more general $P(d\theta)$, where P is a probability measure on θ. Similarly, in our future explanations, we will have no pretence at a mathematical rigour, which is above the level necessary for understanding the main underlying ideas.

One interpretation of de Finetti's theorem is, that under the exchangeability assumption, there must exist a θ parameter such that given θ, x_i variables become independent and identically distributed. In other words, the existence of θ is only justified under exchangeability.

In our previous Gaussian example, one can show that $x_1, \ldots, x_n$ are i. i. d. with $x_i \sim \mathcal{N}(\theta, v - \rho)$ conditioned on θ, while $p(\theta) = \mathcal{N}(0, \rho)$. Noticeably, this gives us an alternative way of sampling from our multivariate Gaussian $\mathcal{N}_n(\mathbf{0}, \mathbf{\Sigma})$. Namely, instead of sampling the whole vector $(x_1, \ldots x_n)$ at once, we can first sample θ from $\mathcal{N}(0, \rho)$ and then sample the values x_1 to x_n independently from $\mathcal{N}(\theta, v - \rho)$.

The definition of exchangeability and Eq. 1.5 from de Finetti's theorem can be formulated in terms of predictive distributions $p(x_{n+1}|x_{1:n})$. Exchangeabilty is then defined as two conditions that both need to hold [35]:

$$\begin{aligned} p(x_{n+1}|x_1, \ldots, x_n) &= p\left(x_{n+1}|x_{\pi(1)}, \ldots, x_{\pi(n)}\right) \\ p(x_{n+1} = a, x_{n+2} = b|x_{1:n}) &= p(x_{n+1} = b, x_{n+2} = a|x_{1:n}), \end{aligned} \tag{1.6}$$

and Eq. 1.5 in de Finetti's theorem takes the form of the predictive posterior, which we have already seen in Eq. 1.2:

$$p(x_{n+1}|x_{1:n}) = \int p(x_{n+1}|\theta)p(\theta|x_{1:n})d\theta. \tag{1.7}$$

Pondering about the meaning of Eq. 1.5 or Eq. 1.7, and given that exchangeability is a sufficient and necessary condition in de Finetti's theorem, another possible interpretation follows: learning to fit an exchangeable model to sequences of data is implicitly the same as learning to reason about the latent variables behind the data [59].

1.3 Exchangeability via Recurrent Neural Networks

Let us first summarize the two equivalent ways of Bayesian learning as established by de Finetti's theorem.

One strategy is through explicit Bayesian modelling: one defines a prior $p(\theta)$, a likelihood $p(x_i|\theta)$ and calculates the posterior in Eq. 1.1 or predictive posterior in Eq. 1.2. Here, the key difficulty is the intractability of both expressions as they require integrating over the parameter θ, so we might need to use approximations. This could violate the exchangeability property and make the explicit approach difficult.

On the other hand, we do not have to explicitly represent the posterior. If the goal is to make predictions, one could define a predictive (autoregressive) distribution $p(x_n|x_{1:n-1})$ directly, and as long as the process is exchangeable, it is consistent with Bayesian reasoning. To our best knowledge, this thesis makes the first attempt in exploring this implicit way of doing inference. The key difficulty here is defining an easy-to-calculate $p(x_n|x_{1:n-1})$ from an exchangeable process. One possible idea on how to implement this approach is discussed next.

Recurrent neural networks (RNN) are powerful models of autoregressive processes which satisfy the requirement of an easy evaluation of the predictive distribution. RNNs found many applications in the domains such as text modelling, speech recognition, music

generation and robot control problems to name a few [47, 61, 104, 110]. In all these problems, the order in which the inputs appear is important, and RNNs are excellent in modelling it. However, RNNs' inherent order dependence poses a challenge if we want to use them for exchangeable data. To model an exchangeable process with RNNs, we need to make sure that conditions in Eq. 1.3 or Eq. 1.6 are satisfied. If we could do so, this "exchangeable RNN" would implicitly do Bayesian inference while giving us exact predictive probabilities almost for free. In this implicit approach, we might not be able to recover the usual components such as the prior, likelihood and the posterior over the latent variables θ. However, relying on de Finetti's theorem, we can rest assured that they exist and can be very complex given the expressiveness of RNNs. After all, in many practical applications, the predictive distribution is often the only relevant thing.

Implementing exchangeable models via RNNs is not a straightforward task and this thesis holds no answer to how one can train or modify the architecture of an ordinary RNN to model exchangeable data. Instead, we will follow a path that combines certain features of both implicit and explicit strategies into a model, which we called ***BRUNO**: Bayesian RecUrrent Neural mOdel*. Its main idea is to leverage deep learning tools to perform exact Bayesian inference on sets of high dimensional, complex observations with a relatively little computational cost.

1.4 General application

The modelling of unordered sets has been a recent focus in machine learning, both due to relevant application domains and to efficiency gains when dealing with groups of objects [116, 127, 137]. An exchangeable sequence can be naturally viewed as a Bayesian counterpart of a set. This view renders many set-related problems suitable for the Bayesian approach. For example, consider a task, where given a few pictures of swooses as in Figure 1.1, we are asked to draw something alike. We might not have seen a swoose before, thus all our knowledge about them will be based only on a couple of images. However, as humans, we have an enormous

experience with concept learning and priors on what water birds look like. Therefore, it should be easy to identify some common features in the given images and produce a plausible answer. In doing so, we perform inference under assumptions that 1) the order of images is irrelevant, and 2) there exists a correlation between them. Together these two properties informally describe exchangeability excluding the i. i. d. case.

Figure 1.1 An example of a swoose (on the left) – a rare hybrid between a swan (on the right) and a goose.

In machine learning, the problem above is referred to as a few-shot generation, which we can regard as completing a short exchangeable sequence of examples from a previously unseen class. From a learning perspective, one can think of it as learning a new task, i.e. drawing a swoose based on a few examples and knowing how to draw birds in general. A larger set of methods where the same principle applies, is called *meta-learning* or *learning to learn* [107, 119]. Image classification, optimization, imitation learning, reinforcement learning [80, 128, 130, 138] are only a few tasks put into the meta-learning framework in recent years. In many of these cases, the Bayesian approach and the exchangeability assumption that goes with it, are applicable.

Being able to learn how to learn is seen as one of the premises for developing artificial agents that learn and think like humans [74]. Given a soaring interest in artificial general intelligence (AGI) and fascinating results of deep learning methods, the past few years have seen a lot of research in meta-learning and related areas such as transfer, continual and multi-task learning [93, 101, 131]. In this thesis, we contribute to expanding the collection of meta-learning techniques by proposing a novel type of a generative

model, in which the ideas from deep learning and exchangeability are combined.

1.5 Thesis Structure

Chapter 2 provides the necessary background on the two main building blocks for the models presented in the thesis: Gaussian or Student-t processes and normalizing flows. It also explains the meta-learning setup and a short overview of the existing meta-learning approaches. Chapter 3 describes the BRUNO model – our main contribution in this thesis. Chapter 4 outlines a conditional version of de Finetti's theorem and presents conditional BRUNO. We then apply it to the problem of meta reinforcement learning in Chapter 5. Chapter 6 concludes the thesis with a final overview of our work and future directions.

The main chapters expand on the following publications:

1. **I. Korshunova**, J. Degrave, F. Huszár, Y. Gal, A. Gretton, J. Dambre. BRUNO: a deep recurrent model for exchangeable data. *Advances in neural information processing systems 31 (NIPS)*, 2018.

2. **I. Korshunova**, Y. Gal, A. Gretton, and J. Dambre. Conditional BRUNO: a neural process for exchangeable labelled data. *27th European Symposium on Artificial Neural Networks (ESANN)*, 2019.

3. **I. Korshunova**, J. Degrave, A. Gretton, J. Dambre, F. Huszár. Exchangeable models in meta reinforcement learning. *4th Lifelong Learning Workshop at ICML*, 2020.

   The source code accompanying this thesis is publicly available at `github.com/IraKorshunova/bruno` and `github.com/IraKorshunova/bruno-sac`.

Among other contributions are works that were published over the course of the PhD, but are not included in the thesis:

1. B. Sturm, J. F. Santos, **I. Korshunova**. Folk music style modelling by recurrent neural networks with long short term memory units. *16th International Society for Music Information Retrieval Conference (ISMIR)*, late-breaking demo session, 2015.

2. J. Degrave, J. Burms, **I. Korshunova**, J. Dambre. Using deep learning to estimate systolic and diastolic volumes from MRI-images. *Benelearn*, 2016.

3. B. Brinkmann, J. Wagenaar, D. Abbot, P. Adkins, S. Bosshard, M. Chen, Q. Tieng, J. He, F. Muñoz-Almaraz, P. Botella-Rocamora, J. Pardo, F. Zamora-Martinez, M. Hills, W. Wu, **I. Korshunova**, W. Cukierski, C. Vite, E. Patterson, B. Litt, G. Worrell. Crowdsourcing reproducible seizure forecasting in human and canine epilepsy. *Brain* 139 (6): 1713–1722, 2016.

4. **I. Korshunova**, W. Shi, J. Dambre, L. Theis. Fast face-swap using convolutional neural networks. *IEEE International Conference on Computer Vision (ICCV)*, 2017.

5. **I. Korshunova**, P.-J. Kindermans, J. Degrave, T. Verhoeven, B. Brinkmann, J. Dambre. Towards improved design and evaluation of epileptic seizure predictors. *IEEE Transactions on Biomedical Engineering*, vol. 65, no. 3, pp. 502–510, 2018.

6. L. Theis, **I. Korshunova**, A. Tejani, F. Huszár. Faster gaze prediction with dense networks and Fisher pruning. *arXiv preprint arXiv:1801.05787*, 2018.

7. **I. Korshunova**, H. Xiong, M. Fedoryszak, L. Theis. Discriminative topic modeling with logistic LDA. *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 2019.

8. L. Theis, **I. Korshunova**, W. Shi, Z. Wang. Fast face-morphing using neural networks. *United States Patent No. 1055297*, 2020.

2

Background

2.1 Machine Learning

In addition to concepts from Bayesian statistics introduced in the previous chapter, it is useful to review some general machine learning definitions that become helpful in understanding the models we present later in this thesis.

Machine learning can be broadly defined as a study of algorithms that can learn from data to perform tasks without being explicitly programmed to do them. Loosely depending on how the data is used, which data is available, and the purpose of the algorithm, one can identify three basic paradigms in machine learning: supervised, unsupervised and reinforcement learning [8].

Supervised learning deals with problems of learning a mapping $f_{\boldsymbol{\phi}}$ from inputs $\boldsymbol{x}$ to outputs $\boldsymbol{y}$ given a set of training pairs $D_{\text{train}} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \dots (\boldsymbol{x}_N, \boldsymbol{y}_N)\}$. The function $f_{\boldsymbol{\phi}}$ is referred to as a model parameterized by $\boldsymbol{\phi}$. During the learning process (training), we wish to optimize the model's parameters $\boldsymbol{\phi}$ to get the smallest loss on D_{train}, e.g. $\boldsymbol{\phi}^* = \arg\min \sum_{i=1}^{N} \mathcal{L}(\boldsymbol{\phi}; (\boldsymbol{x}_i, \boldsymbol{y}_i))$. The loss function is usually chosen depending on the problem. In regression, where $\boldsymbol{y}$ is continuous, one can use a squared error: $\mathcal{L}(\boldsymbol{\phi}; (\boldsymbol{x}, \boldsymbol{y})) = \|\boldsymbol{y} - f_{\boldsymbol{\phi}}(\boldsymbol{x})\|^2$. When classification is the goal, it is common for the models to output probabilities of categorical labels $p(\boldsymbol{y}|\boldsymbol{x})$. Then training often amounts to maximizing the likelihood of a correct class with $\mathcal{L}(\boldsymbol{\phi}; (\boldsymbol{x}, \boldsymbol{y})) = -\log p_{\boldsymbol{\phi}}(\boldsymbol{y}|\boldsymbol{x})$.

Optimizing these and many other loss functions falls under the framework of maximum likelihood estimation (MLE) in which one wishes to find values of the parameters that maximize the likelihood of observed data under the probability distribution given by the model. Of course, our ultimate goal is not to perform well only on the train set, but rather have a model that can generalize to the unseen examples, which is the most important property of machine learning algorithms.

Unsupervised learning works with settings where no targets $\boldsymbol{y}$, but only inputs $\boldsymbol{x}$ are available. The goals of unsupervised learning are less concrete compared to the supervised case, however, given far greater amounts of unlabelled data, unsupervised techniques play a vital role in the machine learning toolkit. Among the central unsupervised learning tasks are clustering [60], dimensionality reduction [125] and density estimation. The latter aims to determine the distribution of the inputs $p(\boldsymbol{x})$. We will describe two models of this type in the next section.

Unlike the previous two paradigms, reinforcement learning (RL) considers interactive problems where no fixed training or testing datasets are available [115]. The aim is to build software agents capable of learning from their own experiences when taking actions in an environment and trying to maximize cumulative rewards. An elaborate explanation of RL principles will be given in the background section of Chapter 5.

Another relevant notion left for us to review is the dichotomy between discriminative and generative models [9]. These should not be conflated with supervised and unsupervised learning since both discriminative and generative models can be trained with or without supervision. Though, it is indeed less common to find unsupervised discriminative models. For instance, latent Dirichlet allocation [12] is an unsupervised generative model, however, we could build its discriminative variant which still does not require supervision [68]. According to the definition given by Bishop and Lasserre [9], a discriminative model factorizes the joint distribution $p_{\boldsymbol{\phi}}(\boldsymbol{y}, \boldsymbol{x})$ of inputs $\boldsymbol{x}$ and labels $\boldsymbol{y}$ as $p_{\boldsymbol{\phi}}(\boldsymbol{y}|\boldsymbol{x})p(\boldsymbol{x})$, and models only the conditional $p_{\boldsymbol{\phi}}(\boldsymbol{y}|\boldsymbol{x})$. On the other hand, a generative model assigns a joint distribution to all the variables involved, i.e. $p_{\boldsymbol{\phi}}(\boldsymbol{y}, \boldsymbol{x})$ when we have labels or $p_{\boldsymbol{\phi}}(\boldsymbol{x})$ when we do

not. The same principles hold for models with latent variables. In the Gaussian example from Section 1.2, we specified a generative model by defining both $p(\theta)$ and $p(x|\theta)$. More recently, the term "generative" acquired a broader meaning following an intuition that these models can generate new data instances. For example, when $\boldsymbol{x}$ is a high-dimensional object and not merely a class label, models of the conditional distribution $p(\boldsymbol{x}|\boldsymbol{y})$ are sometimes called conditional generative [30]. Using either definition, many of the models we discuss in this thesis are generative, so let us look at what purposes they serve.

If we have a density $p_{\boldsymbol{\phi}}(\boldsymbol{x})$, sampling from this distribution is one of the first tasks we can think of. This so-called synthesis task has been picked up by artists and is since growing popular in creative domains such as music, poetry or painting [29]. More practical applications of generative models include compression, inpainting and denoising of images [118], model-based RL [115], and topic modelling [12]. Without linking to specific applications, generative models can be used in unsupervised feature learning, semi-supervised learning and for improving generalization properties of supervised models.

2.2 Optimization

In the previous section, we mentioned that model training equates to finding the values of parameters $\boldsymbol{\phi}$ in order to minimize some loss function across the training dataset: $\mathcal{L}(\boldsymbol{\phi}) = \sum_{i=1}^{N} \mathcal{L}_i(\boldsymbol{\phi})$. When dealing with high-dimensional parameter spaces, non-convex loss surfaces and big datasets, as we have in case of neural networks, often stochastic gradient descent (SGD) and its variants are the only practical optimization algorithms.

SGD is a simple iterative method that requires only the first-order derivatives of the loss function with respect to $\boldsymbol{\phi}$ in order to find a local minimum. The word "stochastic" reflects the fact that only a random batch of n observations ($1 \leq n < N$) is used at every iteration to compute the loss. For simplicity, we will still denote this loss as $\mathcal{L}(\boldsymbol{\phi})$. At each step of SGD, parameters are updated by following the direction of the steepest descent:

$\phi \leftarrow \phi - \lambda \nabla_\phi \mathcal{L}(\phi)$, where λ is a learning rate. With a carefully tuned λ, these vanilla updates are often sufficient. However, it is common to use additional heuristics to improve convergence and make the algorithm less sensitive to the choice of the learning rate. Among the popular methods are SGD with momentum [102], RMSProp [120] and Adam [64].

2.3 Deep Learning

Deep learning is a subfield of machine learning focusing on methods that are based on deep neural networks (DNN). DNNs are powerful function approximators that, for instance, can learn intricate mappings between inputs and outputs or model distributions of high-dimensional complex data types. Following some important discoveries in the 90s, the full potential of DNNs to tackle difficult problems was illustrated by Krizhevsky et al. [70] in the 2012 ImageNet Large Scale Visual Recognition Challenge [22]. Their approach based on deep convolutional neural networks [79] and efficient use of GPUs significantly outperformed existing state-of-the-art techniques, thus revolutionizing the field of computer vision [77]. Improvements of recurrent neural networks [42] had a similar effect, for instance, in natural language processing and speech processing. In the next section, we will go over these types of DNN architectures in more detail.

Generative modelling is one niche where the progress in deep learning can be readily observed by looking at pictures, reading texts or listening to audio samples that these models produce. For instance, current algorithms are capable of generating faces of non-existing people that are practically indistinguishable from real photos [62]. In Sections 2.3.2 and 2.3.3, we will review two classical deep generative models: variational autoencoders and normalizing flows. Those will be used in constructing BRUNO or when explaining methods related to it.

2.3.1 Basic types of deep neural networks

From a constructional perspective, a DNN is a stack of layers, where each layer performs a certain non-linear transformation of its inputs. By learning parameters of the transformations, DNNs can build a multi-level representation of the data in a way that is useful for a given task. For example, a convolutional network trained to classify images can learn a hierarchy of features ranging from simple edges and shapes in the lower layers to high-level features like eyes or faces in the top layers [84]. In what follows, we will describe a few commonly used types of layers and their associated DNN architectures.

Multi-layer perceptrons

Multi-layer perceptrons (MLP) are vanilla feed-forward neural networks that use dense layers as their main building block. If we let $\boldsymbol{y}_{n-1}$ denote the input vector to the n-th dense layer, then its output $\boldsymbol{y}_n$ is computed as:

$$\boldsymbol{y}_n = f(\boldsymbol{W}_n \boldsymbol{y}_{n-1} + \boldsymbol{b}_n),$$

where $\boldsymbol{W}_n$ is a matrix of weights, $\boldsymbol{b}_n$ is a vector of biases, and f is an elementwise nonlinearity, also known as an activation function. Widely used choices of activations include sigmoids, e.g $(1 + e^{-x})^{-1}$ or $\tanh(x)$, and rectifiers such as $\max(0, x)$ and its variants [19, 39, 87]. Given an input vector $\boldsymbol{x}$, a neural network of N layers implements a sequence of mappings $\boldsymbol{x} = \boldsymbol{y}_0 \mapsto \boldsymbol{y}_1 \mapsto \boldsymbol{y}_2, \ldots, \mapsto \boldsymbol{y}_N$, where $\boldsymbol{y}_N$ is the output. Training of this neural network amounts to minimizing some loss function with respect to $\boldsymbol{\phi} = \{\boldsymbol{W}_n, \boldsymbol{b}_n\}_{n=1,\ldots N}$ – a set of weights and biases from all the layers. To use the gradient-based optimization as described in Section 2.2, we can compute partial derivatives of the loss with respect to individual parameters by a successive application of the chain rule starting from the last layer. In the context of deep learning, this method is called backpropagation [102].

Convolutional neural networks

Densely connected networks are efficient in processing data that does not exhibit spatial or temporal structure. Therefore, they are not particularly suited for images, where such structure exists. In this case, one should opt for a convolutional neural network (CNN) – a type of DNNs that encodes relevant inductive biases, or in other words, a set of assumptions derived from our prior knowledge about the nature of the task. For instance, we know that objects are likely to change their positions on different pictures. Thus, in order to locate or classify them, we should at least make parts of our model equivariant to translations. By saying that some function $f(x)$ is equivariant, we mean that $f(g(x)) = g(f(x))$, where in our case, g is a translation transformation. In CNNs, this type of equivariance is implemented by restricting the connectivity patterns and sharing of the weights. Unlike dense layers, whose inputs and outputs are vectors, convolutional layers work with feature maps. For example, an RGB image can be represented as three feature maps $\boldsymbol{X}^1$, $\boldsymbol{X}^2$, $\boldsymbol{X}^3$, one per each colour channel. An application of the convolutional layer to this input can be written as:

$$\boldsymbol{Y}^l = f\big(\sum_{k=1}^{K} \boldsymbol{W}^{k,l} * \boldsymbol{X}^k + b^l\big),$$

where $*$ denotes the 2D convolution, each $\boldsymbol{W}^{k,l}$ is a matrix of weights, and b^l is a bias. Here, $\boldsymbol{Y}^l$ represents only one of the output feature maps. Ranging l from 1 to L yields a stack of output feature maps that serve as inputs to the next layer. In classification tasks, the stack of convolutional layers usually ends with an MLP, whose last softmax layer outputs the probability over classes.

Recurrent neural networks

As we mentioned earlier, RNNs are powerful autoregressive models for sequential data in which the order of inputs matters. RNNs have an internal state $\boldsymbol{h}$ that evolves over time as the network gets more inputs. Namely, given a previous state $\boldsymbol{h}_{t-1}$ and a current input $\boldsymbol{x}_t$, the simplest RNN update rule equates to:

$$\boldsymbol{h}_t = f(\boldsymbol{W}_{in}\boldsymbol{x}_t + \boldsymbol{U}\boldsymbol{h}_{t-1} + \boldsymbol{b}), \tag{2.1}$$

where $\boldsymbol{W}_{in}$ and $\boldsymbol{U}$ are the weights, $\boldsymbol{b}$ is the vector of biases, and f is usually a tanh nonlinearity. At every step, we can also compute the output as a function of the state as $\boldsymbol{o}_t = g(\boldsymbol{W}_{out}\boldsymbol{h}_t + \boldsymbol{b}_{out})$. For simplicity, we assumed single-layer transformations, though in general, the mappings between $\boldsymbol{x}$, $\boldsymbol{h}$ and $\boldsymbol{o}$ can be deep.

Unrolling the RNN, that is writing down Eq. 2.1 for every step $t = 1, \ldots, T$, results in an equivalent feed-forward network with T layers with shared parameters as illustrated in Figure 2.1. The longer the sequence, the deeper this network becomes. With increasing depth, our simple update rule usually becomes inadequate, as it may result in training instabilities. Moreover, in practice, such an RNN is unable to maintain long-term dependencies that are crucial for certain applications. Improved update rules as implemented by the long short-term memory (LSTM) [52] or gated recurrent units [18] eliminate these issues.

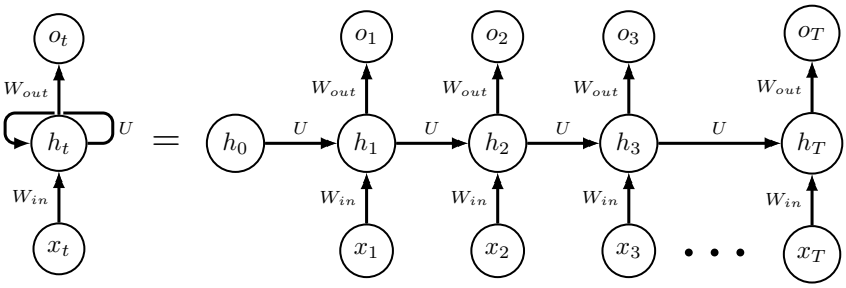


Figure 2.1 A schematic of a simple RNN: on the left in a compact representation containing a loop, and on the right its unrolled version for T steps.

In many tasks, $\boldsymbol{o}_t$ is designed to represent the distribution $p(\boldsymbol{x}_{t+1}|\boldsymbol{h}_t)$ of the next input in the sequence. In theory, the state $\boldsymbol{h}_t$ encapsulates the history of observations $\boldsymbol{x}_{1:t}$, thus $p(\boldsymbol{x}_{t+1}|\boldsymbol{h}_t)$ corresponds to the predictive distribution $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_{1:t})$. Training of the whole model amounts to maximizing the one-step-ahead predictions $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_{1:t})$ for the observed data with respect to the weights and biases in all the layers. It is equivalent to maximizing the likelihood of all observations since the joint distribution $p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$ can be factorized as $\prod_{t=0}^{T-1} p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_{1:t})$.

2.3.2 Variational Autoencoders

The variational autoencoder (VAE) [66] is currently one of the most popular algorithms for implementing learning in deep latent variable models. A few meta-learning methods related to BRUNO, which we will discuss in later chapters, are based on VAEs. Therefore, it is useful to understand their construction and the associated concepts, such as variational inference, amortization and the reparameterization trick.

Consider a latent variable model that specifies a joint distribution $p_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{z})$ over an observed variable $\boldsymbol{x}$ and a latent variable $\boldsymbol{z}$. In particular, we are interested in models whose joint distribution is factorized as $p_{\boldsymbol{\phi}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\phi}}(\boldsymbol{z})$. In order to learn the parameters $\boldsymbol{\phi}$ via MLE, we need to compute the marginal probability of $\boldsymbol{x}$, also known as the *evidence*:

$$p_{\phi}(\boldsymbol{x}) = \int p_{\boldsymbol{\phi}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\phi}}(\boldsymbol{z})d\boldsymbol{z}. \tag{2.2}$$

The integral in Eq. 2.2 has a closed-form solution only for a small class of distributions, which would be of little use when trying to model complex data. To allow for more expressive models, it is common to approximate this intractable integral.

The intractability of the marginal distribution is related to the intractability of the posterior $p_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ since the two are connected via $p_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = p_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{z})/p_{\boldsymbol{\phi}}(\boldsymbol{x})$. Variational inference techniques aim to approximate the intractable posterior by a simpler variational distribution $q_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{x})$, i.e. $p_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \approx q_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{x})$. In VAEs, $q_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2\boldsymbol{I})$ where $\boldsymbol{I}$ is the identity matrix and $(\boldsymbol{\mu}, \boldsymbol{\sigma}) = \text{MLP}_{\boldsymbol{\psi}}(\boldsymbol{x})$, meaning that parameters of this Gaussian distribution are the outputs of the MLP that takes $\boldsymbol{x}$ as an input, and whose weights and biases constitute $\boldsymbol{\psi}$. Using a common terminology, we will say that the MLP parameterizes the distribution q. VAEs call this MLP the encoder or the inference network.

VAEs use a single inference network to compute the approximate posterior for all points in the dataset. This so-called, amortized inference, deviates from more traditional practices, where each

data point $\boldsymbol{x}_i$ gets a separate $\boldsymbol{\psi}_i$ parameter [11]. The latter technique might be costly for large datasets since the number of parameters grows with the number of observations, and each of these parameters requires a separate optimization loop. VAEs, on the other hand, do not have these scaling issues.

In addition to the encoder, a VAE defines a decoder or a generation network that takes $\boldsymbol{z}$ as an input, and outputs parameters of the distribution $p_{\boldsymbol{\phi}}(\boldsymbol{x}|\boldsymbol{z})$. The encoder-decoder pair that makes a VAE is illustrated in Figure 2.2.

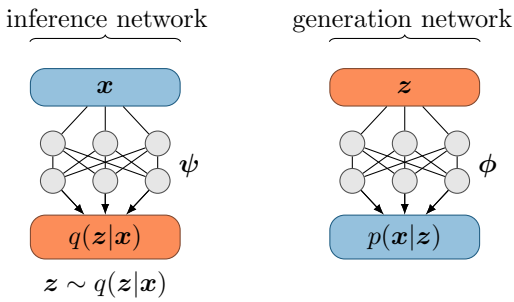


Figure 2.2 Encoder-decoder architecture of the VAE. The encoder, or the inference network, is an MLP that parameterizes the variational posterior over the latent variable $\boldsymbol{z}$. Samples from this posterior are then fed into the decoder that parameterizes $p(\boldsymbol{x}|\boldsymbol{z})$, from which new instances of $\boldsymbol{x}$ can be sampled.

Let us return to the question of approximating the marginal distribution $p_{\boldsymbol{\phi}}(\boldsymbol{x})$. Using $q_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{x})$ instead of the true intractable posterior $p_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$, allows to obtain the following bound on $\log p_{\boldsymbol{\phi}}(\boldsymbol{x})$:

$$\begin{aligned}\log p_{\phi}(\boldsymbol{x}) &\geq \mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\psi}; \boldsymbol{x}) \\ &:= \mathbb{E}_{\boldsymbol{z}\sim q_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\phi}}(\boldsymbol{x}|\boldsymbol{z})] - \mathrm{KL}[q_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\phi}}(\boldsymbol{z})],\end{aligned} \tag{2.3}$$

where KL is the Kullback–Leibler divergence between two distributions. The term $\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\psi}; \boldsymbol{x})$ is called the evidence lower bound (ELBO). It can serve as a surrogate objective in the MLE framework, where one needs to find variational parameters $\boldsymbol{\psi}$ and generative parameters $\boldsymbol{\phi}$ that maximize ELBO across the training data, i.e. $\max_{\boldsymbol{\phi},\boldsymbol{\psi}} \sum_{i=1}^{N} \mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\psi}; \boldsymbol{x}_i)$.

It is not straightforward to reliably estimate the gradient of the ELBO with respect to variational parameters $\boldsymbol{\psi}$ as those appear

in the distribution under which the expectation of the likelihood in Eq. 2.3 is taken [83]. VAEs use the reparameterization trick to obtain a differentiable Monte Carlo estimator of this expectation. The idea is to express $\boldsymbol{z}$ as a deterministic function $g_{\boldsymbol{\psi}}$ of an auxiliary noise variable $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$, i.e. $\boldsymbol{z} = g_{\boldsymbol{\psi}}(\boldsymbol{x}, \boldsymbol{\epsilon})$. In Gaussian VAEs with $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2 \boldsymbol{I})$ and $(\boldsymbol{\mu}, \boldsymbol{\sigma}) = \text{MLP}_{\boldsymbol{\psi}}(\boldsymbol{x})$, we have:

$$\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} \text{ with } \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}),$$

where $\odot$ denotes an elementwise product. Using this trick, we get the following Monte Carlo estimator for the expectation term in Eq. 2.3:

$$\mathbb{E}_{\boldsymbol{z} \sim q_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\phi}}(\boldsymbol{x}|\boldsymbol{z})] \approx \frac{1}{L} \sum_{l=1}^{L} \log p_{\boldsymbol{\phi}}(\boldsymbol{x}|g_{\boldsymbol{\psi}}(\boldsymbol{x}, \boldsymbol{\epsilon}^{(l)}))$$
$$\text{with } \boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}).$$

Given the above estimator and the analytic form for the KL divergence $\text{KL}[q_{\boldsymbol{\psi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\phi}}(\boldsymbol{z})]$ between a Gaussian variational distribution and a fixed Gaussian prior $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, one can now train a VAE using standard stochastic gradient methods.

2.3.3 Normalizing Flows

Normalizing flows [117] define a general framework for constructing complex probability distributions by pushing simple base distributions through a series of differentiable bijective mappings that typically incorporate neural networks [92]. According to Papamakarios et al. [92], there are two major groups of flow-based models: flows composed of a finite number of simple transformations and continuous-time flows defined by ordinary differential equations. While most flows deal with real-valued inputs and transformations in the Euclidean space, some of the recent models include flows for discrete random variables [54, 122] and flows on spheres [98], among others. The progress is likely to continue since normalizing flows are currently an active area of research.

Real-valued non-volume preserving transformation (Real NVP) [24] is one of the flow models that became popular due to its

efficiency and relative simplicity. Real NVP proposes a design for a bijective function $f : \mathcal{X} \mapsto \mathcal{Z}$ with $\mathcal{X} = \mathbb{R}^D$ and $\mathcal{Z} = \mathbb{R}^D$ such that **(a)** the inverse is easy to evaluate, i.e. the cost of computing $\boldsymbol{x} = f^{-1}(\boldsymbol{z})$ is the same as for the forward mapping $\boldsymbol{z} = f(\boldsymbol{x})$, and **(b)** computing the Jacobian determinant takes linear time in the number of dimensions D. Real NVP assumes a simple distribution for $\boldsymbol{z}$, so one can use a change of variables formula to evaluate $p(\boldsymbol{x})$:

$$p(\boldsymbol{x}) = p(\boldsymbol{z}) \left| \det \left(\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \right) \right|.$$

In Real NVP, as in its predecessors [21, 23] and other standard flow models [92], $p(\boldsymbol{z})$ is chosen to be a fully factorized distribution, i.e. $p(\boldsymbol{z}) = \prod_{i=1}^{D} p(z_i)$, usually with $p(z_i) = \mathcal{N}(0, 1)$. This implies that by training the model to maximize the likelihood of inputs in $\mathcal{X}$, we are trying to find a mapping $\boldsymbol{z} = f(\boldsymbol{x})$ such that the components of $\boldsymbol{z}$ are independent and normally distributed.

The Real NVP transformation f is defined as a sequence of coupling layers, where each layer implements a mapping $\mathcal{Y}_{in} \mapsto \mathcal{Y}_{out}$ that transforms half of its inputs while copying the other half directly to the output:

$$\begin{cases} \boldsymbol{y}_{out}^{1:d} = \boldsymbol{y}_{in}^{1:d} \\ \boldsymbol{y}_{out}^{d+1:D} = \boldsymbol{y}_{in}^{d+1:D} \odot \exp(s(\boldsymbol{y}_{in}^{1:d})) + t(\boldsymbol{y}_{in}^{1:d}), \end{cases} \tag{2.4}$$

where $\odot$ is an elementwise product, s (scale) and t (translation) are arbitrarily complex functions, e.g. convolutional neural networks whose outputs match the dimensionality of $\boldsymbol{y}_{in}^{d+1:D}$. The inverse of this mapping is given by the following transformation:

$$\begin{cases} \boldsymbol{y}_{in}^{1:d} = \boldsymbol{y}_{out}^{1:d} \\ \boldsymbol{y}_{in}^{d+1:D} = \left(\boldsymbol{y}_{out}^{d+1:D} - t(\boldsymbol{y}_{out}^{1:d})\right) \odot \exp(-s(\boldsymbol{y}_{out}^{1:d})). \end{cases}$$

It indicates that the backward pass through the network requires the same amount of compute as the forward pass.

One can show that the coupling layer is a bijective mapping with a triangular Jacobian, and that a composition of coupling layers preserves these properties. The conditioning structure of

coupling layers gives Real NVP the ability to perform sampling and density evaluation in a single pass. This comes at a cost of expressive power: a coupling layer is not a universal density approximator [92]. In practice, however, one can obtain a highly nonlinear mapping $f(\boldsymbol{x})$ by stacking coupling layers $\mathcal{X} \mapsto \mathcal{Y}_1 \mapsto \mathcal{Y}_2 \cdots \mapsto \mathcal{Y}_K \mapsto \mathcal{Z}$ while alternating the dimensions that are being copied to the output.

The coupling layer splits its input into halves and elementwise transforms the second part as a function of the first. In the other extreme case, one could split the input into D parts and transform each of them as function of previous parts, e.g. $y^i_{out} = y^i_{in} \exp(s(\boldsymbol{y}^{1:i-1}_{in})) + t(\boldsymbol{y}^{1:i-1}_{in})$. This would constitute a fully autoregressive flow layer, and depending on the implementation, give rise to flow models such as Masked Autoregressive Flow (MAF) [91] or Inverse Autoregressive Flow (IAF) [67]. Unlike Real NVP, these models, in theory, can express any distribution $p(\boldsymbol{x})$. However, they do not have the computational symmetry: MAF requires D sequential passes to generate samples and a single pass for density evaluation. For IAF, the reverse relations hold. These properties should be kept in mind when choosing a flow model.

The real-valued flows are designed to model densities, which means their inputs are supposed to be instances of a continuous random variable. If the inputs are discrete, the continuous model will collaps to a degenerate solution, where it places high probability spikes on the discrete values [124]. Thus, when dealing with images, usually stored as 8-bit integer arrays, it is common to dequantize pixel values $\boldsymbol{x} \in [0, 255]^D$ by adding uniform noise $\boldsymbol{u} \in [0, 1)^D$ and then model the density of $\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{u}$. Theis et al. [118] showed, that by training a continuous model $p_{\text{model}}(\boldsymbol{y})$ on data $\boldsymbol{y} \sim p_{\text{data}}$, we effectively maximize the lower bound on the expected log-likelihood of the original integer-valued data. The latter can be written as $\mathbb{E}_{\boldsymbol{x} \sim P_{\text{data}}} \big[\log P_{\text{model}}(\boldsymbol{x}) \big]$, where P_{data} is the discrete data distribution and $P_{\text{model}}(\boldsymbol{x})$ is the log-likelihood of $\boldsymbol{x}$ under a discrete model. The reasoning of Theis et al. [118] goes as

follows:

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{y}\sim p_{\text{data}}}\big[\log p_{\text{model}}(\boldsymbol{y})\big] &= \sum_{\boldsymbol{x}} P_{\text{data}}(\boldsymbol{x}) \int_{[0,1)} \log p_{\text{model}}(\boldsymbol{x}+\boldsymbol{u}) d\boldsymbol{u} \\
\text{(by Jensen's inequality)} \qquad &\leq \sum_{\boldsymbol{x}} P_{\text{data}}(\boldsymbol{x}) \log \int_{[0,1)} p_{\text{model}}(\boldsymbol{x}+\boldsymbol{u}) d\boldsymbol{u} \\
&= \mathbb{E}_{\boldsymbol{x}\sim P_{\text{data}}}\big[\log P_{\text{model}}(\boldsymbol{x})\big].
\end{aligned}$$

Two important conclusions are to be made from this derivation. Firstly, a continuous model trained on data, which is dequantized using uniform noise, cannot collapse to a degenerate solution since its expected log-likelihood is bounded from above. Secondly, one needs to be cautious when using average log-likelihoods to benchmark models in which likelihoods might have different meanings.

There is another issue we need to address when working with images or other types of data whose values are limited to a certain range. During generation, we first sample $\boldsymbol{z}$ from a distribution whose domain is unbounded and then compute the inverse mapping $\boldsymbol{x} = f^{-1}(\boldsymbol{z})$. This procedure gives no guarantees that values of $\boldsymbol{x}$ fall within a permissible range. A common solution is to include an extra transformation as a first layer of the flow model that rescales $\boldsymbol{x}$ (dequantized if needed) to a $[0, 1)$ interval, and then applies an elementwise function $f(x) = \text{logit}(\alpha + (1 - 2\alpha)x)$ with some small α [24]. The inverse of this logit function produces outputs between $\frac{-\alpha}{1-2\alpha}$ and $\frac{1-\alpha}{1-2\alpha}$, which can then be rescaled and safely clipped to the range of the original data.

2.4 Meta-learning

Most of the models we discussed above are domain-specific. For example, a DNN trained to classify between cats and dogs is likely to succeed exclusively in that single task. Moreover, DNNs are known to be data-hungry, requiring a large number of training samples in order to perform well. This is in contrast to how humans learn: we can use past experiences to learn quickly from a small number of examples. Meta-learning tries to imitate this process in a way we formalize below. Our explanation will focus

on the image classification task, however, the same principles apply to other tasks such as image generation or policy adaptation in reinforcement learning [25, 130].

The distinguishing feature of meta-learning is that training is done on a number of tasks, where each task T is associated with some dataset D. For example, classifying cats versus dogs can be one task, and classifying birds versus fish can be another. In general, a meta-learning model aims to minimize some loss $\mathcal{L}$ across the distribution of tasks with respect to model parameters $\boldsymbol{\phi}$ [106]:

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} \mathbb{E}_{D \sim p(D)}[\mathcal{L}(\boldsymbol{\phi}; D)].$$

Often, in supervised learning problems, the training proceeds in an episodic manner by randomly sampling from each dataset $D = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$ two sets: a *support set* S_D (usually small) and a *query set* Q_D [128]. The meta-learning model can use the information in the support set to compute probability $p_{\boldsymbol{\phi}}(y|\boldsymbol{x}, S)$ of a label y associated with an observation $\boldsymbol{x}$ from the query set. We wish to maximize the probability of correct labels across the training tasks with respect to the model parameters:

$$\boldsymbol{\phi}^* = \arg\max_{\boldsymbol{\phi}} \mathbb{E}_D \Big[\mathbb{E}_{S_D, Q_D} \Big[\sum_{(\boldsymbol{x}, y) \in Q_D} \log p_{\boldsymbol{\phi}}(y|\boldsymbol{x}, S_D) \Big] \Big].$$

This training objective matches the desired behaviour at test time: the model might be asked to classify images from previously unseen classes while relying on a few labelled examples in the support set and its ability to learn rapidly from those examples.

It is common to group meta-learning methods into model-, metric- and optimization-based approaches [126] depending on how $p_{\boldsymbol{\phi}}(y|\boldsymbol{x}, S)$ is defined.

- Model-based methods specify this probability using a neural network $f_{\boldsymbol{\phi}}(\boldsymbol{x}, S)$ that can quickly adapt to new tasks either because of its own structure [106] or when it is a subordinate network governed by a supervisory system [53, 85].
- Metric-based models take a non-parametric approach akin to k-nearest neighbours algorithm and kernel density

estimation. Their main idea is to learn a similarity function $k_{\boldsymbol{\phi}}$ over pairs of inputs, and use it to relate a query to examples in the support set. For instance, in matching networks [128], $k_{\boldsymbol{\phi}}$ weighs the labels in S, i.e. $p_{\boldsymbol{\phi}}(y|\boldsymbol{x}, S) \propto \sum_{(\boldsymbol{x}_i, y_i) \in S} k_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{x}_i) y_i$.

- Optimization-based methods represent $p_{\boldsymbol{\phi}}(y|\boldsymbol{x}, S)$ as a mapping $f_{\boldsymbol{\phi}(S)}(\boldsymbol{x})$ whose parameters $\boldsymbol{\phi}(S)$ depend on the support set via the updates of some initial parameters $\boldsymbol{\phi}_0$. In general, these updates can be written as: $\boldsymbol{\phi}(S) = g_{\boldsymbol{\psi}}(\boldsymbol{\phi}_0, \{\nabla_{\boldsymbol{\phi}_0} \mathcal{L}(\boldsymbol{\phi}_0; \boldsymbol{x}_i, y_i)\}_{(\boldsymbol{x}_i, y_i) \in S})$ [126]. Stochastic gradient descent is special case of this update rule: $\boldsymbol{\phi}(S) = \boldsymbol{\phi}_0 - \psi \sum_{(\boldsymbol{x}_i, y_i) \in S} \nabla_{\boldsymbol{\phi}_0} \mathcal{L}(\boldsymbol{\phi}_0; \boldsymbol{x}_i, y_i)$. Its use as a meta-optimizer was proposed by MAML [34] – one of the most popular methods in this group.

There also exist methods that belong to more than one group. For example, Meta PixelCNN [96] combines the ideas from model-based and optimization-based approaches.

As with the ImageNet's role in promoting deep learning, the advances in meta-learning would not have been possible without datasets. In 2015, Lake et al. [72] released the Omniglot dataset together with five concept learning tasks where the challenge was to build a single model to solve them all at a human level. The tasks were: one-shot classification, one-shot generation, parsing and generation of new concepts either unconditionally or given their type. Being perceived as the most relevant and perhaps the easiest of all tasks, one-shot classification became mainstream while less research was dedicated to the other four. The Omniglot challenge thus remains [73], even though its relevance is debated.

The Omniglot itself is a dataset of 1623 handwritten characters from 50 alphabets, where each character is drawn by 20 different people. Examples of characters grouped by alphabet are given in Figure 2.3. This is unlike the famous MNIST [78] dataset of handwritten digits with only 10 classes and 7000 examples per class. Having a large number of classes with relatively few data points per class makes Omniglot suitable for testing few-shot classification and generation methods that we present in this thesis.

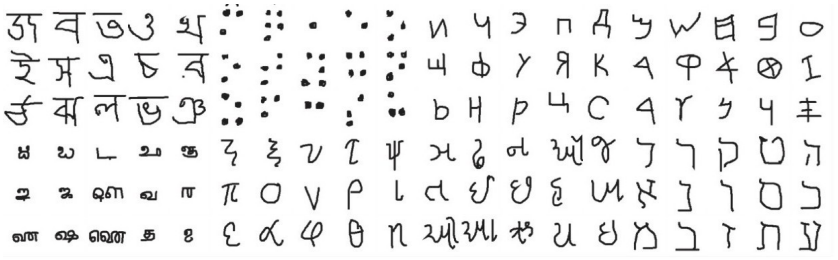


Figure 2.3 Examples of Omniglot characters grouped by alphabet (adapted from Lake et al. [72]).

2.5 Gaussian and Student-t Processes

A Gaussian process (GP) is a stochastic process defined as a collection of random variables, in which any finite collection has a multivariate normal distribution [95]. For example, if we index the random variables in $\mathcal{Y}$ using a set of integers as $y_1, y_2, y_3, \ldots$, then in a GP, $(y_1, y_2, \ldots y_n) \sim \mathcal{N}_n(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. GPs can also be viewed as a generalization of a Gaussian distribution to infinite dimensions. In other words, while a multivariate Gaussian describes a distribution of a random vector, a Gaussian process describes properties of a function – an infinite-dimensional object. This also implies that the space of parameters is infinite-dimensional, which is the reason why GPs are classified as non-parametric models. If dealing with infinite-dimensional objects feels uncomfortable, one can imagine functions as very long vectors. After all, computations required by GPs make the difference imperceivable as long as we work with finite datasets.

The marginalization property is what makes GPs practical. For instance, if a GP specifies $p(y_1, y_2) = \mathcal{N}_2(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, then it also needs to specify the marginal $p(y_1) = \mathcal{N}(\mu_1, \boldsymbol{\Sigma}_{11})$. This property holds irrespective of the number of variables, even when the larger set contains infinitely many variables.

It is most common to see GPs used in regression problems where y depends on some input vector $\boldsymbol{x} \in \mathcal{X}$ [95]. In this case, both the mean and covariance functions depend on $\boldsymbol{x}$. The latter is usually defined using a kernel $k(\boldsymbol{x}, \boldsymbol{x}')$ – a closed-form expression for the

dot product between infinitely many features of $\boldsymbol{x}$ and $\boldsymbol{x}'$. In this case, it is common to write: $y(\boldsymbol{x}) \sim \mathcal{GP}(\mu(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'))$.

The adoption of kernels makes GP an expressive model, which is especially well suited for small datasets – a regime where deep neural networks tend to struggle. Moreover, GPs are valuable for providing predictive uncertainty. Namely, instead of giving the point estimate for the function value y^* at the location $\boldsymbol{x}^*$, GP outputs a normal distribution $p(y^*|\boldsymbol{x}^*, y_{1:n}, \boldsymbol{x}_{1:n})$, where $\{(y_1, \boldsymbol{x}_1), \ldots, (y_n, \boldsymbol{x}_n)\}$ is a set of training points. On the downside, GPs are known to be computationally expensive as their predictive distribution requires $n \times n$ covariance matrix inversion, thus entailing a cubic cost $\mathcal{O}(n^3)$.

A GP can be seen as a limiting case of a Student-t process (TP) – the most general elliptically symmetric processes with an analytically representable density [109, 136]. Similarly to the definition of GPs, a TP is a random process where any finite collection of variables has a multivariate Student-t distribution: $(y_1, y_2, \ldots y_n) \sim MVT_n(\nu, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ with degrees of freedom $\nu \in \mathbb{R}_+ \setminus [0, 2]$, mean $\boldsymbol{\mu} \in \mathbb{R}^n$ and a positive definite $n \times n$ covariance matrix $\boldsymbol{\Sigma}$. Denoting $\boldsymbol{y} = (y_1, y_2, \ldots y_n)$, the density of MVT_n is given by

$$
\begin{aligned}
p(\boldsymbol{y}) = & \frac{\Gamma(\frac{\nu+n}{2})}{((\nu-2)\pi)^{n/2}\Gamma(\nu/2)}|\boldsymbol{\Sigma}|^{-1/2} \\
& \times \left(1 + \frac{(\boldsymbol{y}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{y}-\boldsymbol{\mu})}{\nu-2}\right)^{-\frac{\nu+n}{2}}.
\end{aligned}
$$

The multivariate t-distribution can be derived by integrating over a scale parameter for the covariance. Namely, when

$$
r \sim \text{Inv-Gamma}(\frac{\nu}{2}, \frac{1}{2}) \text{ and } \boldsymbol{y}|r \sim \mathcal{N}_n(\boldsymbol{\mu}, r(\nu-2)\boldsymbol{\Sigma}),
$$

then $\boldsymbol{y} \sim MVT_n(\nu, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ [95, 109, 136]. Shah et al. [109] showed that the same marginal distribution can be obtained from a TP which was derived by placing an inverse Wishart process prior on the kernel function.

Finding a predictive distribution in case of GPs or TPs equates to computing a conditional distribution in jointly Gaussian or

Student-t distributions. Suppose we can partition $\boldsymbol{y}$ into two consecutive parts $\boldsymbol{y}_a \in \mathbb{R}^{n_a}$ and $\boldsymbol{y}_b \in \mathbb{R}^{n_b}$, such that

$$\begin{bmatrix} \boldsymbol{y}_a \\ \boldsymbol{y}_b \end{bmatrix} \sim MVT_n\left(\nu, \begin{bmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{aa} & \boldsymbol{\Sigma}_{ab} \\ \boldsymbol{\Sigma}_{ba} & \boldsymbol{\Sigma}_{bb} \end{bmatrix}\right).$$

Then conditional distribution $p(\boldsymbol{y}_b|\boldsymbol{y}_a)$ is given by

$$\begin{aligned} &p(\boldsymbol{y}_b|\boldsymbol{y}_a) = MVT_{n_b}\Big(\nu + n_a, \tilde{\boldsymbol{\mu}}_{\boldsymbol{b}}, \frac{\nu + \beta_a - 2}{\nu + n_a - 2}\tilde{\boldsymbol{\Sigma}}_{bb}\Big), \text{ where} \\ &\tilde{\boldsymbol{\mu}}_{\boldsymbol{b}} = \boldsymbol{\Sigma}_{ba}\boldsymbol{\Sigma}_{aa}^{-1}(\boldsymbol{y}_a - \boldsymbol{\mu}_a) + \boldsymbol{\mu}_b \\ &\beta_a = (\boldsymbol{y}_a - \boldsymbol{\mu}_a)^T\boldsymbol{\Sigma}_{aa}^{-1}(\boldsymbol{y}_a - \boldsymbol{\mu}_a) \\ &\tilde{\boldsymbol{\Sigma}}_{bb} = \boldsymbol{\Sigma}_{bb} - \boldsymbol{\Sigma}_{ba}\boldsymbol{\Sigma}_{aa}^{-1}\boldsymbol{\Sigma}_{ab}. \end{aligned} \tag{2.5}$$

The expressions for the mean and covariance are the same in GPs, where $p(\boldsymbol{y}_b|\boldsymbol{y}_a) = \mathcal{N}_{n_b}(\tilde{\boldsymbol{\mu}}_{\boldsymbol{b}}, \tilde{\boldsymbol{\Sigma}}_{\boldsymbol{bb}})$. Here, we see that the conditional distribution requires a matrix inverse – the major computational bottleneck in both GPs and TPs. But do not worry: in this thesis, we will be dealing with a special case of GPs and TPs with linear time and constant memory complexity.

The parameter ν, representing the degrees of freedom, has a large impact on the behaviour of TPs. It controls how heavy-tailed the t-distribution is: as ν increases, the tails get lighter and the t-distribution gets closer to the Gaussian. From Eq. 2.5, we can see that as ν or n_a tends to infinity, the predictive distribution tends to the one from a GP, while for small ν and n_a, a TP would give less certain predictions than its corresponding GP.

A second feature of the TP is the scaling of the predictive variance with a β_a coefficient, which explicitly depends on the values of the conditioning observations. Looking at the weight $(\nu+\beta_a-2)/(\nu+n_a-2)$, we see that the variance of $p(\boldsymbol{y}_b|\boldsymbol{y}_a)$ is increased over the Gaussian default when $\beta_a > n_a$, and is reduced otherwise. In other words, when the samples are dispersed more than they would be under the Gaussian distribution, the predictive uncertainty is increased compared with the Gaussian case.

3

BRUNO

3.1 Introduction

In 2015, machine learning algorithms were confronted with the inability to model two important aspects of human learning: generalization from a single or few examples, and forming abstract, rich, and flexible representations that can be used for a multitude of functions [72]. Since then, a lot of progress has been made, especially in discriminative settings, e.g. [17, 133]. In the field of generative image modelling, however, the problem of a few-shot generation remains challenging, and at present, there are only few flexible deep generative models to solve this problem [3, 20, 28].

In this chapter, we present BRUNO – a model that combines elements of implicit and explicit approaches to Bayesian inference as discussed in Section 1.3, and is suitable for both few-shot generation and classification. BRUNO is *provably exchangeable* and makes use of deep features learned from observations so as to model complex data types such as images. The main idea behind BRUNO is to construct a *bijective* mapping between random variables $\boldsymbol{x}_i \in \mathcal{X}$ in the observation space and their features $\boldsymbol{z}_i \in \mathcal{Z}$. In the feature space, we then define a simple exchangeable model for sequences $\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3, \ldots$, where $p(\boldsymbol{z}_{n+1}|\boldsymbol{z}_{1:n})$ can be easily computed using a recurrent formulation. Inverting the mapping between $\mathcal{Z}$ and $\mathcal{X}$ allows us to evaluate and sample from

the predictive distribution $p(\boldsymbol{x}_{n+1}|\boldsymbol{x}_{1:n})$. BRUNO's construction avoids computing the posterior over the latent variables, however, if we wish to do so, an alternative formulation of the model allows for it.

The rest of this chapter is structured as follows. In Section 3.2 we look at a few methods selected to highlight the relation of our work with previous approaches to modelling exchangeable data. In Section 3.3 we describe BRUNO, basing our explanation on the background information from Chapter 2. Section 3.4 provides an alternative formulation of our model in which the posterior is represented explicitly. In Section 3.5, we illustrate the use of BRUNO for few-shot image generation, classification, and set anomaly detection. The discussion follows in Section 3.6.

3.2 Related work

Bayesian sets [38] aim to model exchangeable sequences of binary random variables by analytically computing the integrals involving the posterior:

$$\begin{aligned} p(\boldsymbol{x}_{1:n}) &= \int \prod_{i=1}^{n} p(\boldsymbol{x}_i|\boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta} \\ p(\boldsymbol{x}_{n+1}|\boldsymbol{x}_{1:n}) &= \int p(\boldsymbol{x}_{n+1}|\boldsymbol{\theta}) p(\boldsymbol{\theta}|x_{1:n}) d\boldsymbol{\theta} \end{aligned} \tag{3.1}$$

This is made possible by using a Bernoulli distribution for the likelihood and a beta distribution for the prior. To apply this method to other types of data, e.g. images, one needs to engineer a set of binary features [49]. In that case, there is usually no one-to-one mapping between the input space $\mathcal{X}$ and the features space $\mathcal{Z}$: in consequence, it was not possible to use it as a generative model, so the applications were limited to retrieval.

One group of methods extending this line of research to more expressive distributions renders the integrals in Eq. 3.1 intractable and so the variational approximations are employed. These methods are based on the ideas from variational autoencoders [66, 97] and how to apply them to sets. Neural statistician [28] is one

example. As in VAEs, neural statistician learns an approximate inference network over the latent variable $\boldsymbol{z}_i$ corresponding to observations $\boldsymbol{x}_i$. However, it also implements an approximate inference over a latent variable $\boldsymbol{c}$ – a context that is global to the set $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$. The architecture for the inference network $q(\boldsymbol{c}|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ maps every $\boldsymbol{x}_i$ into a feature vector and applies a mean pooling operation across these representations. The resulting vector is then used to produce parameters of a Gaussian distribution over $\boldsymbol{c}$. Mean pooling makes $q(\boldsymbol{c}|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ invariant under permutations of the inputs. In addition to the inference networks, the neural statistician also has a generative component $p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n|\boldsymbol{c})$ which assumes that $\boldsymbol{x}_i$'s are independent given $\boldsymbol{c}$. Here, it is easy to see that $\boldsymbol{c}$ plays the role of θ from Eq. 3.1. In the neural statistician, it is intractable to compute $p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, so its variational lower bound is used instead. In our model, we perform an implicit inference over θ and can exactly compute predictive distributions and the marginal likelihood. Despite these differences, both neural statistician and BRUNO can be applied to similar tasks, namely few-shot classification and image generation. Among other methods that rely on variational inference are generative matching networks [3], variational homoencoders [50], and a class of sequential generative models that incorporate VAEs [99].

There also exists a group of generative models capable of few-shot learning that derive mostly from deep learning ideas and usually do not have a straightforward Bayesian interpretation, e.g. energy-based memory models [4], few-shot image generation using Reptile [20] and meta PixelCNN [96]. For this reason, we will not relate BRUNO to these methods.

To summarize, the majority of prior research builds around ideas of variational inference and VAEs in particular. Thus, BRUNO outlines a novel meta-learning approach that can compete with existing methods as we will show in Section 3.5, and allows for easy ways to extend it as demonstrated by conditional BRUNO in Chapter 4.

3.3 Method

In this section, we introduce BRUNO – an exchangeable architecture in which we combine Student-t processes [109] and Real NVP [24] resulting in a model with following properties:

1. predictive distribution $p(\boldsymbol{x}_{n+1}|\boldsymbol{x}_{1:n})$ is tractable to evaluate and can be sampled from with a linear complexity in the number of conditioning samples n
2. memory complexity is constant
3. training can be done via maximum likelihood in an RNN-like fashion of optimizing one-step-ahead predictions
4. we can uncover the posterior $p(\theta|\boldsymbol{x}_{1:n})$ and the likelihood $p(\boldsymbol{x}_i|\theta)$ of the corresponding Bayesian model, though they are unnecessary for computing $p(\boldsymbol{x}_{n+1}|\boldsymbol{x}_{1:n})$.

The appealing computational and memory complexity of BRUNO stems from restricting the covariances of TPs to be compound-symmetric as in Eq. 1.4, which enables us to derive recurrent updates for the predictive distribution. In the following, we provide these recurrent equations and lay out BRUNO's assumptions. The schematic of our model is given in Figure 3.1.

Assume we are given a finite subsequence $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ from an infinite exchangeable sequence, where every element is a D-dimensional vector: $\boldsymbol{x}_i = (x_i^1, \ldots x_i^D)$. We apply a Real NVP transformation to every $\boldsymbol{x}_i$, which results in an exchangeable sequence in the latent space: $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n$, where $\boldsymbol{z}_i \in \mathbb{R}^D$. The proof that the latter sequence is exchangeable is given in Appendix A.1.1.

We make the following assumptions about the latent variables:

A1: the latent variables $\{z^d\}_{d=1,\ldots,D}$ are independent, thus $p(\boldsymbol{z}) = \prod_{d=1}^{D} p(z^d)$

A2: for every dimension d, we assume the following: $(z_1^d, \ldots z_n^d) \sim MVT_n(\nu^d, \mu^d \mathbf{1}, \boldsymbol{K}^d)$, with parameters:

- degrees of freedom $\nu^d \in \mathbb{R}_+ \setminus [0, 2]$

- mean $\mu^d \mathbf{1}$ is a $1 \times n$ dimensional vector of ones multiplied by the scalar $\mu^d \in \mathbb{R}$
- $n \times n$ covariance matrix $\boldsymbol{K}^d$ with $\boldsymbol{K}^d_{ii} = v^d$ and $\boldsymbol{K}^d_{ij,i\neq j} = \rho^d$, where $0 \leq \rho^d < v^d$ to make sure that $\boldsymbol{K}^d$ is positive-definite.

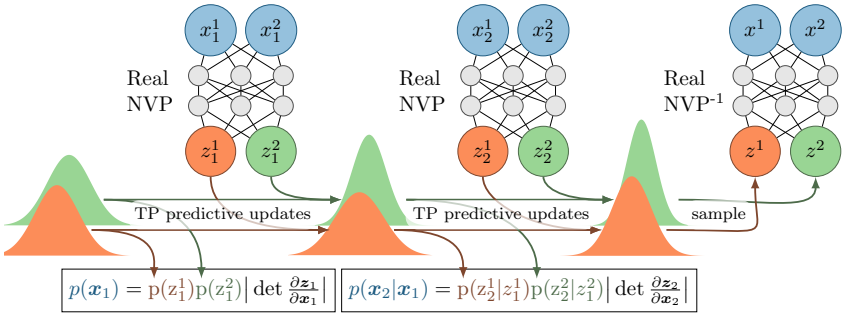


Figure 3.1 A schematic of the BRUNO model. It depicts how Bayesian thinking can lead to an RNN-like computational graph in which Real NVP is a bijective feature extractor and the recurrence is represented by Bayesian updates in a compound symmetry Student-t process. Here, the model is unrolled for two update steps, where data points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are mapped into latent vectors $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$. The TPs' parameters are updated after processing each input. For example, after observing $\boldsymbol{z}_1$, we update the parameters of the priors $p(z_1^1)$ and $p(z_1^2)$. At every step, we can evaluate the distributions $p(\boldsymbol{x}_1)$, $p(\boldsymbol{x}_2|\boldsymbol{x}_1), \ldots, p(\boldsymbol{x}_{n+1}|\boldsymbol{x}_{1:n})$ and sample from them by first sampling z^1 and z^2 and then applying the inverse Real NVP mapping.

The CS structure of the covariance matrix and having the same mean for every n guarantees that the sequence $z_1^d, z_2^d \ldots z_n^d$ is exchangeable. Because the covariance matrix is simple, we can derive recurrent updates for the parameters of $p(z_{n+1}^d|z_{1:n}^d)$. Using the recurrence is a lot more efficient compared to the closed-form expressions in Eq. 2.5 since during training we need to compute the predictive distribution at every step.

We start from a prior Student-t distribution for $p(z_1)$ with parameters $\mu_1 = \mu$, $v_1 = v$, $\nu_1 = \nu$, $\beta_1 = 0$. Here, we will drop the dimension index d to simplify the notation. A detailed derivation of the following results is given in Appendix A.1.2. To compute the

degrees of freedom, mean and variance of $p(z_{n+1}|z_{1:n})$ for every n, we have the recurrence relations:

$$\begin{aligned}\nu_{n+1} &= \nu_n + 1,\\ \mu_{n+1} &= (1-d_n)\mu_n + d_n z_n,\\ v_{n+1} &= (1-d_n)v_n + d_n(v-\rho),\end{aligned} \tag{3.2}$$

where $d_n = \frac{\rho}{v+\rho(n-1)}$. Note that if we wanted to use a GP intead of a TP, i.e. assuming that $(z_1^d, \dots z_n^d) \sim \mathcal{N}_n(\mu^d \mathbf{1}, \boldsymbol{K}^d)$, then recursions would simply use the latter two equations for μ_{n+1} and v_{n+1}. For TPs, however, we also need to compute β – a data-dependent term that scales the covariance matrix as in Eq. 2.5. To update β, we introduce recurrent expressions for the auxiliary variables:

$$\begin{aligned}\tilde{z}_i &= z_i - \mu,\\ a_n &= \frac{v+\rho(n-2)}{(v-\rho)(v+\rho(n-1))},\\ b_n &= \frac{-\rho}{(v-\rho)(v+\rho(n-1))},\\ \beta_{n+1} &= \beta_n + (a_n - b_n)\tilde{z}_n^2 + b_n(\sum_{i=1}^{n}\tilde{z}_i)^2 - b_{n-1}(\sum_{i=1}^{n-1}\tilde{z}_i)^2.\end{aligned}$$

From these equations, we see that computational complexity of making predictions in compound symmetry TPs or GPs scales linearly with the number of observations, i.e. $\mathcal{O}(n)$ instead of a general $\mathcal{O}(n^3)$ case where one needs to compute an inverse covariance matrix.

So far, we have constructed a Student-t process in the latent space $\mathcal{Z}$. By coupling it with a bijective Real NVP mapping, we get an exchangeable process in space $\mathcal{X}$. Although we do not have a simple form of the transitions in $\mathcal{X}$, we still can sample from this process and evaluate the predictive distribution via the change of variables formula.

Having an easy-to-evaluate autoregressive distribution $p(\boldsymbol{x}_{n+1}|\boldsymbol{x}_{1:n})$ allows us to use a training scheme that is common for RNNs, i.e. maximize the likelihood of the next element in the sequence at every step. Here, Figure 3.1 can aid in understanding some details of this training procedure. Our

objective function for a single sequence of fixed length N can be written as $\mathcal{L} = \sum_{n=0}^{N-1} \log p(\boldsymbol{x}_{n+1}|\boldsymbol{x}_{1:n})$, which is equivalent to maximizing the joint log-likelihood $\log p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$. While we do have a closed-form expression for the latter, we chose not to use it during training in order to minimize the difference between the implementation of training and testing phases. Note that at test time, dealing with the joint log-likelihood would be inconvenient or even impossible due to high memory costs when N gets large, which again motivates the use of a recurrent formulation.

During training, we update the weights of the Real NVP model and also learn the parameters of the prior Student-t distribution. For the latter, we have three trainable parameters per dimension: degrees of freedom ν^d, variance v^d and covariance ρ^d. The mean μ^d is fixed to 0 for every d and is not updated during training.

3.4 Alternative formulation

Until now, we only needed the predictive distribution $p(\boldsymbol{x}_{n+1}|\boldsymbol{x}_{1:n})$ for which we derived a closed-form recurrent formulation without any reference to the underlying Bayesian model. The latter would specify prior $p(\boldsymbol{\theta})$, likelihood $p(\boldsymbol{x}_i|\boldsymbol{\theta})$, and posterior $p(\boldsymbol{\theta}|\boldsymbol{x}_{1:n})$ distributions, which some applications might make use of. Unlike RNNs, BRUNO is only partially a black box since it allows to recover these distributions. Due to the bijective mapping $f : \mathcal{X} \mapsto \mathcal{Z}$, exchangeable processes in $\mathcal{X}$ and $\mathcal{Z}$ are governed by the same latent variables $\boldsymbol{\theta}$. It implies that:

$$
\begin{aligned}
p(\boldsymbol{\theta}|\boldsymbol{x}_{1:n}) &= p(\boldsymbol{\theta}|\boldsymbol{z}_{1:n}) \\
p(\boldsymbol{x}_i|\boldsymbol{\theta}) &= p(\boldsymbol{z}_i|\boldsymbol{\theta}) \left| \det \left(\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \right) \right|,
\end{aligned}
$$

thus formulating BRUNO as a latent variable model. In case of CS GPs or TPs, it is relatively straightforward to find $p(\boldsymbol{\theta}|\boldsymbol{z}_{1:n})$ and $p(\boldsymbol{z}_i|\boldsymbol{\theta})$, especially when we have independence assumptions and these distributions factorize, i.e. $p(\boldsymbol{\theta}|\boldsymbol{z}_{1:n}) = \prod_{d=1}^{D} p(\theta^d|z^d_{1:n})$. Firstly, we will deal with GPs as those will later serve as a basis for derivations in TPs.

As usual, we assume an exchangeable process where for any finite n, $z_1, \dots z_n \sim \mathcal{N}_n(\mu\mathbf{1}, \mathbf{\Sigma})$ with $\mathbf{\Sigma}_{ii} = v$, $\mathbf{\Sigma}_{ij,i\neq j} = \rho$. Using the implications of de Finetti's theorem [76], one can show that:

$$\begin{aligned} p(z_i|\theta) &= \mathcal{N}(\theta, v-\rho), \\ p(\theta) &= \mathcal{N}(\mu, \rho). \end{aligned} \tag{3.3}$$

We can arrive to this result in a more intuitive way by using a parameterization based on standard normal random variables. For example, a random variable $y \sim \mathcal{N}(\mu, \sigma^2)$ can be written as $y = \mu + \sigma\epsilon$ with $\epsilon \sim \mathcal{N}(0, 1)$. Given this form for n-dimensional normal vectors [75], we can write z_i as $z_i = \mu + \sqrt{\rho}\eta + \sqrt{v-\rho}\xi_i$, where $\eta \sim \mathcal{N}(0, 1)$ and $\xi_i \sim \mathcal{N}(0, 1)$. From this representation, the expressions for $p(z_i|\theta)$ and $p(\theta)$ in Eq. 3.3 follow immediately.

Models with Gaussian priors and likelihoods are well-studied objects in conjugate Bayesian analysis [86]. Thus, we can look up mean and variance parameters of the posterior $p(\theta|z_{1:n}) = \mathcal{N}(\mu_n, \sigma_n^2)$:

$$\begin{aligned} \mu_n &= \left(\frac{1}{\rho} + \frac{n}{v-\rho}\right)^{-1}\left(\frac{\mu}{\rho} + \frac{\sum_{i=1}^n z_i}{v-\rho}\right) \\ \sigma_n^2 &= \left(\frac{1}{\rho} + \frac{n}{v-\rho}\right)^{-1} \end{aligned} \tag{3.4}$$

In the same way, we can find the posterior predictive distribution: $p(z_{n+1}|z_{1:n}) = \mathcal{N}(\mu_n, \sigma_n^2 + v - \rho)$. However, we have already derived this result in Appendix A.1.2, where our starting point was the expression for the conditional distribution of multivariate normal distribution. This again proves the equivalence between the autoregressive and the Bayesian description of BRUNO.

Finding the posterior concludes our interpretation of BRUNO as a latent variable model. The same analysis holds for TP-based BRUNO models, though, the equations become more elaborate, and we present them in Appendix A.1.3.

3.5 Experiments

In this section, we will consider a few problems that fit naturally into the framework of modelling exchangeable data. We chose to work with sequences of images, so the results are easy to analyse, yet BRUNO does not make any image-specific assumptions, and our conclusions can generalise to other types of data. Specifically, for non-image data, one can use a general-purpose Real NVP coupling layer as proposed by [91]. In contrast to the original Real NVP model, which uses convolutional architecture for scaling and translation functions in Eq. 2.4, a general implementation has s and t composed from fully connected layers. We experimented with both convolutional and non-convolutional architectures, the details of which are given in Appendix A.2.1.

In our experiments, all BRUNO models are trained on thousands of image sequences, each of length 20. We form every sequence by uniformly sampling a class and then selecting 20 random images from that class. In this scheme, we imply that the exchangeability assumption holds within each sequence, and sequences themselves are i.i.d. A model trained to maximize the likelihood of images within each sequence thus implicitly learns to infer a class label that is global to the sequence. In what follows, we will see how this property can be used in a few tasks.

3.5.1 Image generation

We first consider a problem of generating samples conditionally on a set of images, which reduces to sampling from a predictive distribution. This is different from VAE-based approaches, where one needs to infer the posterior over some meaningful latent variable and then "decode" it.

To draw samples from $p(\boldsymbol{x}_{n+1}|\boldsymbol{x}_{1:n})$, we first sample $\boldsymbol{z} \sim p(\boldsymbol{z}_{n+1}|\boldsymbol{z}_{1:n})$ and then compute the inverse Real NVP mapping: $\boldsymbol{x} = f^{-1}(\boldsymbol{z})$. Since we assumed that dimensions of $\boldsymbol{z}$ are independent, we can sample each z^d from a univariate Student-t distribution. To do so, we modified Bailey's polar

t-distribution generation method [2] to be computationally efficient for GPU. Its algorithm is given in Appendix A.2.1.

In Figure 3.2, we show samples from the prior distribution $p(\boldsymbol{x}_1)$ and conditional samples from a predictive distribution $p(\boldsymbol{x}_{n+1}|\boldsymbol{x}_{1:n})$ at steps $n = 1, \ldots, 20$. Here, we used a convolutional Real NVP model as a part of BRUNO. The model was trained on Omniglot [72] same-class image sequences of length 20 and we used the train-test split and preprocessing as defined by [128]. Namely, we resized the images to 28×28 pixels and augmented the dataset with rotations by multiples of 90 degrees yielding 4,800 and 1,692 classes for training and testing respectively.

To better understand how BRUNO behaves, we test it on special types of input sequences that were not seen during training. In Figure 3.3, we give an example where the same image is used throughout the sequence. Here, we want to illustrate how samples variability depends on the variance of the inputs. From Figure 3.3, we see that in the case of a repeated input image, samples get more coherent as the number of conditioning inputs grows. It also shows that BRUNO does not merely generate samples according to the inferred class label. This property does not hold for the neural statistician model [28], discussed in Section 3.2. As mentioned earlier, the neural statistician computes the approximate posterior $q(\boldsymbol{c}|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ and then uses its mean to sample $\boldsymbol{x}$ from a conditional model $p(\boldsymbol{x}|\boldsymbol{c}_{mean})$. This scheme does not account for the variability in the inputs as a consequence of applying mean pooling over the features of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ when computing $q(\boldsymbol{c}|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$. Thus, when all x_i's are the same, it would still sample different instances from the class specified by x_i. Given the code provided by the authors of the neural statistician and following email exchange, we could not reproduce the results from their paper, so we refrained from making direct comparisons.

While Omniglot is limited to 20 images per class, we can experiment with distributions conditioned on more examples using MNIST [78], Fashion-MNIST [134] or CIFAR-10 [69]. In Figure 3.4, we show samples from the model trained on images of even MNIST digits. More samples from convolutional and non-convolutional architectures trained on these datasets are given in Appendix A.2.1.

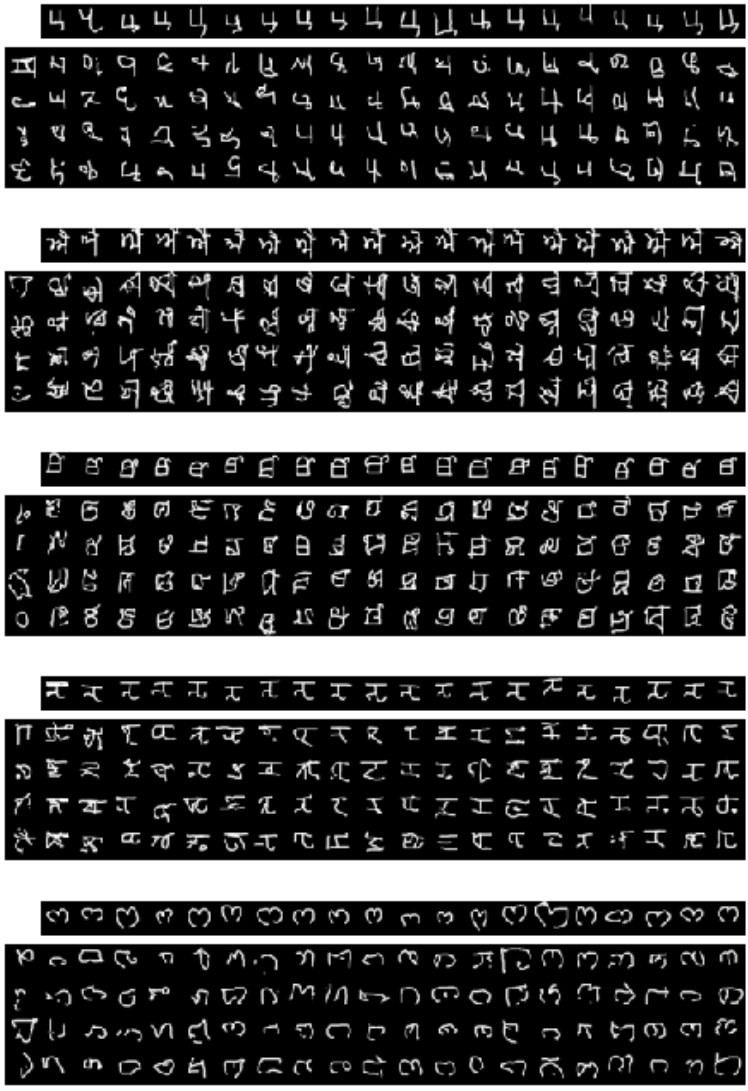


Figure 3.2 Samples generated conditionally on the sequence of the unseen Omniglot character class. An input sequence is shown in the top row and samples in the bottom 4 rows. Every column of the bottom subplot contains 4 samples from the predictive distribution conditioned on the input images up to and including that column. That is, the 1st column shows samples from the prior $p(\boldsymbol{x})$ when no input image is given; the 2nd column shows samples from $p(\boldsymbol{x}|\boldsymbol{x}_1)$ where $\boldsymbol{x}_1$ is the 1st input image in the top row and so on. More examples are given in Appendix A.2.1.

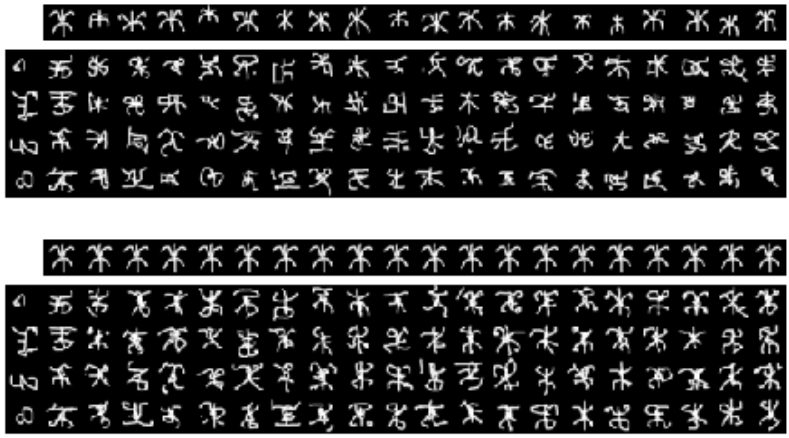


Figure 3.3 Samples generated conditionally on images from an unseen Omniglot character class. *Top:* input sequence of 20 images from one class. *Bottom:* the same image is used as an input at every step.

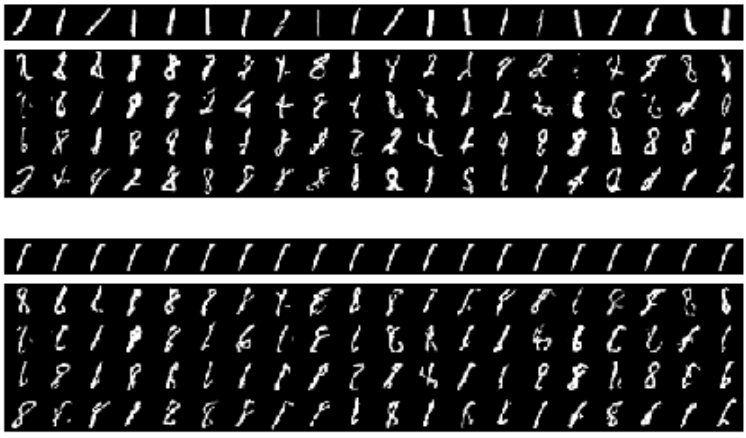


Figure 3.4 MNIST samples from $p(\boldsymbol{x}|\boldsymbol{x}_{1:n})$ for every $n = 480, \ldots, 500$. *Top*: input sequence (given in the top row of each subplot) is composed of random same-class test images. *Bottom*: same image is given as input at every step. The model was trained only on even digits, so it did not see digit '1' during training.

Finally, it is useful to analyse the covariance parameters of TPs after training. We observe that BRUNO learns to correlate only a small portion of dimensions. For the Omniglot model, Figure 3.5 plots the number of TPs where correlations ρ^d/v^d exceed a certain value on the x-axis. Here, only 28 dimensions out of 784 have a correlation higher than 0.1. Those processes would largely be responsible for model's adaptation – the ability to shift the predictive distribution based on a given set of inputs. One can also think about this from a perspective of task-specific parameters, as opposed to global parameters that are shared between tasks. Global parameters can include the weights of the Real NVP and parameters of those TPs, where correlations are negligible. The latter implies that predictive distribution remains close to prior. Therefore, it is reasonable to expect that dimensions z^d where $\rho^d/v^d \approx 0$ can only capture general image features as in a stand-alone Real NVP model. To sum up, the bijective mapping in BRUNO's construction restricts the dimensionality of the feature space $\mathcal{Z}$. However, given that $\mathcal{Z}$ is large enough, BRUNO is able to assign a portion of the dimensions in $\mathcal{Z}$, or rather TPs associated with those dimensions, to be responsible for the meta-learning aspect of the model.

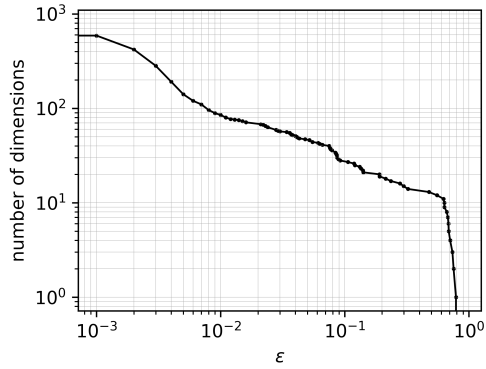


Figure 3.5 Number of dimensions where $\rho^d/v^d > \epsilon$ for the Omniglot model (plotted on a double logarithmic scale). Similar plots for CIFAR-10 and MNIST models are given in Appendix A.2.1.

3.5.2 Few-shot classification

Previously, we saw that BRUNO can generate images of the unseen classes even after being conditioned on a couple of examples. In this section, we will see how one can use its conditional probabilities not only for generation, but also for a few-shot classification.

We evaluate the few-shot learning accuracy of our Omniglot model from the previous section on the unseen Omniglot characters from the 1,692 testing classes following the n-shot and k-way classification setup proposed by [128]. For every test case, we randomly draw a test image $\boldsymbol{x}_{n+1}$ and a sequence of n images from the target class. At the same time, we draw n images for every of the $k-1$ random decoy classes. To classify an image $\boldsymbol{x}_{n+1}$, we compute $p(\boldsymbol{x}_{n+1}|\boldsymbol{x}_{1:n}^{C=i})$ for each class $i = 1 \dots k$ in the batch. An image is classified correctly when the conditional probability is highest for the target class compared to the decoy classes. This evaluation is performed 20 times for each of the test classes and the average classification accuracy is reported in Table 3.1.

For comparison, we considered three models from [128]: **(a)** k-nearest neighbours (k-NN), where matching is done on raw pixels (Pixels), **(b)** k-NN with matching on discriminative features from a state-of-the-art classifier (Baseline Classifier), and **(c)** Matching networks.

We observe that BRUNO model from Section 3.5.1 outperforms the baseline classifier, despite having been trained on relatively long sequences with a generative objective, i.e. maximizing the likelihood of the input images. Yet, it cannot compete with matching networks – a model tailored for a few-shot classification and trained in a discriminative way on short sequences such that its test-time protocol exactly matches the training time protocol. One can argue, however, that a comparison between models trained generatively and discriminatively is not fair. Generative modelling is a more general, harder problem to solve than discrimination, so a generatively trained model may waste a lot of statistical power on modelling aspects of the data which are irrelevant for the classification task. To verify our intuition, we fine-tuned BRUNO with a discriminative objective, i.e. maximizing the likelihood of correct labels in n-shot, k-way classification episodes formed from the

training examples of Omniglot. While we could sample a different n and k for every training episode like in matching networks, we found it sufficient to fix n and k during training. Namely, we chose the setting with $n = 1$ and $k = 20$. From Table 3.1, we see that this additional discriminative training makes BRUNO competitive with state-of-the-art models across all n-shot and k-way tasks.

Table 3.1 Classification accuracy for a few-shot learning task on the Omniglot dataset.

Model	**5-way**		**20-way**	
	1-shot	5-shot	1-shot	5-shot
PIXELS [128]	41.7%	63.2%	26.7%	42.6%
BASELINE CLASSIFIER [128]	80.0%	95.0%	69.5%	89.1%
MATCHING NETS [128]	98.1%	98.9%	93.8%	98.5%
BRUNO	86.3%	95.6%	69.2%	87.7%
BRUNO (discriminative fine-tuning)	97.1%	99.4%	91.3%	97.8%

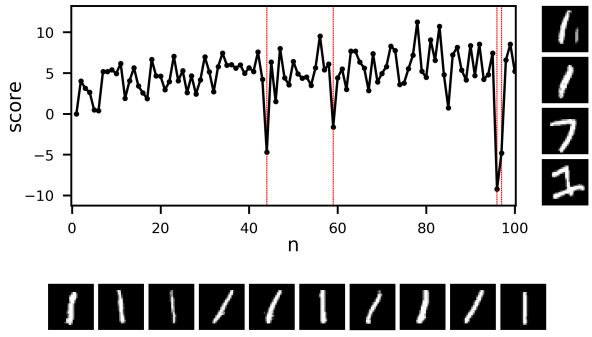
3.5.3 Set anomaly detection

Online anomaly detection for exchangeable data is another application where we can use BRUNO. This problem is closely related to the task of content-based image retrieval, where we need to rank an image $\boldsymbol{x}$ on how well it fits with the sequence $\boldsymbol{x}_{1:n}$ [49]. For the ranking, we use the probabilistic score proposed in Bayesian sets [38]:

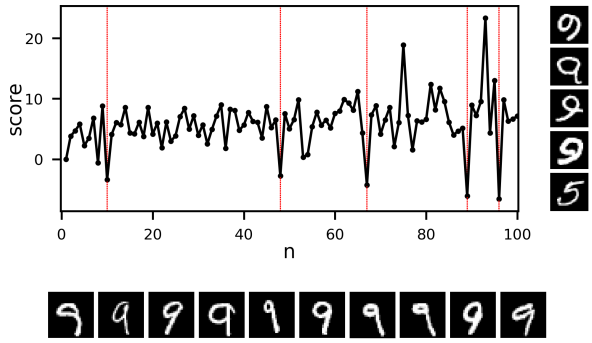
$$\text{score}(\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\boldsymbol{x}_{1:n})}{p(\boldsymbol{x})}.$$

This score compares the density of $\boldsymbol{x}$ under the predictive distribution to its density under the prior. Note that in BRUNO, when we care exclusively about comparing ratios of conditional densities of $\boldsymbol{x}$ under different sequences $\boldsymbol{x}_{1:n}$, we can compare densities in the latent space $\mathcal{Z}$ instead. This is because the Jacobian from the change of variables formula does not depend on the sequence we condition on.

For the following experiment, we trained a small convolutional version of BRUNO only on even MNIST digits (30,508 training images). In Figure 3.6, we give typical examples of how the score evolves as the model gets more data points and how it behaves in the presence of inputs that do not conform with the majority of the sequence. This preliminary experiment shows that our model can detect anomalies in a stream of incoming data.



(a) An input sequence of digit 1 images with a single image of a 7.



(b) An input sequence of digit 9 images with a single image of a 5.

Figure 3.6 Evolution of the score as the model sees more images from the input sequence whose typical representatives are plotted horizontally. Outliers that were identified based on the interquartile range are marked with vertical red lines and plotted on the right in the order from top to bottom. Note that the model was trained only on images of even digits.

3.5.4 GP-based models

When jointly optimizing Real NVP with TPs or GPs on top, we noticed that training of TP-based models can be easier compared to GP-based models. BRUNO with TPs appears more robust to anomalous training inputs and is less sensitive to the choice of hyperparameters. Under certain conditions, we were not able to obtain convergent training with GP-based models which was not the case when using TPs. We could pinpoint a few determining factors: **(a)** the use of weight normalisation [105] in the Real NVP layers, **(b)** an initialisation of the covariance parameters, and **(c)** presence of outliers in the training data. In Figure 3.7, we give examples of learning curves when BRUNO with GPs tends not to work well. Here, we use a convolutional Real NVP and train on Fashion MNIST. To simulate outliers, we occasionally feed sequences with some of the images completely white.

The results of this partial ablation study are insufficient to draw general conclusions, and we found many other settings where both versions of BRUNO diverge or they perform equally well in terms of test likelihoods, sample quality and few-shot classification results. Based on our experience, we can speculate that when extending BRUNO to new problems, it is reasonable to opt for GP-based models due to their simpler implementation. We also recommend using weight normalisation, small initial covariances, and small learning rates for a start. However, when finding a good set of hyperparameters is difficult, it is worth using TPs instead. Also, it remains true that a Student-t process is a strictly richer model class for the latent space with negligible additional computational costs.

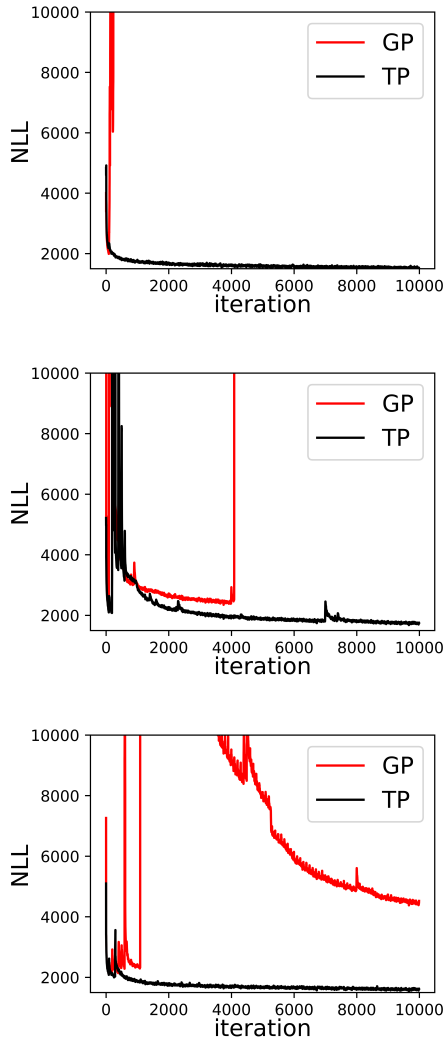


Figure 3.7 Negative log-likelihood of TP and GP-based BRUNO models on the training batches, smoothed using a moving average over 10 points. *Top:* without weight normalisation, initial covariances are sampled from $\mathcal{U}(0.1, 0.95)$ for every dimension. Here, the GP-based model diverged after a few hundred iterations. Adding weighnorm fixes this problem. *Middle:* with weight normalisation, covariances are initialised to 0.1, learning rate is 0.002 (twice the default). In this case, the learning rate is too high for both models, but the GP-based model suffers from it more. *Bottom:* with weight normalisation, covariances are initialised to 0.95.

3.6 Discussion and conclusion

In this chapter, we introduced BRUNO, a new technique combining deep learning and Student-t or Gaussian processes for modelling exchangeable data. With this architecture, we may carry out Bayesian inference, avoiding the need to compute posteriors and eliminating the high computational cost or approximation errors often associated with explicit Bayesian inference. At the same time, BRUNO can be viewed as a latent variable model with a closed-form posterior which can be computed if we wish to. This can serve both as an advantage and a limitation. On one hand, it is pleasing that BRUNO is a relatively simple model for the types of problems it undertakes. However, we did not achieve the goal of designing a black-box exchangeable RNN capable of implicit Bayesian inference with possibly infinite-dimensional latents, which would be an increasingly expressive model.

If we ignore the Bayesian aspect of BRUNO, one might question the practicality of having a bijective mapping between high-dimensional inputs and their features. This part of the model introduces the bulk of the computations, which could possibly be eliminated if not for the theoretical foundations. For example, we could use an autoencoder to obtain low-dimensional representations of high-dimensional inputs and train BRUNO on top of those embeddings. This would violate our claim to modelling an exchangeable process in the original input space, but might still make a useful model with the same capabilities as BRUNO.

In its present form, BRUNO shows promise for applications such as conditional image generation, few-shot concept learning, few-shot classification and online anomaly detection. The probabilistic construction makes the BRUNO approach particularly useful and versatile in transfer learning and multi-task situations. To demonstrate this, we showed that BRUNO trained in a generative way achieves good performance in a downstream few-shot classification task without any task-specific retraining. However, the performance can be significantly improved with discriminative fine-tuning.

While we are satisfied with BRUNO's performance on Omniglot, we cannot disregard a line of research that questions the ca-

pacity of Omniglot and other common benchmarks, e.g. mini-ImageNet [128], for testing few-shot capabilities of different algorithms [57]. This is mainly attributed to the similarity in class semantics between training and testing sets. For instance, in Omniglot, all of the images are characters, where even some train and test characters are visually similar. Meta-Datasets [123] is a recently introduced benchmark that reflects more realistic meta-learning conditions, where tasks can originate from different datasets, each with distinct data distribution. It would be insightful to test BRUNO on Meta-Datasets, though the type of its images requires larger Real NVP models, which are out of reach due to our hardware constraints. This was also the main reason why our experiments focused on Omniglot.

Training BRUNO is a form of meta-learning or learning-to-learn: it learns to perform Bayesian inference on various sets of data. Just as encoding translational invariance in convolutional neural networks seems to be the key to success in vision applications, we believe that the notion of exchangeability is equally central to data-efficient meta-learning. In this sense, exchangeable architectures like BRUNO can be seen as the most natural starting point for these applications.

As a consequence of exchangeability-by-design, BRUNO is endowed with a hidden state which integrates information about all inputs regardless of sequence length. This desired property for meta-learning is usually difficult to ensure in general RNNs as they do not automatically generalise to longer sequences than they were trained on and are sensitive to the ordering of inputs. Based on this observation, the most promising applications for BRUNO may fall in the many-shot meta-learning regime, where larger sets of data are available in each episode. Handling of large datasets or settings with constrained memory is also enabled by the recurrent formulation of BRUNO, where an observation $\boldsymbol{x}_i$ can be discarded after we used it to update the parameters of the predictive distribution. Such requirements naturally arise in privacy-preserving on-device machine learning, which is another potential future application area for BRUNO.

In the next chapter, we will present a conditional version of BRUNO which models a distribution $p(\boldsymbol{x}_{n+1}|\boldsymbol{h}_{n+1}, \boldsymbol{x}_{1:n}, \boldsymbol{h}_{1:n})$ with

$\boldsymbol{h}_i$ being labels or tags associated with observations $\boldsymbol{x}_i$. This conditional extension allows BRUNO to tackle problems where machine learning algorithms have only recently started to show significant progress as the result of building on top of advances in generative modelling and using current hardware capabilities.

4

Conditional BRUNO

4.1 Introduction

A natural extension to models introduced in the previous chapter is to define a BRUNO process with non-trivial labels, i.e when each $\boldsymbol{x}$ depends on some vector $\boldsymbol{h}$ of labels or tags. An illustrative task for this model would be to generate new viewpoints of a scene given a few images of that scene under different camera positions [31]. Here, the camera location corresponds to $\boldsymbol{h}$, and $\boldsymbol{x}(\boldsymbol{h})$ represents a picture as seen from $\boldsymbol{h}$. This is analogous to the common GP regression setting as discussed in Section 2.5, except that our model is best adapted to work with high-dimensional outputs, such as images, and it needs to be trained in a meta-learning fashion.

Following an accepted notation in the literature, we will write the predictive distribution in our model as $p(\boldsymbol{x}_{n+1}|\boldsymbol{h}_{n+1}, \boldsymbol{x}_{1:n}, \boldsymbol{h}_{1:n})$. However, we do not assume $\boldsymbol{h}$ to be a random variable. In particular, $p(\boldsymbol{x}|\boldsymbol{h})$ does not refer to the conditional distribution in a bivariate process of $\boldsymbol{x}$ and $\boldsymbol{h}$. Initially, this was a point of confusion for us, however, a brilliant clarification has been given in the work of McCullagh [82]. We will use his paper as a basis for our further explanation of exchangeability in regression models.

McCullagh [82] defines $\boldsymbol{h} : \mathcal{U} \mapsto \Omega$ as a function on the countably infinite set of statistical units $\mathcal{U}$ that takes values in an arbitrary set Ω. The set $\mathcal{U}$ is the index set on which the stochastic process

is defined, and Ω is known as the set of labels. For example, in the viewpoints generation task, we can define $\mathcal{U}$ as a set of natural numbers and establish a surjective mapping to a finite set of camera positions Ω. In practice, we only observe $\boldsymbol{x}$-values of the process on some finite subset $S = \{u_1, \dots, u_n\} \subset \mathcal{U}$. For any size $n = |S|$, a stochastic process needs to assign a distribution P_S over $\boldsymbol{x}(S) = \big(\boldsymbol{x}(\boldsymbol{h}_1), \dots, \boldsymbol{x}(\boldsymbol{h}_n)\big)$ in such a way that all finite-dimensional distributions are consistent. Namely, when $S' \subset S$ then $P_{S'}$ is a marginal of P_S under the deletion of coordinates that are not in S'.

If we have two finite sets S and S' of any equal size n, then according to McCullagh [82], a process is *regression exchangeable* if $P_S = P_{S'}$ when $\boldsymbol{h}$ takes the same values on S and S'.

To better undestand this definition, let us consider a Gaussian process $\boldsymbol{x}(\boldsymbol{h}) \sim \mathcal{GP}(\mu(\boldsymbol{h}), k(\boldsymbol{h}, \boldsymbol{h}'))$– an example of a regression exchangeable process. For $n = 2$, assume that $\boldsymbol{h}$ takes the values (a, b) on S and (b, a) on S'. Then the densities of P_S and $P_{S'}$ are respectively:

$$\mathcal{N}\Big(\begin{bmatrix} \mu(a) \\ \mu(b) \end{bmatrix}, \begin{bmatrix} k(a,a) & k(a,b) \\ k(b,a) & k(b,b) \end{bmatrix} \Big) \text{ and } \mathcal{N}\Big(\begin{bmatrix} \mu(b) \\ \mu(a) \end{bmatrix}, \begin{bmatrix} k(b,b) & k(b,a) \\ k(a,b) & k(a,a) \end{bmatrix} \Big).$$

Since the first Gaussian corresponds to $p(\boldsymbol{x}(a), \boldsymbol{x}(b))$, and the second one – to $p(\boldsymbol{x}(b), \boldsymbol{x}(a))$, we conclude that P_S and $P_{S'}$ define the same distribution.

Translating McCullagh's [82] definition of regression exchangeability into the form of Eq. 1.3, that focuses on sequences of random variables rather than distributions, gives us:

$$p(\boldsymbol{x}_1, \dots, \boldsymbol{x}_n | \boldsymbol{h}_1, \dots, \boldsymbol{h}_n) = p\left(\boldsymbol{x}_{\pi(1)}, \dots, \boldsymbol{x}_{\pi(n)} | \boldsymbol{h}_{\pi(1)}, \dots, \boldsymbol{h}_{\pi(n)}\right), \quad (4.1)$$

where we denoted $\boldsymbol{x}_i = \boldsymbol{x}(\boldsymbol{h}_i)$.

Similarly, the consistency property can be written as:

$$p(\boldsymbol{x}_{1:m} | \boldsymbol{h}_{1:m}) = \int p\left(\boldsymbol{x}_{1:n} | \boldsymbol{h}_{1:n}\right) d\boldsymbol{x}_{m+1:n} \text{ for } 1 \leq m < n. \quad (4.2)$$

As discussed in Section 1.2, in order to make a connection between exchangeability and Bayesian modelling, we need de Finetti's theorem, which we previously formulated using Eq. 1.5 and 1.7. For

regression exchangeable processses, we can write the following analogous equations:

$$p(\boldsymbol{x}_{1:n}|\boldsymbol{h}_{1:n}) = \int p(\boldsymbol{\theta}) \prod_{i=1}^{n} p(\boldsymbol{x}_i|\boldsymbol{h}_i, \boldsymbol{\theta}) d\boldsymbol{\theta} \tag{4.3}$$

$$p(\boldsymbol{x}_{n+1}|\boldsymbol{h}_{n+1}, \boldsymbol{x}_{1:n}, \boldsymbol{h}_{1:n}) = \int p(\boldsymbol{\theta}|\boldsymbol{x}_{1:n}, \boldsymbol{h}_{1:n}) p(\boldsymbol{x}_{n+1}|\boldsymbol{h}_{n+1}, \boldsymbol{\theta}) d\boldsymbol{\theta} \tag{4.4}$$

It is interesting to note, that according to Garnelo et al. [37] and Yang et al. [135], the conditions in Eq. 4.1 and 4.2 that make de Finetti's theorem applicable, can be gathered from the Kolmogorov extension theorem [90]. Also, these papers refer to the above equations as the conditional version of de Finetti's theorem, which is the terminology we will sometimes use as well.

Following the reasoning behind the original BRUNO, we would like to design a model where we directly construct a stochastic process that satisfies Eq. 4.1 and Eq. 4.2, and whose predictive distribution $p(\boldsymbol{x}_{n+1}|\boldsymbol{h}_{n+1}, \boldsymbol{x}_{1:n}, \boldsymbol{h}_{1:n})$ is easy to evaluate and to sample from. In Section 4.3, we show how this can be done by slightly modifying the architecture of BRUNO. Despite our earlier remarks on the use of conditional distributions in reference to $p(\boldsymbol{x}|\boldsymbol{h})$, we still name this model "conditional" in order to make a distinction from the previously introduced model. Thus, in this chapter, we will present conditional BRUNO, or C-BRUNO for short.

4.2 Related work

In the related work section of BRUNO 3.2, we identified a group of methods that build upon the ideas of VAEs. The same group exists in the conditional case. One example that became prominent in recent years is the Neural Process (NP) [37]. NPs rely on the conditional version of de Finetti's theorem and define $p(\boldsymbol{\theta})$ and $p(\boldsymbol{x}_i|\boldsymbol{h}_i, \boldsymbol{\theta})$ such that one can use variational techniques to approximate the integrals in Eq. 4.3 and 4.4. Given that the main goal of NPs is to make predictions, it is more relevant to compute the lower bound for $\log p(\boldsymbol{x}_{n+1}|\boldsymbol{x}_{1:n}, \boldsymbol{h}_{1:n+1})$ in Eq. 4.4. Alternatively, NPs derive the lower bound for the predictive distribution $\log p(\boldsymbol{x}_{n+1:n+m}|\boldsymbol{x}_{1:n}, \boldsymbol{h}_{1:n+m})$ of m query points at once:

$$\log p(\boldsymbol{x}_{n+1:n+m}|\boldsymbol{x}_{1:n},\boldsymbol{h}_{1:n+m})$$
$$\geq \mathbb{E}_{\boldsymbol{\theta}\sim q_{\boldsymbol{\psi}}(\boldsymbol{\theta}|\boldsymbol{x}_{1:n+m},\boldsymbol{h}_{1:n+m})}\left[\sum_{i=n+1}^{n+m}\log p(\boldsymbol{x}_i|\boldsymbol{h}_i,\boldsymbol{\theta})\right]$$
$$- KL\left[q_{\boldsymbol{\psi}}(\boldsymbol{\theta}|\boldsymbol{x}_{1:n+m},\boldsymbol{h}_{1:n+m})||q_{\boldsymbol{\psi}}(\boldsymbol{\theta}|\boldsymbol{x}_{1:n},\boldsymbol{h}_{1:n})\right].$$

Here, $q_{\boldsymbol{\psi}}(\boldsymbol{\theta}|\boldsymbol{x}_{1:n},\boldsymbol{h}_{1:n})$ and $q_{\boldsymbol{\psi}}(\boldsymbol{\theta}|\boldsymbol{x}_{1:n+m},\boldsymbol{h}_{1:n+m})$ are Gaussian variational posterior distributions parameterized by a permutation-invariant neural network. To achieve the invariance, NPs use mean pooling to aggregate representations over n or $n+m$ pairs of inputs $(\boldsymbol{x}_i,\boldsymbol{h}_i)$ as shown in Figure 4.1. Note that the use of query points in the variational posterior introduces a mismatch between the training and testing regimes of NPs. In the latter case, since we do not know the query points up front, the variational posterior is only conditioned on n points from the context set. Figure 4.1 illustrates how NPs are used during test time. The effectiveness of NPs was demonstrated on problems with low-dimensional observations such as 1D regression, where NPs assume that $p(x|h,\boldsymbol{\theta})=\mathcal{N}(\mu,\sigma^2)$ with $(\mu,\sigma)=\text{MLP}_{\boldsymbol{\phi}}(h,\boldsymbol{\theta})$. If one wishes to model complex high-dimensional inputs, the information in $\boldsymbol{\theta}$ can be used to condition a deep generative model. For example, Generative Query Networks (GQN) [31] build upon NPs principles, but use powerful recurrent density models for $p(\boldsymbol{x}|\boldsymbol{h},\boldsymbol{\theta})$, thus achieving remarkable results in tasks like 3D scene reconstruction.

NPs and several other important approaches in meta-learning, such as the model-agnostic meta-learning (MAML) [34], prototypical networks [111], hyper-networks [43], can be viewed under a framework of meta-learning approximate probabilistic inference for prediction (ML-PIP) [40]. As usual, the aim here is to approximate the predictive posterior $p(\boldsymbol{x}_{n+1}|\boldsymbol{h}_{n+1},\boldsymbol{x}_{1:n},\boldsymbol{h}_{1:n})$. To do so, ML-PIP uses a distribution $q_\psi(\boldsymbol{x}_{n+1}|\boldsymbol{h}_{n+1},\boldsymbol{x}_{1:n},\boldsymbol{h}_{1:n})$, which is constructed based on the amortized approximate posterior $q_\psi(\boldsymbol{\theta}|\boldsymbol{x}_{1:n},\boldsymbol{h}_{1:n})$ over the task-specific latent parameters $\boldsymbol{\theta}$:

$$q_\psi(\boldsymbol{x}_{n+1}|\boldsymbol{h}_{n+1},\boldsymbol{x}_{1:n},\boldsymbol{h}_{1:n})=\int p(\boldsymbol{x}_{n+1}|\boldsymbol{h}_{n+1},\boldsymbol{\theta})q_\psi(\boldsymbol{\theta}|\boldsymbol{x}_{1:n},\boldsymbol{h}_{1:n})d\boldsymbol{\theta},$$

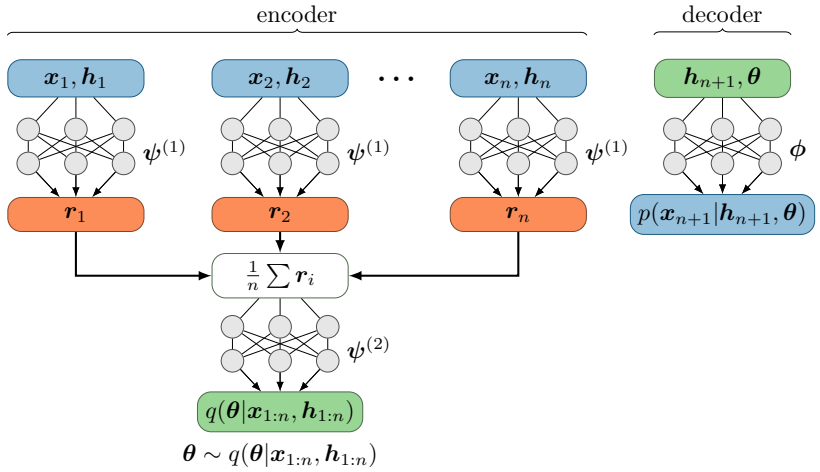


Figure 4.1 A schematic of Neural Processes as used at test time. The encoder maps every tuple $(\boldsymbol{x}_i, \boldsymbol{h}_i)$ in the context set into a representation $\boldsymbol{r}_i$ and aggregates the obtained $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_n$ by taking their average. The mean representation is then mapped to the parameters of the variational posterior $q_{\boldsymbol{\psi}}(\boldsymbol{\theta}|\boldsymbol{x}_{1:n}, \boldsymbol{h}_{1:n})$. Here, $\boldsymbol{\psi}$ denotes all weights and biases in the encoder network, i.e. $\boldsymbol{\psi} = \{\boldsymbol{\psi}^{(1)}, \boldsymbol{\psi}^{(2)}\}$. A sample $\boldsymbol{\theta} \sim q_{\boldsymbol{\psi}}(\boldsymbol{\theta}|\boldsymbol{x}_{1:n}, \boldsymbol{h}_{1:n})$ is fed to the decoder together with some label. From the distribution parameterized by the decoder we can sample new instances of $\boldsymbol{x}$ conditionally on the label $\boldsymbol{h}$ and the value of $\boldsymbol{\theta}$.

where q_ψ is an order-invariant neural network with parameters ψ, and the integral can be approximated using Monte Carlo sampling.

Versa [40] is a particular instance of the ML-PIP framework which is suitable for view reconstruction. In Section 4.4, we will compare Versa with C-BRUNO on this task.

Outside of the meta-learning context, it is possible to draw parallels between C-BRUNO and the idea of warped GPs [112]. In a regression problem, their approach is to map a 1-dimensional target x into a latent variable z using a learnable invertible transformation f such that the transformed data z can be well-modelled by a GP, i.e. $z(\boldsymbol{h}) \sim \mathcal{GP}(\mu(\boldsymbol{h}), k(\boldsymbol{h}, \boldsymbol{h}'))$. This leads to a non-Gaussian process in the original space with more complex predictive dis-

tributions, which unlike Gaussians, could be multimodal and asymmetric. It is possible, however, to estimate the shape of the predictive distribution by computing the median and percentiles. The idea of constructing a non-Gaussian process by the means of invertible mappings is what connects our model to warped GPs.

4.3 Method

Construction of conditional BRUNO requires the same mathematical tools as the unconditional model: compound symmetry Gaussian or Student-t processes, and an expressive bijective neural network, for which we chose Real NVP. To model distributions $p(\boldsymbol{x}_{n+1}|\boldsymbol{h}_{n+1}, \boldsymbol{x}_{1:n}, \boldsymbol{h}_{1:n})$, the only modification we need is to make Real NVP depend on the labels $\boldsymbol{h}_i$. In this case, it implements a mapping $\boldsymbol{z}_i = f(\boldsymbol{x}_i, \boldsymbol{h}_i)$, where every observation $\boldsymbol{x}_i$ is mapped to features $\boldsymbol{z}_i$ conditionally on $\boldsymbol{h}_i$. If as previously, we assume a fixed distribution for features in $\mathcal{Z}$, we would still model a conditional density of inputs in $\mathcal{X}$ since the conditioning on $\boldsymbol{h}$ is baked into the Jacobian, i.e.:

$$p(\boldsymbol{x}|\boldsymbol{h}) = p(\boldsymbol{z}) \left| \det \left(\frac{\partial f(\boldsymbol{x}, \boldsymbol{h})}{\partial \boldsymbol{x}} \right) \right|.$$

To model conditional distributions $p(\boldsymbol{x}|\boldsymbol{h})$ with Real NVP, we need to ensure that scaling and translation networks s and t in Eq. 2.4 depend on $\boldsymbol{h}$. One simple way to do it is to concatenate the embeddings of $\boldsymbol{h}$ to the inputs of dense and convolutional layers within the s and t networks. Other works confirmed the effectiveness of this approach, though exact implementation details of this idea can vary [44, 91].

Having built a conditional bijective mapping, we still need to specify the stochastic process in the feature space $\mathcal{Z}$ to fully define our model. In fact, C-BRUNO has the same set of assumptions as its unconditional counterpart, but for completeness, we list them below.

A1: dimensions $\{z^d\}_{d=1,\dots,D}$ are independent, so $p(\boldsymbol{z}) = \prod_{d=1}^{D} p(z^d)$
A2: for each dimension d, we assume that $(z_1^d, \dots z_n^d) \sim \mathcal{N}_n(\boldsymbol{0}, \boldsymbol{\Sigma}^d)$,

where $\mathbf{\Sigma}^d$ is a $n \times n$ covariance matrix with $\mathbf{\Sigma}^d_{ii} = v^d$ and $\mathbf{\Sigma}^d_{ij,i\neq j} = \rho^d$ with $0 \leq \rho^d < v^d$.

Here we opted for using GPs instead of TPs as we noticed that conditional image density models are easier to train compared to unconditional models in Section 3.5.1, so the extra training stability offered by TPs was redundant. A schematic of the full C-BRUNO model and its workings is given in Figure 4.2.

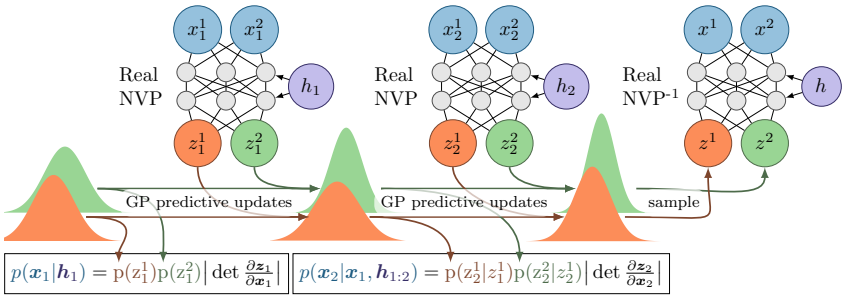


Figure 4.2 A schematic of C-BRUNO unrolled for two update steps and a sampling step. This illustrates how our model is able to update, evaluate and sample from the predictive distribution, which is now dependent on the vectors $\boldsymbol{h}$ representing some labels or tags.

4.4 Experiments

We consider the task of few-shot image reconstruction, where the model is required to infer how an object looks from various angles based on a small set of observed views [40]. This problem can be framed as generating samples from a predictive conditional distribution $p(\boldsymbol{x}_{n+1}|\boldsymbol{h}_{n+1}, \boldsymbol{x}_{1:n}, \boldsymbol{h}_{1:n})$, where $\boldsymbol{h}_{n+1}$ is a desired angle and $\boldsymbol{x}_{1:n}$ is a set of observed views associated with angles $\boldsymbol{h}_{1:n}$. We use a set of 12 object categories from ShapeNetCore v2 [15] as selected by Gordon et al. [40], and train C-BRUNO to predict different views from a single shot. Namely, given a random view $\boldsymbol{x}_1$ and its angle $\boldsymbol{h}_1$, the goal is to predict N views of the same object under angles $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N$. Thus, the training objective is to maximize the likelihood of ground-truth images under the

model distribution, i.e. $\mathcal{L} = \sum_{n=1}^{N} \log p(\boldsymbol{x}_n | \boldsymbol{h}_n, \boldsymbol{x}_1, \boldsymbol{h}_1)$. This loss is optimized with respect to the Real NVP parameters and variance-covariance parameters of the GPs. We train C-BRUNO in a batch-mode on all 12 object classes at once and use the same train-test split as Versa so that the two models are comparable.

In Figure 4.3, we show samples from C-BRUNO when the model is given a single viewpoint of an object that was not seen during training. In the majority of cases, our samples have correct orientation and are visually sharper compared to samples from Versa. The difference between the two models increases with more uncertainty in the object's appearance or when the object is less similar to the training examples. In this case, C-BRUNO generates samples with higher variance and more inaccuracies while Versa samples become more blurry. In Figure 4.4, this is illustrated for the airplane object. When a single shot is given, from which we cannot infer the wing configuration, C-BRUNO samples more diverse airplanes compared to when we condition on multiple distinctive viewpoints. With more airplane shots, the quality of Versa samples increases as well. However, as we can see from the car example, this does not always hold, which might be an indication that Versa requires training with multiple input shots in order to perform well in these testing conditions. C-BRUNO, on the other hand, is more agnostic to the length of sequences it is trained or tested on. More examples of generated images are given in Appendix A.2.2.

C-BRUNO and Versa focus on different aspects of image modelling, which complicates the quantitative comparison between the two models in terms of image quality metrics. Namely, Versa generates blurry images with negligible variability between samples drawn from the same predictive distribution. Such images would be favoured, for instance, by the mean squared error. C-BRUNO, on the other hand, generates sharp samples with a number of inaccuracies. Also, if we draw multiple samples from the predictive distribution of C-BRUNO, the resulting set of images will be more diverse in comparison to those generated by Versa. Arguably, it would be best to compare the two models with respect to other applications, but when our goal is to synthesise images, a subjective evaluation is generally viewed as appropriate [118].

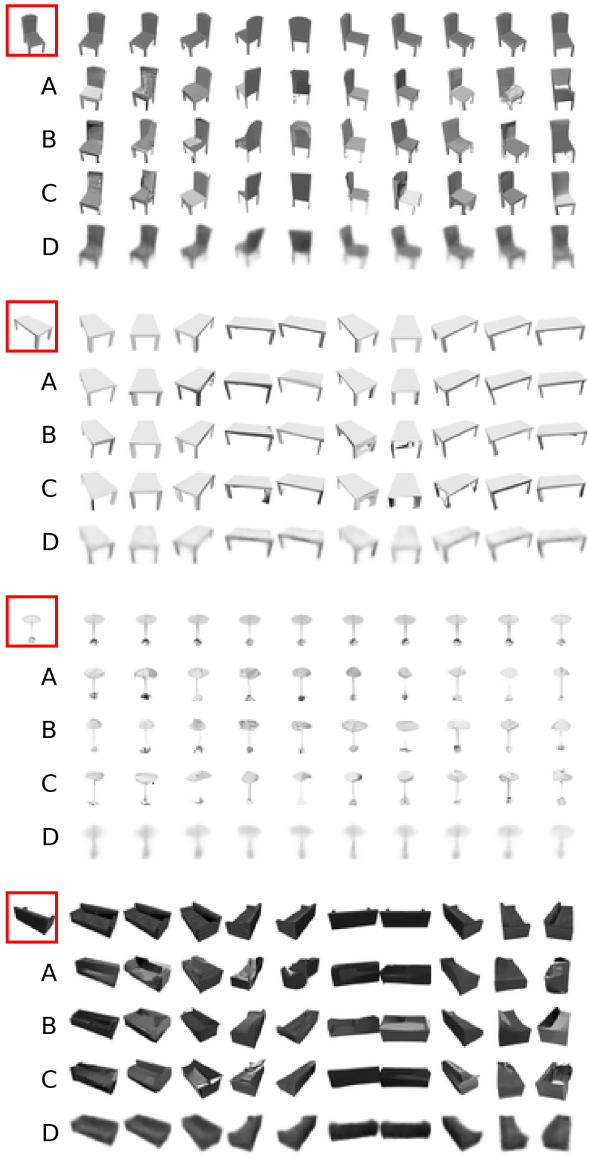


Figure 4.3 One-shot C-BRUNO samples in rows A-C and Versa samples in row D for the unseen test objects. Here, we condition on a single view $(\boldsymbol{x}_1, \boldsymbol{h}_1)$ marked in red. The top row of each plot contains ground truth images, whereas the three rows A to C are the C-BRUNO samples from $p(\boldsymbol{x}|\boldsymbol{h}, \boldsymbol{x}_1, \boldsymbol{h}_1)$ conditioned on a different angle $\boldsymbol{h}$ in each column.

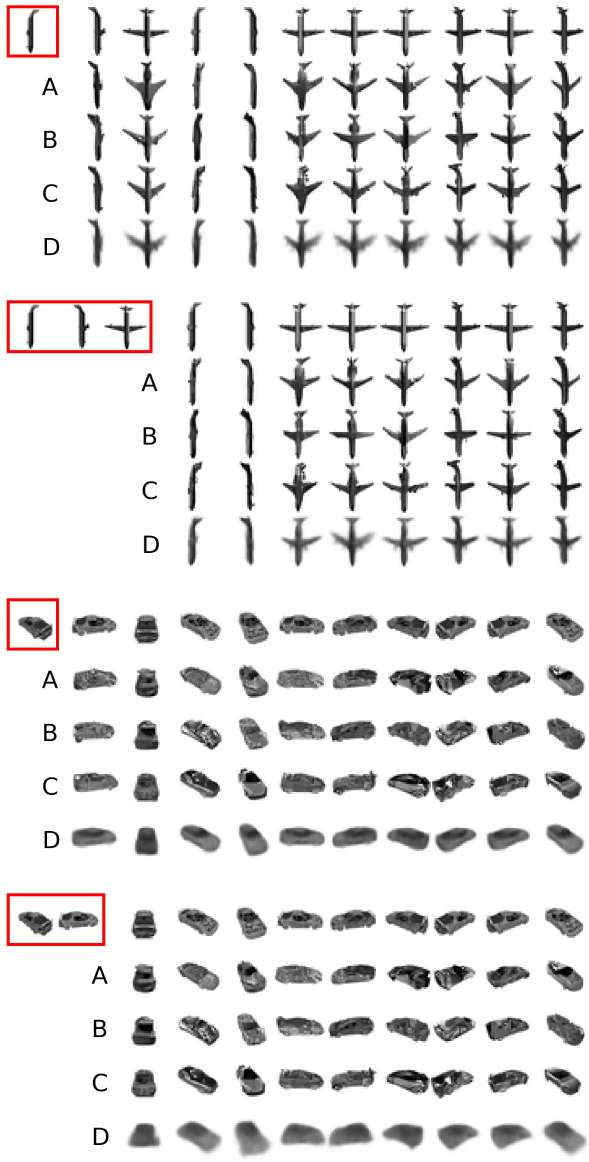


Figure 4.4 C-BRUNO samples in rows A-C and Versa samples in row D for the unseen test objects. Here, we condition on views $(\boldsymbol{x}_{1:n}, \boldsymbol{h}_{1:n})$ marked in red. The top row of each plot contains ground truth images, whereas the three rows A to C are the C-BRUNO samples from $p(\boldsymbol{x}|\boldsymbol{h}, \boldsymbol{x}_{1:n}, \boldsymbol{h}_{1:n})$ conditioned on a different angle $\boldsymbol{h}$ in each column.

4.5 The bottleneck problem

As we mentioned in the previous chapter, hardware constraints may limit our ability to train bigger models required for datasets with large-size colour images. However, BRUNO and C-BRUNO encounter more fundamental issues in low-dimensional data settings. Experimentally this was observed when trying to model 3D point cloud data [5]. To explain this conceptually, let us contrast C-BRUNO to NPs. Unlike C-BRUNO, NPs do not use bijections, so they can map one or low-dimensional inputs to high-dimensional latent spaces. In particular, one can choose the size of the latent variable $\boldsymbol{\theta}$ whose variational posterior $q(\boldsymbol{\theta}|\boldsymbol{x}_{1:n}, \boldsymbol{h}_{1:n})$ is parameterized by the encoder network. A sufficiently large latent space allows NPs to approximate, for instance, the distribution of functions drawn from a GP with an exponentiated quadratic kernel. In C-BRUNO, because of the bijection, one-dimensional inputs result in one-dimensional latents, which makes for a powerless model. This bottleneck problem of normalizing flows is a subject of ongoing research. The few existing works on this topic propose to construct invertible models on the augmented input space, where data is coupled with random noise or constant-valued dimensions [16, 27, 56]. We will follow a similar approach and base our explanation on the existing derivations and terminology introduced by Huang et al. [56].

In practice, the idea is simple: we concatenate an independent noise variable $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ to each input $\boldsymbol{x}$, and model their joint distribution $p(\boldsymbol{x}, \boldsymbol{\epsilon})$. This amounts to a so-called augmented maximum likelihood estimation (AMLE). Maximizing $\mathbb{E}_{\boldsymbol{x},\boldsymbol{\epsilon}}[\log p(\boldsymbol{x}, \boldsymbol{\epsilon})]$ is equivalent to maximizing the lower bound on $\mathbb{E}_{\boldsymbol{x}}[\log p(\boldsymbol{x})]$. To show this for a single $\boldsymbol{x}$, we can use the following reasoning [56]:

$$\begin{aligned}\log p(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{\epsilon}\sim q(\boldsymbol{\epsilon})}[\log p(\boldsymbol{x}, \boldsymbol{\epsilon})] &= -\mathbb{E}_{\boldsymbol{\epsilon}\sim q(\boldsymbol{\epsilon})}[\log p(\boldsymbol{\epsilon}|\boldsymbol{x})] \\ &= \mathrm{KL}[q(\boldsymbol{\epsilon})||p(\boldsymbol{\epsilon}|\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{\epsilon}\sim q(\boldsymbol{\epsilon})}[\log q(\boldsymbol{\epsilon})].\end{aligned}$$

Here, the term $-\mathbb{E}_{\boldsymbol{\epsilon}\sim q(\boldsymbol{\epsilon})}[\log q(\boldsymbol{\epsilon})]$ is the entropy $(\boldsymbol{\epsilon})$ of our noise distribution, which is constant. Moving it to the other side of the equation and noting that KL-divergence is non-negative gives:

$$\log p(\boldsymbol{x}) - \big(\mathbb{E}_{\boldsymbol{\epsilon}\sim q(\boldsymbol{\epsilon})}[\log p(\boldsymbol{x}, \boldsymbol{\epsilon})] + \mathcal{H}(\boldsymbol{\epsilon})\big) = \mathrm{KL}[q(\boldsymbol{\epsilon})||p(\boldsymbol{\epsilon}|\boldsymbol{x})] \geq 0.$$

Thus, the lower bound on the marginal log-likelihood equals to $\mathbb{E}_{\boldsymbol{\epsilon}\sim q(\boldsymbol{\epsilon})}[\log p(\boldsymbol{x}, \boldsymbol{\epsilon})] + \mathcal{H}(\boldsymbol{\epsilon})$. Huang et al. [56] showed a correspondence between the variational gap $\mathrm{KL}[q(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})]$ as in VAEs and $\mathrm{KL}[q(\boldsymbol{\epsilon})||p(\boldsymbol{\epsilon}|\boldsymbol{x})]$ in the augmented Real NVP with two coupling layers: one serving as an encoder that transforms $\boldsymbol{\epsilon}$ depending on $\boldsymbol{x}$ into $\boldsymbol{z}$, while the other decodes $\boldsymbol{z}$. Having this augmentation gap and not being able to directly compute the exact likelihood $p(\boldsymbol{x})$ would compromise the theoretical properties of BRUNO, which heavily rely on a bijective mapping between $\mathcal{X}$ and $\mathcal{Z}$. While we think that using augmented flows as a part of BRUNO is not a satisfactory solution, we do not have a better way of circumventing the bottleneck problem. Nevertheless, as the experiments will show, for some simple classes of distributions, our models remain functional.

The augmented MLE objective does not change the training procedure of BRUNO models. From this perspective, it is neater than in case of VAEs as we do not have to deal with a two-part loss function or the reparameterization trick. For evaluation purposes, whenever we need to compute the marginal likelihood, it can be done using the importance sampling estimator [14]:

$$\log p(\boldsymbol{x}) \approx \log \frac{1}{K} \sum_{k=1}^{K} \frac{p(\boldsymbol{x}, \boldsymbol{\epsilon}_k)}{q(\boldsymbol{\epsilon}_k)},$$

where we draw i.i.d. samples of $\boldsymbol{\epsilon}$ from the noise distribution.

When dealing with low-dimensional inputs, we will also replace Real NVP with MAF [91] since the latter is more flexible and, in our experience, more suitable for non-image data.

Experiments

To illustrate how C-BRUNO behaves with one-dimensional inputs, we will look at a few toy problems. Firstly, let us consider a class of functions $x = a \sin(2h)$ with $a \in [0, 2]$ and $h \in [-2, 2]$. Being able to model 1D predictive distributions $p(x_{n+1}|h_{1:n+1}, x_{1:n})$ with MAF requires us to augment the inputs with at least one noise

variable $\epsilon \sim \mathcal{N}(0, 1)$, which results in a two-dimensional latent space.

During training, we construct each sequence by sampling $a \sim U(0, 2)$ and randomly choosing a number of context points k, so that the AMLE loss amounts to $\mathcal{L} = \sum_{n=1}^{N} p(x_n, \epsilon_n | x_{1:k}, h_{1:k}, \epsilon_{1:k})$. At test time, we sample from the predictive distribution and discard the noise dimensions. Figure 4.5 illustrates the results of C-BRUNO, which are interesting to compare to those of NPs in Figure 4.6, especially for the curves outside of the training distribution. Details describing the architectures of C-BRUNO and NPs are given in Appendix A.2.2.

In addition to changing the amplitude of the sine waves, we can also experiment with the translation. For this problem, we consider a class of functions $x = 0.5 \sin(h) + t$, with $t \in [-2, 2]$. After training, both C-BRUNO and NPs can perfectly predict the underlying sine curve if it comes from the training distribution. However, for the values of t outside the $[-2, 2]$ range, C-BRUNO performs better than NPs. Examples of this behaviour are illustrated in Figure 4.7.

While C-BRUNO can succeed in these simple problems, it experiences difficulties when learning to approximate the distribution of functions drawn from a GP with an exponentiated quadratic kernel $k(x, x') = \sigma^2 \exp\left(-\frac{(x-x')^2}{2l^2}\right)$, where we used $l = 0.6$ and $\sigma = 1$. From Figure 4.8, it is clear that this problem is a better match for NPs.

Even though our experiments illustrate that problems with low-dimensional inputs may not the best niche for our model, in the next chapter, we will again use C-BRUNO in 1D settings. There, we will see how C-BRUNO helps to achieve competitive performance in the downstream reinforcement learning tasks.

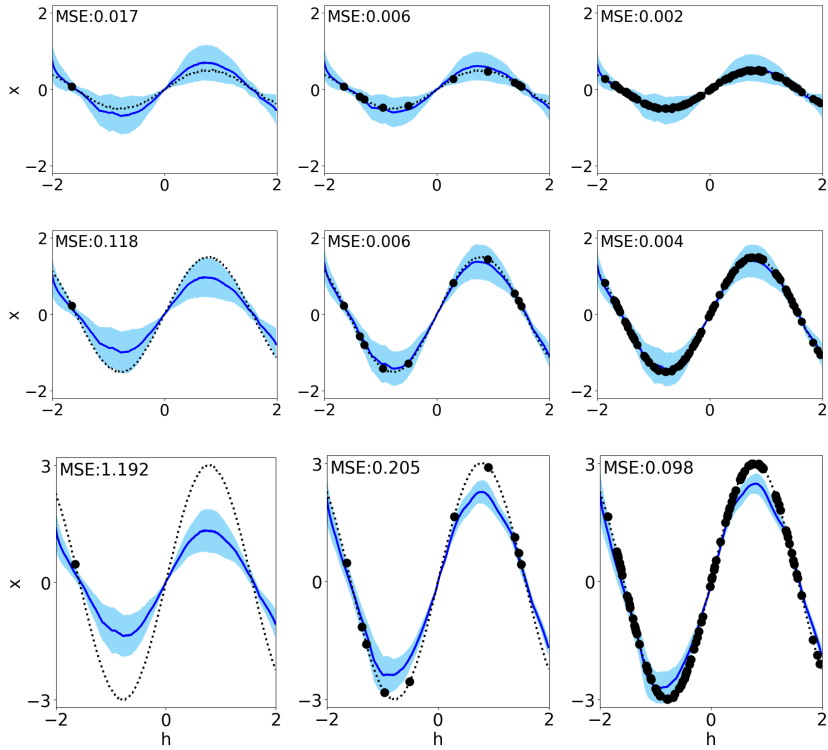


Figure 4.5 1D regression results of C-BRUNO with a 2-D latent space on the scaled sine problem. In blue, we plot the mean and two standard deviations around the mean of samples from a predictive distribution, where we condition on the context points in black. The context length is 1, 10 and 100 for left, middle and right columns respectively. Each row corresponds to a sine wave $x = a\sin(2h)$ with $a = \{0.5, 1.5, 3\}$. The underlying ground truth curve is given in black. For each case, we compute the mean squared error (MSE) between the ground truth and our mean prediction on 400 evenly spaced points between $[-2, 2]$. In the bottom row, we plot the curve with $a = 3$, which is far outside the range of amplitudes that were used during training ($a \in [0, 2]$). Notwithstanding, the mean predictions of C-BRUNO remain reasonable. This contrasts the behaviour of NPs in Figure 4.6. As to the standard deviation of the predictive distribution of C-BRUNO, we would expect it to be smaller when we condition on 10 and 100 input points. In some applications, such poor uncertainty quality might be considered as a drawback.

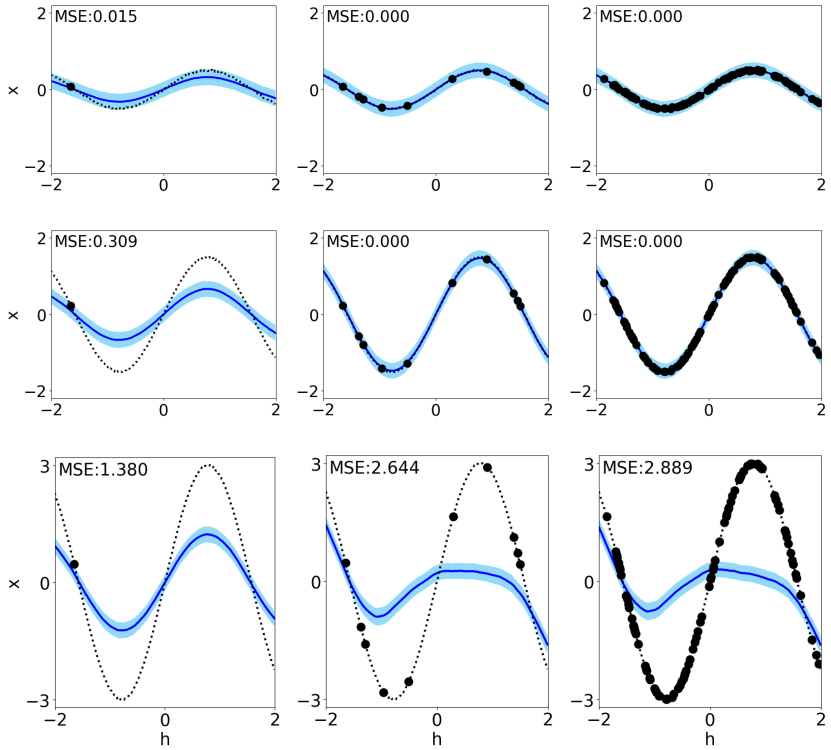
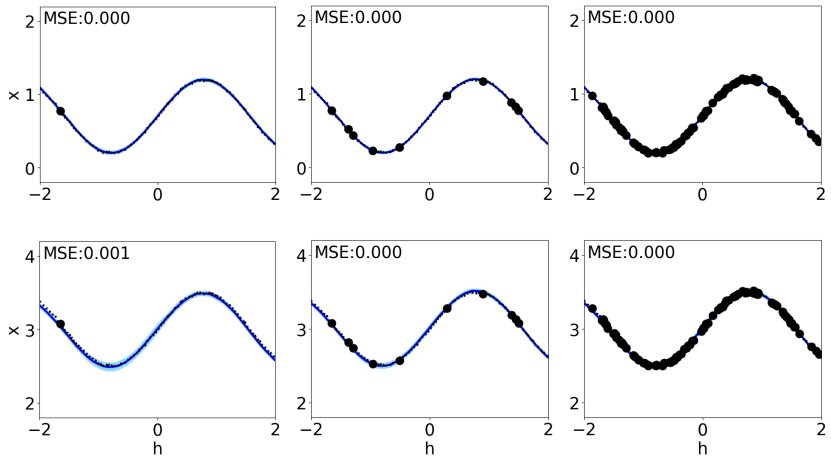
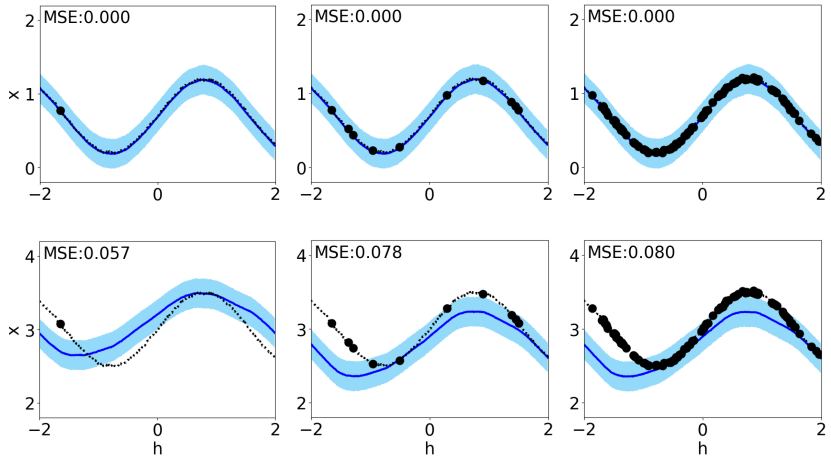


Figure 4.6 1D regression results of NPs with a 2-D latent space on the set of test tasks from Figure 4.5. Notice how NPs behave in comparison to C-BRUNO when presented with data outside the training distribution, i.e. when $a = 3$ in the bottom row. We can also observe an unusual behaviour of the standard deviation, especially for the leftmost column where the uncertainty is clearly underestimated. The likely explanation is that in the authors' implementation of NPs [63], which we used for these experiments, the standard deviation of $p(x_{n+1}|h_{n+1}, \boldsymbol{\theta})$ is bounded from below by 0.1, and the standard deviation of the posterior $p(\boldsymbol{\theta}|x_{1:n}, h_{1:n})$ is limited to $[0.1, 1]$ range. If we modify the code of NPs and allow the standard deviation of $p(\boldsymbol{\theta}|x_{1:n}, h_{1:n})$ to be greater than 1, then the standard deviation of the posterior predictive distribution becomes more reasonable.

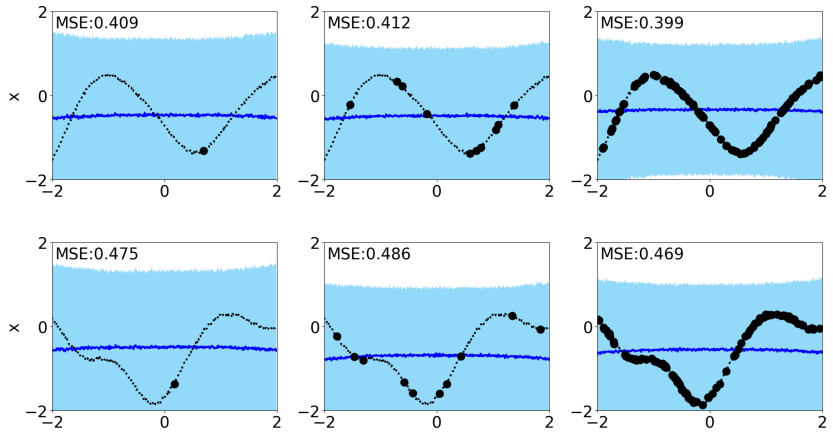


(a) C-BRUNO

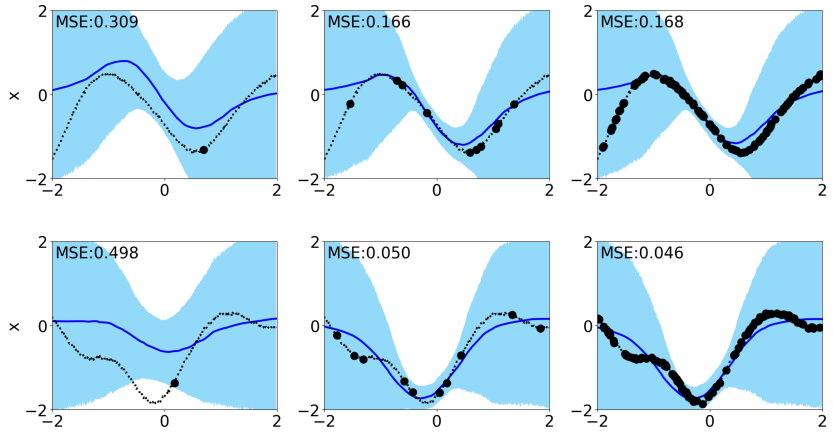


(b) NPs

Figure 4.7 1D regression results of C-BRUNO and NPs on the translated sine wave problem. In blue, we plot the mean and two standard deviations around the mean of samples from a predictive distribution, where we condition on the context points in black. The context length is 1, 10 and 100 for left, middle and right columns respectively. Each row corresponds to a sine wave $x = 0.5\sin(2h) + t$ with $t = \{0.7, 3\}$. The underlying ground truth curve is plotted in black. Note how the predictions of two models differ for $t = 3$ – a value outside of the $[-2, 2]$ range that was used during training.



(a) C-BRUNO



(b) NPs

Figure 4.8 Results of C-BRUNO and NPs with a 2-D latent space on the GP regression problem. In blue, we plot the mean and two standard deviations around the mean of samples from a predictive distribution, where we condition on the context points in black. The context length is 1, 10 and 100 for left, middle and right columns respectively. Each row corresponds to a curve drawn from a GP, which is plotted in black. NPs clearly outperform BRUNO, even though the expressive power of both models is limited as a result of using a 2-D latent space. For C-BRUNO, we could not find a regime where our model starts learning even the general shapes of the curves. The reasons for such behaviour are yet to be explored.

4.6 Discussion and conclusion

In this chapter, we presented C-BRUNO – an extension of BRUNO to the conditional case, which maintains its appealing properties, such as **(a)** exact likelihoods **(b)** fast sampling and inference, **(c)** no retraining or changes to the architecture at test time, and **(d)** a recurrent formulation. These features constitute a powerful meta-learning model with a flexible design. Together, BRUNO and C-BRUNO cover a broad range of meta-learning applications while performing on par with more task-specific state-of-the-art methods. In particular, we showed how C-BRUNO can be used for few-shot conditional image generation.

BRUNO and C-BRUNO build directly on the fundamental property of exchangeability that underlies much of Bayesian statistics. They provide an alternative way to building meta-learning models by shifting away from the commonly used approximate explicit Bayesian inference. BRUNO models combine compound symmetry GPs with powerful bijective feature extractors in the form of flow-based deep neural architectures. While the former component is unlikely to be improved, we expect our models to greatly benefit from the recent advances in normalizing flows, which is currently an active area of research [41, 65]. This would allow us to apply our models to more challenging datasets, thus offering a simpler alternative to more complex models such as GQNs [31].

Even with the current architecture, it would have been desirable to compare C-BRUNO with GQN-like models on more involved tasks such as reconstructing scenes in a room with a variety of objects in different colours, shapes, textures and lights. However, for these problems, models become prohibitively expensive, taking weeks to train on a GPU [71]. While we did not conduct such experiments ourselves, Hou et al. [55] evaluated C-BRUNO, GQN and C-VAE [113] on the task of reconstructing 3D volumes from a few tomographic slices – an important problem in medical imaging. Among the three methods, only C-BRUNO was able to achieve satisfactory results.

Extending NPs to GQNs, such that the model could work with images, is a non-trivial task. To the contrary, C-BRUNO experiences

difficulties in the other direction. While we adapted C-BRUNO to low and one-dimensional inputs, it came at a cost of sacrificing the exactness of our approach. Still, in some of those tasks, C-BRUNO was not able to compete with NPs for the reasons we do not yet understand. However, we noticed surprising generalisation abilities of C-BRUNO in comparison to NPs. In the future, it would be interesting to pinpoint the source of these abilities, and perhaps, combine the strengths of the two approaches. We also hope that given the current amount of research into normalizing flows, the bottleneck problem can be addressed in better ways than augmentation.

5

Conditional BRUNO for meta reinforcement learning

In this chapter, we will apply conditional BRUNO to the problem of task inference in meta-RL settings. The goal and the challenges of meta-RL differ from those of a few-shot image generation problem which we focused on previously. However, the "meta" flavour remains the same and so does the purpose of C-BRUNO: to infer the task from a sequence of observations.

We begin this chapter from a brief introduction to reinforcement learning, where we explain some necessary concepts, terminology and notation along with an outline of RL algorithms that will be used. We then discuss the problem of meta reinforcement learning and describe the existing approaches. Finally, we present our method called *BrunoSAC,* in which an RL agent relies on C-BRUNO for identifying the task that the agent is required to solve. Backed by our experiments, we discuss the relevance of this approach and ponder on the current state of meta-RL research.

5.1 Preliminaries

Key concepts in reinforcement learning

The goal of reinforcement learning is to devise agents that can learn to behave optimally in an unknown environment. Here, an

agent is a decision-maker that needs to choose actions depending on the state of the environment – an entity that the agent interacts with and whose inner workings are unknown to the agent. As a result of its actions, the agent receives a reward, and the environment transitions to a new state. Figure 5.1 illustrates this process. The agent's objective is to maximize a cumulative reward by optimizing its policy – the agent's strategy of selecting actions. We will proceed by presenting these notions more formally.

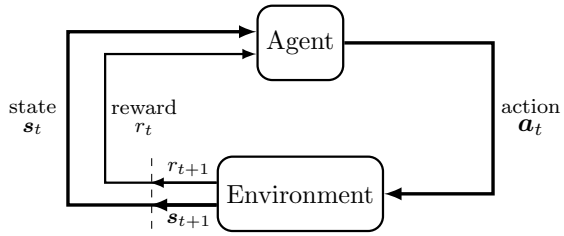


Figure 5.1 The agent–environment interaction process.

A Markov decision process (MDP) is a mathematical basis for describing the RL framework. The main components of MDPs are:

- a set of states $\mathcal{S}$
- a set of agent's actions $\mathcal{A}$
- a transition function $\mathcal{T} = p(\boldsymbol{s}_{t+1} = \boldsymbol{s}' | \boldsymbol{s}_t = \boldsymbol{s}, \boldsymbol{a}_t = \boldsymbol{a})$, which is a probability of transitioning into state $\boldsymbol{s}'$ when executing an action $\boldsymbol{a}$ in state $\boldsymbol{s}$ at time step t. Given a present state $\boldsymbol{s}_t$, the independence on the past states is called a Markov assumption – a property that MDP owes its name to.
- a reward function $\mathcal{R} = p(r_{t+1} = r | \boldsymbol{s}_t = \boldsymbol{s}, \boldsymbol{a}_t = \boldsymbol{a}, \boldsymbol{s}_{t+1} = \boldsymbol{s}')$, which is a probability of receiving a reward r after transitioning from $\boldsymbol{s}$ to $\boldsymbol{s}'$ due to taking an action $\boldsymbol{a}$.

An MDP represents the fully-observed scenario, where the agent has access to the environment's state. However, in many real-world processes, the agent only gets an incomplete description of the state, a so-called observation $\boldsymbol{o}$, which is sampled from the emission probability $p(\boldsymbol{o}|\boldsymbol{s})$. This generalization of the MDP is called a partially observable MDP (POMDP).

We will consider the episodic MDP setting, where the process of acting in the environment can be terminated and started again from a new state sampled from a start-state distribution $p(\boldsymbol{s}_0)$. An episode is terminated, for example, when a certain state or a maximum number of steps is reached. A sequence of states and actions $(\boldsymbol{s}_0, \boldsymbol{a}_0, \boldsymbol{s}_1, \boldsymbol{a}_1, \dots)$ collected within an episode is called a trajectory or a rollout.

A cumulative reward within an episode is called a return, which is defined as $R = \sum_{t=0}^{H} \gamma^t r_t$. Here, γ is a discount factor between 0 and 1. Intuitively, the discounting reflects the fact that we wish to achieve rewards as early as possible. For example, if the agent's goal is to arrive at a certain destination, we prefer it to take the shortest path and reach its goal sooner. The discount factor also makes sure that the sum converges to a finite value when $H = \infty$.

As we mentioned earlier, the core problem in RL is to find a good policy – a rule to choose actions depending on states. There are two types of policies: stochastic and deterministic. The former one is a conditional probability of actions given states: $p(\boldsymbol{a}_t|\boldsymbol{s}_t)$, usually denoted as $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$. At every step t, the agent can sample an action from this policy, i.e. $\boldsymbol{a} \sim \pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$, and execute $\boldsymbol{a}$. One can obtain a deterministic policy from a stochastic one, for instance, by letting $\boldsymbol{a}_t = \arg\max_{\boldsymbol{a}_t} \pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$. In general, a deterministic policy can be defined by any mapping from states to actions: $\boldsymbol{a}_t = \mu(\boldsymbol{s}_t)$.

Finding a good policy is difficult because the environment is unknown: the agent does not know transition and reward functions. Therefore, it needs to acquire some form of this knowledge by interacting with the environment and learning from its experience. Learning about the environment most likely requires choosing suboptimal actions, which is often not aligned with the goal of maximizing the returns. As a consequence, the exploration vs. exploitation dilemma arises [115]: how do we trade off between taking the road less travelled and doing what we currently consider as the most rewarding option. On top of that, there is another challenge: if we receive rewards after performing a sequence of actions, how do we decide which actions were responsible for the end results. This is the so-called credit assignment problem [115].

We do not encounter any of such issues in supervised and unsupervised learning, thus RL requires a different treatment, which we will introduce further.

Our goal is to find an optimal policy π^* that maximizes returns averaged across possible trajectories. A trajectory is a random variable, which we will denote as τ. Its probability depends on the chosen policy π and the environment's transition function, so for a trajectory of H steps, we can write that $p(\tau|\pi, \mathcal{T}) = p(\boldsymbol{s}_0) \prod_{t=0}^{H-1} p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$. Then the expected return over trajectories equals to:

$$J(\pi) = \mathbb{E}_{\tau \sim p(\tau|\pi,\mathcal{T}), r\sim\mathcal{R}}\Big[\sum_{t=0}^{H-1} \gamma^t r_t\Big]. \tag{5.1}$$

In the next section, we will look at some general approaches to the RL problem of maximizing this expected return.

Reinforcement learning algorithms

A general principle behind the construction of RL algorithms is that of alternating learning steps with executing the policy on the environment. During policy execution, we let the policy interact with the environment to collect trajectories and rewards. The learning steps then use the collected data to improve the policy.

The RL algorithms used for the learning step can be roughly divided into two classes: model-based and model-free. In the former case, we estimate the transition and reward functions of the environment based on the collected experiences. Using the model of the environment, the agent can then plan its actions. Planning refers to the process of taking the model as an input and producing or improving a policy [115]. For instance, using transition and reward probabilities given by the model, we can estimate the expected return for any sequence of actions, which allows us to choose a sequence for which it is maximized.

Model-free methods do not rely on having a model but use experiences generated by the environment to directly learn the policy or some quantities that guide the choice of a policy. The

boundary separating model-based from model-free approaches can sometimes be vague. In either case, we will focus on a few RL methods that do not require modelling the transition and reward functions.

As an example, let us consider a parametric policy $\pi_{\boldsymbol{\phi}}$ and use a gradient ascent method to find $\boldsymbol{\phi}$ such that $\boldsymbol{\phi}^* = \arg\max_{\boldsymbol{\phi}} J(\pi_{\boldsymbol{\phi}})$. From the data collected during the policy execution step, the returns are estimated. These returns are then used to compute the gradient of our objective function with respect to $\boldsymbol{\phi}$. In the simplest version of this algorithm, these gradients can be prohibitively noisy, thus impeding learning and making us seek improvements.

Our desire is to have a reliable estimate of the expected return if we start from an arbitrary state $\boldsymbol{s}$ and follow a policy π. This is done by learning a value function: $V^{\pi}(\boldsymbol{s}) = \mathbb{E}_{\tau \sim p(\tau|\pi,\mathcal{T})}\Big[R(\tau)|\boldsymbol{s}_0 = \boldsymbol{s}\Big]$. Here, we denoted the return as $R(\tau)$, and from now on, the expectation with respect to the distribution of rewards will be implicit. In addition to the value function, it is also useful to have an action-value function: $Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{\tau \sim p(\tau|\pi,\mathcal{T})}\Big[R(\tau)|\boldsymbol{s}_0 = \boldsymbol{s}, \boldsymbol{a}_0 = \boldsymbol{a}\Big]$. This Q function gives an expected return if we start in the state $\boldsymbol{s}$, take an action $\boldsymbol{a}$, and only afterwards follow the policy π. The following relationship between the value and action-value functions should be evident: $V^{\pi}(\boldsymbol{s}) = \mathbb{E}_{\boldsymbol{a}\sim\pi}[Q^{\pi}(\boldsymbol{s}, \boldsymbol{a})]$.

Both value and action-value functions obey the Bellman equations:

$$\begin{aligned} V^{\pi}(\boldsymbol{s}) &= \mathbb{E}_{\boldsymbol{a}\sim\pi, \boldsymbol{s}'\sim\mathcal{T}}\Big[r(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') + \gamma V^{\pi}(\boldsymbol{s}')\Big], \\ Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) &= \mathbb{E}_{\boldsymbol{s}'\sim\mathcal{T}}\Big[r(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') + \gamma \mathbb{E}_{\boldsymbol{a}'\sim\pi}[Q^{\pi}(\boldsymbol{s}', \boldsymbol{a}')]\Big]. \end{aligned} \tag{5.2}$$

These equations form the learning objectives for the value and action-value functions. Namely, if we start from arbitrarily initialized $Q^{\pi}(\boldsymbol{s}, \boldsymbol{a})$ and $V^{\pi}(\boldsymbol{s})$, our goal is to make Bellman equations hold by trying to match the right-hand sides to their targets on the left like in an ordinary supervised regression task. Updates of the value and action-value functions can be done using either $(\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}')$ transitions that were collected during any of the policy execution steps or only from the last policy execution step. The

choice between these two options makes for an off-policy or an on-policy method respectively.

Knowledge of the value or the action-value functions can assist in updating the policy. In a group of actor-critic methods, either $V^{\pi}(\boldsymbol{s})$ or $Q^{\pi}(\boldsymbol{s}, \boldsymbol{a})$ can serve as a critic that needs to evaluate and guide the policy $\pi_{\boldsymbol{\phi}}$, here referred to as the actor. Further, we will describe in details a popular instantiation of this approach called soft actor-critic (SAC), which will be used as a part of our meta-RL method.

Soft Actor-Critic

SAC [45, 46] is an off-policy RL algorithm that optimizes a stochastic parametric policy. It is commonly used for environments with continuous actions. In this case, the policy can be modeled by a factorized Gaussian whose mean and variance are the outputs of a neural network that receives states $\boldsymbol{s}$ as an input. We can write it as $\pi_{\boldsymbol{\phi}}(\boldsymbol{a}|\boldsymbol{s}) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{s}), \boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\boldsymbol{s})\boldsymbol{I})$, where $\boldsymbol{\phi}$ is a set of network's parameters. During training, the actions are sampled from $\pi_{\boldsymbol{\phi}}(\boldsymbol{a}|\boldsymbol{s})$, while at test time the deterministic policy is used with $\boldsymbol{a} = \boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{s})$.

SAC is one of the methods implementing a maximum entropy RL approach, where the agent not only trains to maximize the expected return but also maximize the entropy of the policy, which amounts to acting as randomly as possible. This has been observed to improve robustness and exploration [32, 139].

As a maximum entropy RL algorithm, SAC aims to maximize the following objective:

$$J(\pi) = \mathbb{E}_{\tau \sim p(\tau|\pi, \mathcal{T}), r \sim \mathcal{R}} \Big[\sum_{t=0}^{H-1} \gamma^t \Big(r_t + \alpha \mathcal{H}\big(\pi_{\boldsymbol{\phi}}(\cdot|\boldsymbol{s}_t)\big) \Big) \Big].$$

In comparison to the classic RL objective in Eq. 5.1, we see an additional term $\mathcal{H}\big(\pi_{\boldsymbol{\phi}}(\cdot|\boldsymbol{s}_t)\big) = -\mathbb{E}_{\boldsymbol{a} \sim \pi_{\boldsymbol{\phi}}(\boldsymbol{a}|\boldsymbol{s}_t)} \big[\log \pi_{\boldsymbol{\phi}}(\boldsymbol{a}|\boldsymbol{s}_t) \big]$, which is the entropy of the policy at current step. The temperature parameter α weights how important it is to maximize the entropy

compared to the rewards. With the change in the objective, value and Q-value functions are modified accordingly:

$$V^{\pi}(\boldsymbol{s}) = \mathop{\mathbb{E}}_{\tau\sim p(\tau|\pi,\mathcal{T})}\Big[\sum_{t=0}^{H-1}\gamma^t\Big(r_t + \alpha\mathcal{H}\big(\pi_{\boldsymbol{\phi}}(\cdot|\boldsymbol{s}_t)\big)\Big)|\boldsymbol{s}_0=\boldsymbol{s}\Big],$$

$$Q^{\pi}(\boldsymbol{s},\boldsymbol{a}) = \mathop{\mathbb{E}}_{\tau\sim p(\tau|\pi,\mathcal{T})}\Big[\sum_{t=0}^{H-1}\gamma^t r_t + \alpha\sum_{t=1}^{H-1}\gamma^t\mathcal{H}\big(\pi_{\boldsymbol{\phi}}(\cdot|\boldsymbol{s}_t)\big)|\boldsymbol{s}_0=\boldsymbol{s},\boldsymbol{a}_0=\boldsymbol{a}\Big].$$

In this definition of the Q-function, the entropy term is added to all the rewards except for the first one. This results in the following relation between the value and action-value functions:

$$\begin{aligned} V^{\pi}(\boldsymbol{s}) &= \mathbb{E}_{\boldsymbol{a}\sim\pi}Q^{\pi}(\boldsymbol{s},\boldsymbol{a}) + \alpha\mathcal{H}\big(\pi_{\boldsymbol{\phi}}(\cdot|\boldsymbol{s})\big) \\ &= \mathbb{E}_{\boldsymbol{a}\sim\pi}\Big[Q^{\pi}(\boldsymbol{s},\boldsymbol{a}) - \alpha\log\pi_{\boldsymbol{\phi}}(\boldsymbol{a}|\boldsymbol{s})\Big]. \end{aligned} \tag{5.3}$$

Thus, if we learn the Q-function by minimizing the Bellman residual constructed based on Eq. 5.2, then the value function can be trained by trying to match the two sides of Eq. 5.3. Given that both V and Q are modelled by neural networks with their parameters $\boldsymbol{\psi}$ and $\boldsymbol{\xi}$, the two losses with respect to these parameters are:

$$\mathcal{L}_V(\boldsymbol{\psi}) = \mathop{\mathbb{E}}_{\boldsymbol{s}_t\sim\mathcal{D}}\left[\frac{1}{2}\left(V^{\pi}_{\boldsymbol{\psi}}(\boldsymbol{s}_t) - \mathop{\mathbb{E}}_{\boldsymbol{a}_t\sim\pi}\Big[Q^{\pi}_{\boldsymbol{\xi}}(\boldsymbol{s}_t,\boldsymbol{a}_t) - \alpha\log\pi_{\boldsymbol{\phi}}(\boldsymbol{a}_t|\boldsymbol{s}_t)\Big]\right)^2\right]$$

$$\mathcal{L}_Q(\boldsymbol{\xi}) = \mathop{\mathbb{E}}_{\boldsymbol{s}_t,\boldsymbol{a}_t\sim\mathcal{D}}\left[\frac{1}{2}\left(Q^{\pi}_{\boldsymbol{\xi}}(\boldsymbol{s}_t,\boldsymbol{a}_t) - \Big(r_t + \gamma\mathop{\mathbb{E}}_{\boldsymbol{s}'_t\sim\mathcal{T}}\big[V^{\pi}_{\boldsymbol{\psi}}(\boldsymbol{s}'_t)\big]\Big)\right)^2\right].$$

Here, the outer expectation is taken over states and actions sampled from a replay buffer $\mathcal{D}$, which stores past transitions $(\boldsymbol{s},\boldsymbol{a},r,\boldsymbol{s}')$ generated by policies that existed earlier on during training. This is a defining feature of the off-policy methods. In the expression for $\mathcal{L}_V$, the expectation with respect to actions is estimated by sampling one action from the current policy. Similarly, a single $\boldsymbol{s}'_t$ that corresponds to $(\boldsymbol{s}_t,\boldsymbol{a}_t)$ is taken from the replay buffer to estimate the inner expectation in the expression for $\mathcal{L}_Q$.

To learn the policy, we can use the fact that the optimal policy maximizes the value function in each state. Thus, taking the

definition of the value function from Eq. 5.3, we can write down the loss with respect to policy parameters ϕ:

$$\mathcal{L}_\pi(\phi) = -\mathbb{E}_{\boldsymbol{s}_t \sim \mathcal{D}, \boldsymbol{a}_t \sim \pi_\phi} \Big[Q^\pi_{\boldsymbol{\xi}}(\boldsymbol{s}_t, \boldsymbol{a}_t) - \alpha \log \pi_\phi(\boldsymbol{a}_t | \boldsymbol{s}_t) \Big]. \tag{5.4}$$

In Section 2.3.2 on VAEs, we have seen the reparameterization trick – a way of dealing with objectives whose derivative we need to take with respect to parameters that appear in the distribution over which the expectation is taken. In this case, such parameters are ϕ. Using the trick, we can represent actions as a deterministic function of a Gaussian noise $\boldsymbol{\epsilon}$ as $\boldsymbol{a}_t = g_\phi(\boldsymbol{s}_t, \boldsymbol{\epsilon})$. Instead of Eq. 5.4, we can then optimize the following loss:

$$\mathcal{L}_\pi(\phi) = -\mathbb{E}_{\boldsymbol{s}_t \sim \mathcal{D}, \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})} \Big[Q^\pi_{\boldsymbol{\xi}}\big(\boldsymbol{s}_t, g_\phi(\boldsymbol{s}_t, \boldsymbol{\epsilon})\big) - \alpha \log \pi_\phi\big(g_\phi(\boldsymbol{s}_t, \boldsymbol{\epsilon}) | \boldsymbol{s}_t\big) \Big].$$

Given the losses for the critics and the actor, the training proceeds as described in Algorithm 1. Here, we explained the backbone of SAC that does not include tricks for improving training stability. Neither did we mention a formulation that allows to automatically tune the temperature parameter α [45]. While in practice we implemented the elaborate version of SAC, its detailed explanation would be excessive for the purpose of this thesis.

5.2 Meta reinforcement learning

Meta-learning intends to bridge the gap between machine and human learning by designing algorithms that acquire prior knowledge from a multitude of training tasks and use these priors for a rapid adaptation to new tasks. In reinforcement learning, this goal is translated into building agents able to adjust their policies to new environment settings after a handful of environment interactions and with little or no retraining. To give an example, let us consider a problem in which an agent is rewarded for reaching a certain point on a plane. If the location of the point is fixed, this is a standard RL problem dealing with a single MDP. For meta-RL, we can define multiple MDPs by changing the target location and say that each MDP corresponds to a different task. Note that the agent does not know the target location, and so it can only rely on

Algorithm 1 Soft Actor-Critic

Initialize parameters ψ, ξ, ϕ and create a replay buffer $\mathcal{D} = \emptyset$
for each iteration **do**
  % policy execution
  for each environment step **do**
    $\boldsymbol{a}_t \sim \pi_\phi(\boldsymbol{a}_t|\boldsymbol{s}_t)$
    $\boldsymbol{s}'_t \sim p(\boldsymbol{s}'|\boldsymbol{s}_t, \boldsymbol{a}_t)$
    $r_t \sim p(r|\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}'_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup (\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}'_t)$
  end for
  % learning steps
  for each gradient step **do**
    $\psi \leftarrow \psi - \lambda_V \nabla_\psi \mathcal{L}_V(\psi)$
    $\xi \leftarrow \xi - \lambda_Q \nabla_\xi \mathcal{L}_Q(\xi)$
    $\phi \leftarrow \phi - \lambda_\pi \nabla_\phi \mathcal{L}_\pi(\phi)$
  end for
end for

the rewards when trying to choose which direction to move. By training the agent on a number of such tasks, we want it to gain the ability to adapt to new settings of the target location.

Vanilla RL algorithms, designed to deal with a single MDP at a time, are inept for the meta-RL problem. The latter can be formulated as a special case of POMDP: one in which only the task specification is hidden from the agent. However, as we had in the example above, each task on its own can be described by an MDP. Thus, how can we adapt RL agents to these conditions? Further, we will look into a few general approaches to this problem.

One approach to meta-RL proposed by Duan et al. [25] and Wang et al. [130] is based on the idea of representing policies using RNNs. Here, the input to the policy π_ϕ is not only state $\boldsymbol{s}_t$ at a current step t, but also actions and rewards from the previous step $t-1$. In this way, the recurrent policy implements a probability of taking actions conditionally on the past transitions, i.e. $\pi_\phi(\boldsymbol{a}_t|\boldsymbol{s}_{0:t}, r_{0:t-1}, \boldsymbol{a}_{0:t-1})$. Here, the knowledge of history is what allows us to implicitly infer the task. Training of the policy can be done via any standard RL algorithm, which is referred to as "slow" RL since it requires large amounts of training data. Then the "fast"

RL part, which is responsible for quick adaptation to new tasks, gets encoded in the dynamics of the RNN. This dynamics can be viewed as an RL algorithm on its own [13].

Optimisation-based methods comprise another prominent group in meta-RL [34, 100, 114]. Their idea is to learn initial parameters ϕ_0 of a policy π_ϕ such that a few gradient updates are needed for ϕ_0 to obtain a policy adapted for a new task. These methods typically work with non-recurrent policies, so they are lighter compared to RNN-based methods.

Finally, a group of methods that includes our BrunoSAC is based on the idea of separating the algorithm into inference and acting modules [58, 94, 140]. The former is responsible for inferring the task from a sequence of interactions, while the latter needs to choose actions conditionally on the results of inference. This is motivated by the Bayesian approach to solving the special case POMDP we talked about earlier, which in this context is also known as the Bayes-adaptive MDP (BAMDP) [26]. In the next section, we will describe this approach in greater details on the example of BrunoSAC. Related methods will be reviewed in Section 5.4.

5.3 BrunoSAC

As we said, our BrunoSAC method will follow the approach of separating inference and control. The task of the inference module is to maintain a belief state over what the underlying MDP might be. As in most problems, we assume that MDPs have the same action and state spaces but different transition and/or reward functions. Thus, the belief can be equivalently formulated over MDP's transition and reward functions. At any given step t, this belief is a posterior distribution $p(\mathcal{T}, \mathcal{R} | \boldsymbol{s}_{0:t}, \boldsymbol{a}_{0:t}, r_{0:t}, \boldsymbol{s}'_{0:t})$, where we condition on previously observed transitions within a trajectory. Alternatively, we can reason in terms of beliefs $p(\boldsymbol{\theta} | \boldsymbol{s}_{0:t}, \boldsymbol{a}_{0:t}, r_{0:t}, \boldsymbol{s}'_{0:t})$ over a latent variable $\boldsymbol{\theta}$ that encapsulates task specification and thus determines which MDP we are in, i.e. $\mathcal{T} = p(\boldsymbol{s}' | \boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta})$ and $\mathcal{R} = p(r | \boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}', \boldsymbol{\theta})$. In either case, the posterior is often intractable.

VAEs [66] have been previously adapted to approximate the belief state [140]. Here, we explore an alternative way of doing so with C-BRUNO. In the previous chapter, we constructed C-BRUNO without a reference to the posterior, so we will have to revise an alternative formulation from Section 3.4 before we can turn C-BRUNO into a full-fledged replacement to VAEs in meta-RL. With the addition of minor architectural changes, we use C-BRUNO in conjunction with SAC to get a competitive meta-RL algorithm, which we called BrunoSAC to reflect its two-parts nature. Based on a couple of benchmarks, we show that our method is sample efficient, fast and easy to train, and it can adapt to the test tasks given a small number of observations.

In the next few parts, we will explain the ways in which the exchangeability assumption fits into the MDP framework, how to compute the posterior over $\boldsymbol{\theta}$ using C-BRUNO, what architectural changes are needed to make C-BRUNO work with low-dimensional inputs, and finally, how to combine all of these ideas together with SAC into a single meta-RL agent.

Exchangeability in MDPs

When applying BRUNO to a new problem, the first question we need to ask ourselves is whether it is reasonable to assume exchangeability of the variables we wish to model. Let us, therefore, revise the concept of exchangeability and see how it is applicable for the case of MDPs. In particular, we are interested in regression exchangeable processes as described in Section 4.1.

For a stochastic process $\boldsymbol{x}(\boldsymbol{h}_0), \boldsymbol{x}(\boldsymbol{h}_1), \boldsymbol{x}(\boldsymbol{h}_2), \ldots$, where in statistical terms, $\boldsymbol{x}$ is a dependent and $\boldsymbol{h}$ is an independent variable, regression exchangeability amounts to:

$$p(\boldsymbol{x}_0, \ldots, \boldsymbol{x}_n | \boldsymbol{h}_0, \ldots, \boldsymbol{h}_n) = p\left(\boldsymbol{x}_{\pi(0)}, \ldots, \boldsymbol{x}_{\pi(n)} | \boldsymbol{h}_{\pi(0)}, \ldots, \boldsymbol{h}_{\pi(n)}\right), \tag{5.5}$$

which holds for any finite n and any permutation π of $\{0, \ldots, n\}$.

In MDPs, we deal with sequences of state-action-reward-state transitions $(\boldsymbol{s}_0, \boldsymbol{a}_0, r_0, \boldsymbol{s}'_0), (\boldsymbol{s}_1, \boldsymbol{a}_1, r_1, \boldsymbol{s}'_1), \ldots$. As we will explain

shortly, it is reasonable to assume that these transitions form a regression exchangeable process with $\boldsymbol{h}_i = (\boldsymbol{s}_i, \boldsymbol{a}_i)$ and $\boldsymbol{x}_i = (r_i, \boldsymbol{s}'_i)$ if we use the notation from Eq. 5.5. de Finetti's theorem would then give:

$$p(r_{t+1}, \boldsymbol{s}'_{t+1} | \boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}, \boldsymbol{\tau}_{0:t}) = \int p(r_{t+1}, \boldsymbol{s}'_{t+1} | \boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}, \boldsymbol{\theta}) p(\boldsymbol{\theta} | \boldsymbol{\tau}_{0:t}) d\boldsymbol{\theta}, \tag{5.6}$$

where $\boldsymbol{\tau}_{0:t} = \{(\boldsymbol{s}_0, \boldsymbol{a}_0, r_0, \boldsymbol{s}'_0), \ldots, (\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}'_t)\}$ denotes all observed transitions up to step t either in their natural or in a permuted order. The former assumes that $\boldsymbol{s}'_t = \boldsymbol{s}_{t+1}$, while this is not necessarily true for the latter. Here, we also switched to indexing the variables using t instead of n, which is a more natural notation for RL.

Using de Finetti's theorem we can reason out why regression exchangeability of $(r, \boldsymbol{s}')$ given $(\boldsymbol{s}, \boldsymbol{a})$ is a valid assumption. This theorem allows us to think about exchangeable processes as sequences of random variables that are i.i.d. conditionally on an underlying latent factor, which we denoted by $\boldsymbol{\theta}$. In our case, $\boldsymbol{\theta}$ encapsulates the knowledge of the MDP with its reward and transition functions. Then, given $\boldsymbol{\theta}$, and conditionally on $(\boldsymbol{s}, \boldsymbol{a})$, $(r, \boldsymbol{s}')$ become i.i.d.. This conditional independence might still seem unintuitive, especially since we often think of $(\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}')$ transitions in the order they appear in a trajectory. Therefore, it is important to additionally remember the Markov property and note that the current state $\boldsymbol{s}$ is always the conditioning event.

If we wish to model the distributions involved in Eq. 5.6 using VAE-based models, such as neural processes [37], we need to approximate the posterior $p(\boldsymbol{\theta} | \boldsymbol{\tau}_{0:t})$ and derive a lower bound on $\log p(r_{t+1}, \boldsymbol{s}'_{t+1} | \boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}, \boldsymbol{\tau}_{0:t})$. C-BRUNO, on the other hand, can directly model this predictive distribution by constructing a suitable exchangeable process while ignoring the integral on the right-hand side. A corresponding schematic is given in Figure 5.2. Note, that by learning to predict rewards and next states, we effectively learn to model the environment as one would do in model-based meta-RL [103]. While we could use a planning algorithm on the trajectories generated by C-BRUNO, this is different from the approach we intended to follow. Thus, having established

the validity of the exchangeability assumption and suitability of C-BRUNO, we can now turn to the next point on our list – the problem of computing the posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{\tau}_{0:t})$.

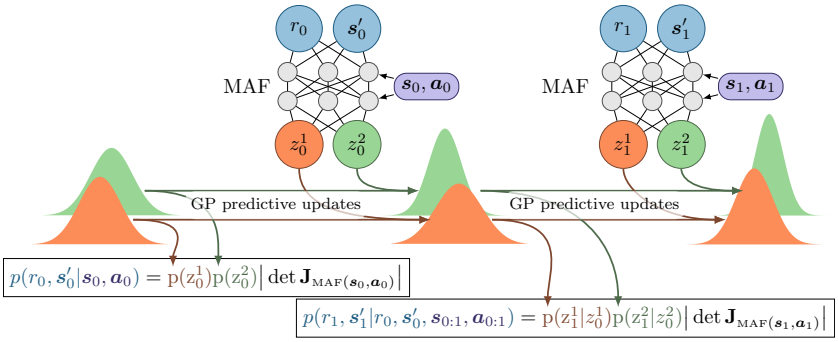


Figure 5.2 A schematic of C-BRUNO for modelling the predictive distribution of rewards and next states in an MDP. The figure illustrates how to update the prior and the predictive distribution in the feature space $\mathcal{Z}$ of the bijective neural network, and how to evaluate the predictive distribution at a particular $(r, \boldsymbol{s}')$ input using the change of variables formula. Though it is not depicted here, sampling of rewards and next states from the predictive distribution is also possible.

Posterior computation

In the notation of C-BRUNO, the posterior $p(\boldsymbol{\theta}|\boldsymbol{\tau}_{0:t})$ corresponds to $p(\boldsymbol{\theta}|\boldsymbol{x}_{0:t}, \boldsymbol{h}_{0:t})$, where we set $\boldsymbol{x} = (r, \boldsymbol{s}')$ and $\boldsymbol{h} = (\boldsymbol{s}, \boldsymbol{a})$. The main derivation of C-BRUNO makes no reference to this posterior, however it exists in the alternative formulation as we showed in Section 3.4. In brief, the use of a bijective mapping and independent dimensions in $\mathcal{Z}$ implies that $p(\boldsymbol{\theta}|\boldsymbol{y}_{0:t}, \boldsymbol{x}_{0:t}) = p(\boldsymbol{\theta}|\boldsymbol{z}_{0:t}) = \prod_{d=1}^{D} p(\theta^d|z_{0:t}^d)$. Thus, we only need an expression for the univariate posterior $p(\theta^d|z_{0:t}^d)$. Further, we will drop the index d since the same equations hold for every dimension in $\mathcal{Z}$, but note that a GP associated with a d-th dimension has its own values of v and ρ that parameterise the GP's kernel matrix. The mean and variance of the Gaussian posterior $p(\theta|z_{0:t})$ are given in Eq. 3.4, for which we also have the following recurrent updates as derived

in Appendix A.1.4:

$$\begin{aligned} \mu_{t+1} &= (1 - d_t)\mu_t + d_t z_t \\ \sigma^2_{t+1} &= (1 - d_t)(\sigma^2_t - v + \rho), \end{aligned} \tag{5.7}$$

with $d_t = \frac{\rho}{v+\rho(t-1)}$, $\mu_0 = 0$ and $\sigma^2_0 = \rho$.

It implies that within one C-BRUNO model, we have two types of recurrent updates: one for parameters of the posterior predictive distribution $p(r_{t+1}, \boldsymbol{s}'_{t+1} | \boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}, \boldsymbol{\tau}_{0:t})$, and one for parameter of the posterior $p(\boldsymbol{\theta}|\boldsymbol{\tau}_{0:t})$. Fortunately, the equations are almost identical, i.e. the means of two distributions are equal, and their variances differ by a constant. Thus, computing the posterior in addition to the predictive posterior incures no additional costs on the part of GPs.

Combining C-BRUNO and SAC

We can now combine the modified C-BRUNO model with a soft actor-critic into a meta-RL algorithm. Most of our choices resemble those used in VariBAD [140], PEARL [94] and Belief [58] – methods that will be discussed in the next section.

The crux of BrunoSAC is to have the SAC policy depend on parameters of the posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{\tau}_{0:t})$ given by BRUNO. As we showed, this posterior is a multivariate Gaussian with independent dimensions, so it can be described by its mean and variance parameters: $\boldsymbol{m}^{\boldsymbol{\theta}}_t = \{\boldsymbol{\mu}_t, \boldsymbol{\sigma}^2_t\}$. Having observed $t \geq 0$ transitions, the agent can sample an action from a stochastic policy conditioned on a state $\boldsymbol{s}$ and current posterior parameters $\boldsymbol{m}^{\boldsymbol{\theta}}_t$, i.e. $\boldsymbol{a} \sim \pi_{\boldsymbol{\phi}}(\boldsymbol{s}, \boldsymbol{m}^{\boldsymbol{\theta}}_t)$. This process is illustrated in Figure 5.3 and explained in Algorithm 2.

Training of BrunoSAC

Training of BrunoSAC can be well separated into learning to model the environments with C-BRUNO and learning the actor-critic functions of SAC. To be precise, the gradients of SAC losses are not propagated through parameters of C-BRUNO, even though these

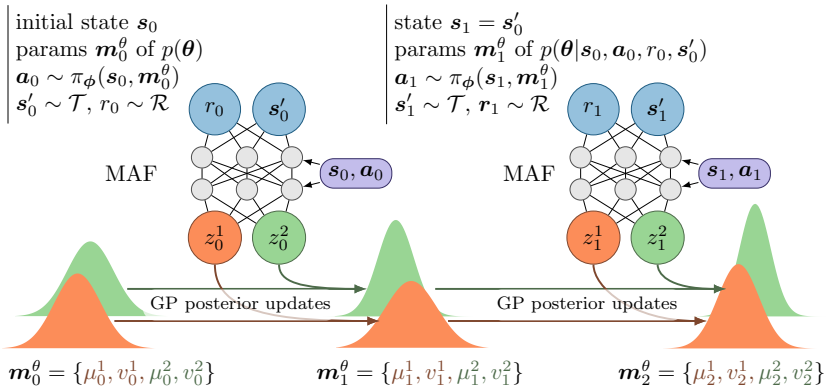


Figure 5.3 Policy roll-out in BrunoSAC.

Algorithm 2 BrunoSAC policy roll-out

sample the initial state s_0
set $m_0^\theta = [0, \ldots, 0, \rho^1, \ldots, \rho^D]$ – parameters of the prior $p(\theta)$
initialise $\tau = \emptyset$
for $t \leftarrow 0$ to H **do**
  $a_t \sim \pi_\phi(a_t|s_t, m_t^\theta)$
  $s'_t \sim p(s'|s_t, a_t)$
  $r_t \sim p(r|s_t, a_t, s'_t)$
  $\tau \leftarrow \tau \cup (s_t, a_t, r_t, s'_t)$
  $s_{t+1} \leftarrow s'_t$
  $m_{t+1}^\theta \leftarrow update(m_t^\theta, (s_t, a_t, r_t, s'_t))$
end for

parameters directly influence the policy via m_t^θ. Nevertheless, updates of both modules are done in parallel, except that we allow for a short pre-training of C-BRUNO on data from a random policy, i.e. $\pi_{\text{random}}(a_t|s_t) = \mathcal{U}(a_{\min}, a_{\max})$, where actions are sampled uniformly within $\mathcal{A}$. The pre-training helps to make m_t^θ more meaningful before the SAC policy starts using it.

To construct training sequences for C-BRUNO and SAC, we use a single replay buffer filled with off-policy data. We create each sequence τ by sampling (s, a, r, s') transitions that come from the same MDP but not necessarily from the same trajectory. Since the design of the meta-training stage is under the researchers' control, we know which transitions belong to which MDPs. Such

information is considered privileged, meaning that it can be used to facilitate training, but it is unavailable at test time [58].

Given a sequence $\boldsymbol{\tau}_{0:H-1}$ of H transitions, the objective of C-BRUNO is to maximize the likelihood of each observed $(r_t, \boldsymbol{s}'_t)$ given $(\boldsymbol{s}_t, \boldsymbol{a}_t)$ and a part of the sequence up to step k. In other words, the goal is to "decode" both past and future transitions with respect to k. In this case, the loss can be written as:

$$\mathcal{L}_{Bruno}(\boldsymbol{\omega}) = -\frac{1}{H}\sum_{t=0}^{H-1} \log p(r_t, \boldsymbol{s}'_t | \boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{\tau}_{0:k}), \tag{5.8}$$

where $\boldsymbol{\omega}$ denotes weights of MAF and variance-covariance parameters of the GPs, i.e. v^d and ρ^d for $d = 1, \ldots, D$. Ideally, one would sample a random k for every training sequence, however, we found that sampling k per batch of sequences or even choosing a fixed k works well in practice.

Given the nature of the meta-RL problem, there is a certain flexibility in choosing what to model. For example, if we know that MDPs differ only with respect to their reward function $\mathcal{R}$, then we can safely withhold from modelling the next state distribution $\mathcal{T}$ since it does not contribute any information to our posterior that could help the policy to identify the task. However, once we are left with predicting scalar rewards, the bottleneck problem becomes acute, and so we need to use the augmented input space as described in Section 4.3.

Training of the policy in BrunoSAC becomes more involved in comparison to the original SAC algorithm, which is a result of having to deal with a recurrent policy. While our policy is not explicitly an RNN, it can be still viewed as one. To see this, we can consider the policy $\pi_{\boldsymbol{\phi}}(\boldsymbol{s}, \boldsymbol{m}_t^{\boldsymbol{\theta}})$ as a fixed transformation on top of the state $\boldsymbol{m}_t^{\boldsymbol{\theta}}$, which is updated recurrently according to Eq. 5.7. Critic functions, on the other hand, can be conditioned on the true task ID or task specification $\boldsymbol{\theta}$ instead of $\boldsymbol{m}_t^{\boldsymbol{\theta}}$, which turns them into simple feed-forward neural networks. Not taking advantage of this information would make the training needlessly harder. In either case, whether we condition on the value of θ or the beliefs over θ, critic functions are discarded at test time.

According to the choices we made above, for a single sequence $\boldsymbol{\tau}_{0:H-1}$ from the replay buffer, whose transitions $(\boldsymbol{s}_0, \boldsymbol{a}_0, r_0, \boldsymbol{s}'_0), \ldots, (\boldsymbol{s}_{H-1}, \boldsymbol{a}_{H-1}, r_{H-1}, \boldsymbol{s}'_{H-1})$ come from the same MDP as specified by $\boldsymbol{\theta}$, we can write the SAC losses as follows:

$$\mathcal{L}_V(\boldsymbol{\psi}) = \frac{1}{2H} \sum_{t=0}^{H-1} \left[V_{\boldsymbol{\psi}}^{\pi}(\boldsymbol{s}_t, \boldsymbol{\theta}) - \bar{V}_t \right]^2 \text{ with}$$

$$\bar{V}_t = \mathbb{E}_{\boldsymbol{a}_t \sim \pi_{\boldsymbol{\phi}}\left(\boldsymbol{s}_t, \boldsymbol{m}_t^{\boldsymbol{\theta}}(\boldsymbol{\tau}_{0:t})\right)} \left[Q_{\boldsymbol{\xi}}^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{\theta}) - \alpha \log \pi_{\boldsymbol{\phi}}\left(\boldsymbol{a}_t | \boldsymbol{s}_t, \boldsymbol{m}_t^{\boldsymbol{\theta}}(\boldsymbol{\tau}_{0:t})\right) \right],$$

$$\mathcal{L}_Q(\boldsymbol{\xi}) = \frac{1}{2H} \sum_{t=0}^{H-1} \left[Q_{\boldsymbol{\xi}}^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{\theta}) - \left(r_t + \gamma V_{\boldsymbol{\psi}}^{\pi}(\boldsymbol{s}'_t, \boldsymbol{\theta}) \right] \right]^2,$$

$$\mathcal{L}_\pi(\boldsymbol{\phi}) = -\frac{1}{H} \sum_{t=0}^{H-1} \mathbb{E}_{\boldsymbol{a}_t \sim \pi_{\boldsymbol{\phi}}\left(\boldsymbol{s}_t, \boldsymbol{m}_t^{\boldsymbol{\theta}}(\boldsymbol{\tau}_{0:t})\right)} \Big[Q_{\boldsymbol{\xi}}^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{\theta}) - \alpha \log \pi_{\boldsymbol{\phi}}\left(\boldsymbol{a}_t | \boldsymbol{s}_t, \boldsymbol{m}_t^{\boldsymbol{\theta}}(\boldsymbol{\tau}_{0:t})\right) \Big].$$

The description of the core training procedure of BrunoSAC is given in Algorithm 3.

5.4 Related work

In this section, we will focus on Belief [58], PEARL [94] and VariBAD [140] – three methods that are most directly related to ours. Like BrunoSAC, they implement the approach of conditioning the policy on results of the task inference. Their main differences can be identified by asking the next questions:

1. how is privileged information, i.e. task IDs or specifications, used during meta-training?
2. what type of model is used to process sequences of transitions?
3. what information from the inference module is passed on to the policy and, optionally, to the value functions?

Algorithm 3 BrunoSAC training

training tasks $\{T_n\}_{n=1,\dots,N_{\text{tasks}}}$
replay buffers $\{\mathcal{D}_n = \emptyset\}_{n=1,\dots,N_{\text{tasks}}}$
for $i \leftarrow 1$ to N_{iter} **do**
  % policy execution
  for each T_n **do**
    if $i > N_{\text{pretrain}}$ **then**
      roll out the policy $\pi_{\boldsymbol{\phi}}$ for H steps by following Algorithm 2
    else
      roll out the random policy π_{random} for H steps
    end if
    $\mathcal{D}_n \leftarrow \mathcal{D}_n \cup \{(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}'_t)\}_{t=0,\dots,H-1}$
  end for
  % learning steps
  for $j \leftarrow 1$ to N_{updates} **do**
    % collect a batch of losses
    for $b \leftarrow 1$ to N_{batch} **do**
      $n \sim \mathcal{U}(1, N_{\text{tasks}})$
      $\boldsymbol{\tau} \sim \mathcal{D}_n$
      compute $\mathcal{L}^b_{Bruno}(\boldsymbol{\omega})$, $\mathcal{L}^b_V(\boldsymbol{\psi})$, $\mathcal{L}^b_Q(\boldsymbol{\xi})$, $\mathcal{L}^b_\pi(\boldsymbol{\phi})$ given $\boldsymbol{\tau}$
    end for
    % model updates
    $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - \lambda_{Bruno} \nabla_{\boldsymbol{\omega}} \left(\frac{1}{N_{\text{batch}}} \sum_b \mathcal{L}^b_{Bruno}(\boldsymbol{\omega}) \right)$
    % actor-critic updates
    if $i > N_{\text{pretrain}}$ **then**
      $\boldsymbol{\psi} \leftarrow \boldsymbol{\psi} - \lambda_V \nabla_{\boldsymbol{\psi}} \left(\frac{1}{N_{\text{batch}}} \sum_b \mathcal{L}^b_V(\boldsymbol{\psi}) \right)$
      $\boldsymbol{\xi} \leftarrow \boldsymbol{\xi} - \lambda_Q \nabla_{\boldsymbol{\xi}} \left(\frac{1}{N_{\text{batch}}} \sum_b \mathcal{L}^b_Q(\boldsymbol{\xi}) \right)$
      $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \lambda_\phi \nabla_{\boldsymbol{\phi}} \left(\frac{1}{N_{\text{batch}}} \sum_b \mathcal{L}^b_\pi(\boldsymbol{\phi}) \right)$
    end if
  end for
end for

4. which RL algorithm is used?

In what follows, we will answer these questions for each of the methods.

Belief uses a supervised approach to learn the belief state. Namely, the inference network is trained to directly predict the task description or its ID given a trajectory. To process the trajectories, Belief uses an LSTM-based architecture [52]. Features from the penultimate layer of this model are passed to the actor and critics of an off-policy SVG(0) [48] or an on-policy PPO [108] RL algorithm. The off-policy method is concluded to be preferable for its sample efficiency.

PEARL is identical to BrunoSAC with respect to how it groups transitions according to their tasks. Moreover, it makes similar exchangeability assumptions and constructs a permutation-invariant inference network for $p(\boldsymbol{\theta}|\boldsymbol{\tau}_{0:t})$, though based on VAEs, in the same way as NPs would do. While PEARL allows for a decoder that could predict future states and rewards, the authors prefer to predict q-values instead. The policy is trained with SAC, where both policy and the critics are conditioned on samples from the posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{\tau})$. Using samples is perhaps the reason why PEARL needs a lot more transitions before converging to a reasonable behaviour at test time, while some methods adapt after very few steps.

VariBAD uses no privileged information during training and works based on trajectories. To process them, VariBAD applies VAEs with a recurrent neural network as an encoder and an MLP decoder that predicts past and future rewards r and states $\boldsymbol{s}'$. Parameters of the posterior distribution are supplied to the policy and critics of PPO. Since PPO is an on-policy method, VariBAD is sample inefficient. Combined with a slow recurrent encoder, this increases the training time of VariBAD in comparison, for instance, to PEARL.

To conclude, each of these methods makes different design choices, whose compatibility, in our opinion, is sometimes unjustified. The reason is that they trade off some desirable properties such as simplicity of the implementation, sample efficiency, fast adaptation at test time or short training times.

BrunoSAC is, therefore, our attempt in combining elements that we think are most sensible with respect to the above criteria.

5.5 Experiments

We applied BrunoSAC to two popular meta-RL benchmarks introduced by Finn et al. [34]: Cheetah-Dir and Cheetah-Vel. Both benchmarks are based on the simulated half-cheetah robot [121] shown in Figure 5.4. The 18-dimensional state space of the half-cheetah is described by positions, angles and velocities of its joints and the spine. One of the dimensions that corresponds to the horizontal position of the spine is unobserved. However, since the problem is invariant under the translation of the horizontal position, half-cheetah remains a fully observable environment. The action space is 6-dimensional, and every action applies torque to each of the 6 hinge joints. One peculiar fact about the half-cheetah robot is that its body is better suited for running backward rather than forward.

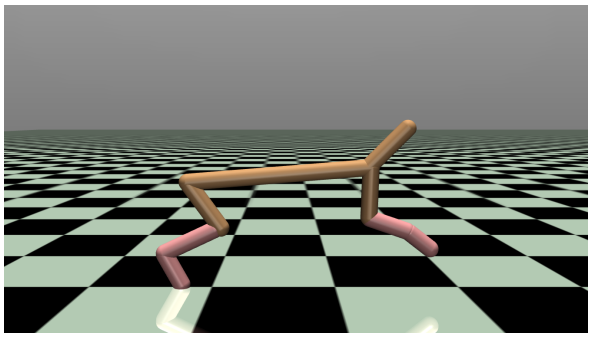


Figure 5.4 MuJoCo-simulated half-cheetah environment.

In the Cheetah-Dir benchmark, the robot needs to run as fast as possible either forward or backward. Thus, the rewards are given by the magnitude of the velocity in either direction. The two directions define the only two tasks, so we use them both during training and testing. While such setup is not ideal in terms of estimating the agent's abilities to adapt to the unseen tasks, Cheetah-Dir is still a difficult problem that cannot be solved by RL algorithms with non-recurrent policies. On the other hand,

Cheetah-Vel does have a different set of tasks for training and testing. Here, in each task, the robot needs to run with a certain velocity. Thus, the reward is calculated as the negative absolute difference between the current and target velocities. There are 100 train and 30 test tasks with target velocity sampled once from $\mathcal{U}(0, 3)$. Such settings were previously used in PEARL and VariBAD. However, we will not directly compare these methods to BrunoSAC since a fair comparison is only possible if all three methods run under the same conditions. In either case, we will try to relate our results to those of PEARL and VariBAD when it is meaningful to do so.

Before we look at the results of BrunoSAC, let us discuss several hyperparameter choices. During training, we roll out the trajectories of length $H = 200$, however, it does not oblige us to train BrunoSAC on sequences of the same length. In our experiments, we used sequences of 100 transitions, and for every batch of sequences, we sampled k from $\mathcal{U}(25, 75)$ to compute the loss in Eq. 5.8. Since in Cheetah-Dir and Cheetah-Vel the states transition function is the same across all tasks, we train BRUNO to predict only the rewards. In order to use MAF, we add 4 extra Gaussian noise dimensions, which results in having a 5-dimensional latent space for $\boldsymbol{\theta}$. A complete list of hyperparameters is given in Appendix A.2.3.

In Figure 5.5 we plot learning curves of BrunoSAC and oracle SAC on the Cheetah-Dir benchmark. The oracle has its policy conditioned on the true task specification, i.e. one-hot encoding of the forward-backward direction. We see that after around 500 thousand collected transitions, BrunoSAC starts matching the oracle's performance, which could only mean that tasks are inferred correctly. To our best estimate, this is at least an order of magnitude fewer steps than required for PEARL and VariBAD, which respectively need to train for 24 hours and 48 hours as reported by Zintgraf et al. [140]. For comparison, it takes about 8 hours to train BrunoSAC on a laptop CPU.

Figure 5.6 plots the learning curves of BrunoSAC and oracle SAC for the Cheetah-Vel problem. Here, the oracle SAC is trained on 30 test tasks, and its policy is conditioned on scalar target velocity. From the figure, we see that BrunoSAC closely approaches the performance of the oracle. For a rough comparison, VariBAD

requires twice as many environment steps to catch up with the oracle, while PEARL needs a bit more than a million steps, which is surprising since it amounts to learning only from 50 trajectories per training task.

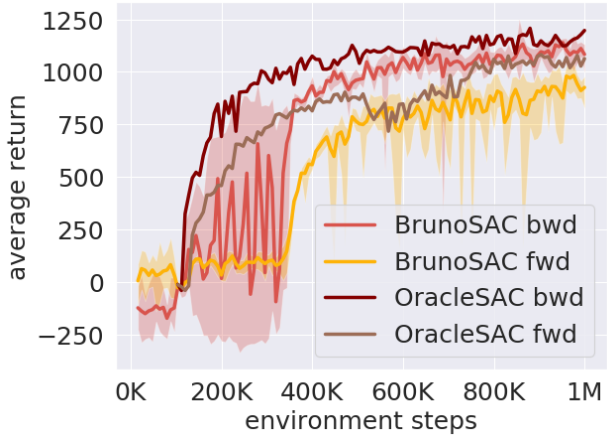


Figure 5.5 Cheetah-Dir test returns versus the number of environment interactions during training. Returns are averaged over 10 trajectories per task. Shaded areas represent minimum and maximum returns of BrunoSAC. Each trajectory is 200 steps long. Returns from the two tasks are plotted separately to illustrate the asymmetry between learning how to run forward and backward.

Figures 5.7 and 5.8 plot the rewards obtained by our trained models when we roll out policies on the test tasks. Similarly to VariBAD, but unlike PEARL, our model requires little observations to infer what the task is and to adapt its behaviour accordingly.

One finding we would like to highlight is that we get good performance regardless of whether we condition the policy on the posterior mean and variance or the mean alone. The redundance of variance indicates that cheetah benchmarks are not suited for exploring the role of uncertainty over tasks. We also admit that the variance we have in our model is inadequate since it does not depend on the data. In future, Student-t processes should be used instead of GPs.

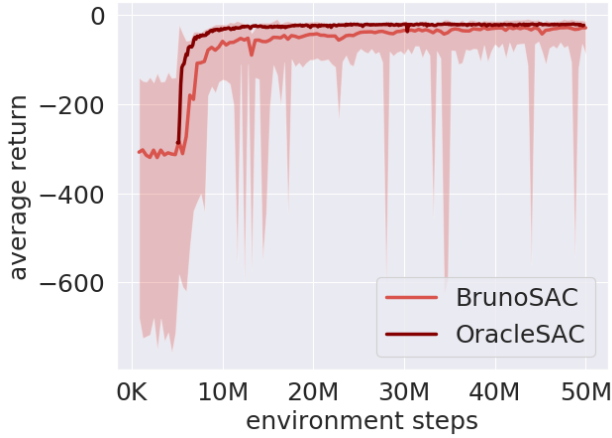


Figure 5.6 Evolution of the Cheetah-Vel test returns during training. The returns are averaged over 150 trajectories (5 trajectories per test task) with each trajectory having 200 steps. Shaded areas represent minimum and maximum returns of BrunoSAC.

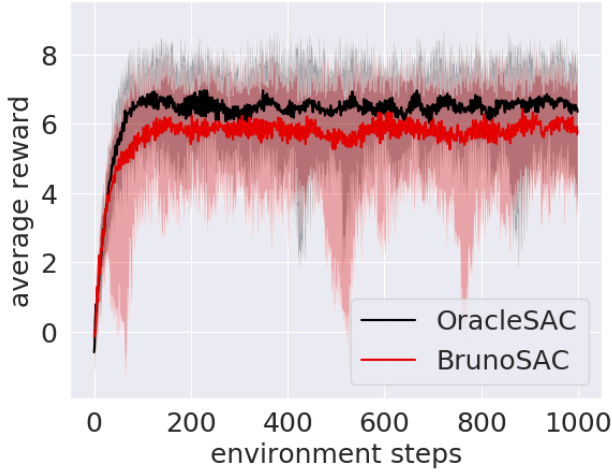


Figure 5.7 Cheetah-Dir test rewards for every environment step. Rewards are averaged over 20 trajectories (10 per task). Shaded regions represent minimum and maximum rewards. Average returns of BrunoSAC and the OracleSAC are 5645 and 6319 respectively. Our results are incomparable to those of VariBAD or PEARL, where the maximum return is ∼2000. This is likely due to using different versions of RL toolkits.

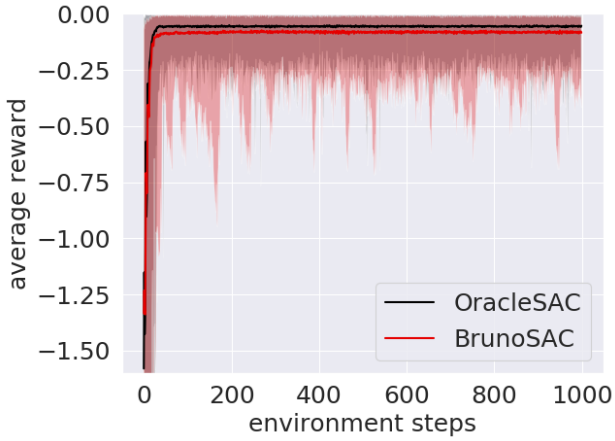


Figure 5.8 Cheetah-Vel average rewards for each step. Rewards are averaged over 300 trajectories (10 per test task). Average returns of BrunoSAC and the OracleSAC are -93 and -66 respectively.

5.6 Discussion and conclusion

We presented a meta-RL method that relies on an exchangeable C-BRUNO architecture suitably repurposed for doing task inference. The latter ability stems from trying to model the reward and transition functions of multiple MDPs. This also makes C-BRUNO appropriate for model-based RL in meta-learning settings, similar to how GPs or neural processes are used [103, 36]. For instance, one could use planning algorithms based on predictive probabilities of rewards and next states given by C-BRUNO. In this work, however, we focused on a different approach in which both C-BRUNO and the policy are trained during the meta-training stage, and no changes are made at test time.

Our BrunoSAC advocates in favour of exchangeable architectures for task inference in meta-RL. Sometimes, they are seen as being more restrictive compared to recurrent neural networks [140]. However, this a sensible restriction since it directly encodes the basic property of the processes we wish to model, as we have shown in section 5.3. The preliminary results of BrunoSAC on the two common benchmarks support our claim, though we admit that any definite conclusions can only be made after an

extensive ablation study. We also anticipate that NPs, as another representative of exchangeable models, would be a successful replacement of C-BRUNO in our overall method. While NPs were already used in PEARL, we think that PEARL made a number of suboptimal choices and thus, in some aspects, was outperformed by VariBAD – a method that does not exploit exchangeability. A further investigation into the behaviour of these models and factors that determine their performance is left for future research.

An interesting application of BrunoSAC could be sim2real: a domain adaptation task addressing the simulation to reality gap [51]. In this problem, we want to find a policy to apply to a plant, for instance, a robot, if we only have an imperfect model of the plant available in the simulation. It is a common issue that policies which work well on the simulator, fail to perform on the plant because of the imperfections in the simulator. In this case, the meta-RL setup is useful: we train BrunoSAC on the distribution of simulators and then test its policy on the plant, where the policy then autonomously narrows down its beliefs over $\boldsymbol{\theta}$ and adjusts its control accordingly.

From the sim2real perspective, we are sceptical that common meta-RL benchmarks give an indication of how performant the algorithms might be in reality. For instance, Cheetah-Dir and Cheetah-Vel use the same transition function $\mathcal{T}$, while only changing the reward function $\mathcal{R}$ across MDPs. This the less useful case for sim2real, where usually $\mathcal{T}$ varies and $\mathcal{R}$ is fixed. Moreover, the meta-RL problem with different reward functions is substantially simpler compared to when the MDP's state transition graph defined by $\mathcal{T}$ changes from one task to another. Only in the latter case, the set of possible trajectories in the environment changes.

BrunoSAC follows the approach of VariBAD and Belief, which use the BAMDPs theory to justify their methods. However, we are uncertain if the theoretical results extend to what is implemented in practice where the inference and acting modules are separated. Namely, BAMDPs assume there exists a Bayes-optimal policy that optimizes the expected return in an MDP whose original states are extended with the belief states. In meta-RL, we train on a number of tasks and, given a sufficiently powerful model, we can find an optimal policy for each of them. But what happens when test tasks

are different from the ones we trained on? The theory does not help us here, so we can only hope that the policy makes good use of the unfamiliar belief states that it sees at test time. How far such a policy is from the optimal is a difficult question to answer. This problem does not appear to be illustrated appropriately in the commonly used locomotion benchmarks.

There is another argument against the approach of separating inference and acting modules as BrunoSAC, VariBAD, PEARL and Belief do. While task inference is based on Bayesian principles, the policy that uses outputs of the Bayesian model to select actions is implemented as a black-box neural network. In many cases, we cannot guarantee a neural network to behave reasonably when presented with far-out inputs. This leads to a natural question whether the policy itself can be made Bayesian while encapsulating task inference procedures in a principled way. We consider this to be an interesting future line of research, where exchangeable models might play an important role.

6

Conclusions and Future Prospects

6.1 Conclusions

In this thesis, we devised a class of BRUNO models that combines some appealing properties of nonparametric Bayesian methods and deep neural networks in a way that is useful in meta-learning tasks with complex high-dimensional observations. In a present state of meta-learning research that is dominated by VAE-based and MAML-based methods, BRUNO provides an alternative approach that was inspired by the fundamental principles behind exchangeability. While exchangeability is often the weakest assumption one could make, it is also, in many cases, the only reliable piece of knowledge that we might have about the data distribution. Explicitly learning to correlate data in order to enforce exchangeability led us to the construction of BRUNO.

In Chapter 3, we described the architecture of BRUNO for modelling exchangeable sequences of random variables and applied it to common few-shot image generation and classification tasks with rather successful results. On a simple example, we also provided a proof of concept for the set anomaly detection. For many models, we analysed their interpretable parameters and gained insights into the behaviour of GPs and TPs. As a part of our research into BRUNO, we obtained some theoretical results such as the recurrent Bayesian updates of the parameters in compound symmetry GPs and TPs, and a more efficient method

of sampling from a Student-t distribution. Both results might be useful additions to the statistical toolkit.

In Chapter 4, we presented C-BRUNO – a version of BRUNO for modelling conditional distributions over the exchangeable sequences. This was accomplished with a minor change to the architecture of the original BRUNO that did not give in any of its valuable properties. The resulting model proved competitive to some of the state-of-the-art methods with more complex designs, and we believe its performance can be further improved by using better normalizing flows, which is an active area of research.

In Chapter 5, we looked at how C-BRUNO can be used for task inference in meta-RL, where the assumption of conditional exchangeability is justified. We constructed BrunoSAC – a meta-RL agent that uses C-BRUNO for inference and SAC for control. In this way, BrunoSAC inherited the appealing features of both components. Namely, from the RL perspective, our method was shown to be sample efficient and fast to train, while from the viewpoint of meta-learning, it could adapt to test tasks given only a handful of observations.

Our models allow for several formulations and interpretations. For example, in Section 3.4, we described BRUNO as a latent variable model. To connect with a greater body of literature, let us outline three other possible perspectives on BRUNO: as a set model, an RNN and a stochastic process.

From a non-Bayesian perspective of other models such as Deep Sets [137], BRUNO's mechanism of enforcing permutation invariance can be viewed as a form of using sum-decomposable functions defined as $f(\boldsymbol{x}_{1:n}) = \rho(\sum_{i=1}^{n} \phi(\boldsymbol{x}_i))$, where ρ and ϕ are some suitable transformations. In BRUNO, the summation happens when we compute parameters of the posterior predictive distribution in GPs or TPs. The form of ϕ and ρ is specified by the stochastic process and the Real NVP mapping. Formulating BRUNO in this way allows to analyse it using a wider set of theoretical tools. For example, Wagstaff et al. [129] showed that having a latent space at least as large as the number of input elements is both necessary and sufficient for representing any continuous function on a set. While in practice we might be driven

by performance measures and not the desire to approximate arbitrary complex functions, these recent theoretical results are still valuable for understanding the limitations of functions on sets and for guiding our design choices when building such models.

Like other RNNs, BRUNO can be viewed as a memory model. At present, the most used RNN architecture is an LSTM [52]. The LSTM keeps its memory in a form of a state – a vector of numbers whose values are updated or read out using gated operations such as read, write and forget. The state of an LSTM can capture long-term dependencies between elements of the input sequences in an order sensitive manner. BRUNO, on the other hand, has an order-invariant memory in the form of parameters of the predictive posterior distribution. Moreover, BRUNO does not have a forgetting mechanism, which would break down the exchangeability. Indeed, if we examine the recurrent updates of BRUNO, we would see that the Bayesian posterior can store information about all the inputs indefinitely.

By construction, BRUNO is a compound model that combines stochastic processes with normalizing flows. Yet in itself, the whole BRUNO is a stochastic process. Unlike with GPs or TPs, here we do not have a simple form for the predictive posterior $p(\boldsymbol{x}_{n+1}|\boldsymbol{x}_{1:n})$, however, we can still evaluate this density at the inputs and sample from it. While we designed our models with high-dimensional inputs settings in mind, when $\boldsymbol{x}$ is low-dimensional, BRUNO becomes more similar to a warped GP [112].

There might exist a number of other ways to analyse, interpret and gain a deeper understanding of the current BRUNO models. Frankly, we are still surprised by the quality of BRUNO's samples considering that its architecture is fairly simple compared to other models built for the same tasks. It is also exciting to contemplate the ways of extending the ideas behind BRUNO and how it would fit into a greater picture of meta-learning. The next section presents our thoughts on these topics.

6.2 Future Prospects

Throughout the thesis, we mentioned a few apparent future lines of research which we will now reiterate. Firstly, the idea we started with, i.e. the exchangeability via traditional RNNs, was not directly tackled in this thesis, and so it remains an open problem to build a more general and possibly a truly black-box exchangeable recurrent model for implicit Bayesian inference in complex data modelling scenarios. Secondly, from a theoretical standpoint, it would be valuable to explore the truly conditional version of de Finetti's theorem for the case of bivariate processes. Thirdly, BRUNO's two-parts architecture allows for a number of variations in both components, so below we will present a few rough ideas on this topic. Afterwards, we will take a step back and speculate about the future of meta-learning research.

Extension of BRUNO to discrete variables

Most of the flow-based models are designed for continuous data, so they require inputs dequantization when dealing with discrete variables and cannot readily yield categorical latents. For these reasons, several discrete flow models have been recently proposed [54, 88, 122]. One example is the Integer Discrete Flow (IDF). It defines a bijective mapping between integer vectors from the input space $\mathcal{X} = \mathbb{Z}^D$ and the latent space $\mathcal{Z} = \mathbb{Z}^D$. IDFs use additive coupling layers with only the translation term and thus no Jacobian in the change of variables formula, i.e. $p(\boldsymbol{x}) = p(\boldsymbol{z})$. The prior $p(\boldsymbol{z})$ is picked to be well-suited for ordinal data: a factored discretized logistic distribution or a mixture of such distributions. These design choices are something to consider when integrating IDFs or other discrete flow models into BRUNO. However, in our opinion, the main undertaking would be to construct a fitting exchangeable random process that would not compromise the computational properties of BRUNO.

More general frameworks for updating belief distributions

In the context of Bayesian neural networks (BNN)s it has been frequently observed that Bayesian posteriors are not the ones performing best. Often, the successful heuristics used in BNNs do not follow the Bayesian paradigms, thus casting doubts on the effectiveness of Bayesian posteriors in complex models and our current understanding of their workings [132]. Below we will briefly explain the idea behind BNNs and the existing alternatives to Bayesian posteriors.

In a classical supervised setting, DNNs are trained to minimize the negative log-likelihood of the labels possibly with an added regularization term $\Omega(\boldsymbol{\phi})$ over DNN's parameters $\boldsymbol{\phi}$. Namely, given a dataset D of N labelled examples, we are interested in finding a single value of parameters that approximately minimize the loss $\mathcal{L}(\boldsymbol{\phi}) = -\frac{1}{n}\sum_{i=1}^{N} \log p(y_i|x_i, \boldsymbol{\phi}) + \Omega(\boldsymbol{\phi})$. In contrast, BNNs wish to find a posterior distribution over the parameters: $p(\boldsymbol{\phi}|D)$. Then predicting labels for new observations will amount to approximating the predictive posterior: $p(y|x, D) = \int p(y|x, \boldsymbol{\phi})p(\boldsymbol{\phi}|D)d\boldsymbol{\phi}$.

In a series of experiments, Wenzel et al. [132] showed that a posterior $p(\boldsymbol{\phi}|D) \propto \exp(-U(\boldsymbol{\phi})/T)$ with an energy function $U(\boldsymbol{\phi}) = -\sum_{i=1}^{N} \log p(y_i|x_i, \boldsymbol{\phi}) - p(\boldsymbol{\phi})$ and a temperature T, has a better predictive performance at lower temperatures, when $T < 1$ or $T \ll 1$. This "cold posterior" is a sharper distribution in comparison to the true Bayesian posterior with $T = 1$. One can also think of a cold posterior as being equivalent to the Bayesian posterior that is computed based on a dataset of $1/T$ replicas of each original observation and a prior $p(\boldsymbol{\phi})^{\frac{1}{T}}$. At present, the question of why the cold posteriors work better in practice largely remains unanswered.

On a related note, Bissiri et al. [10] argue against the use of the traditional likelihood function for some challenging applications. Instead, they propose a more general framework where a prior belief is updated to a posterior via a loss function that connects parameters and observations. When this loss is the negative log-likelihood, we recover standard Bayesian updates. However, under their framework, it is also permissible to have the log-

likelihood scaled with $1/T$ or choose other loss functions that might better serve the end goal of the model.

In connection to BRUNO, it would be interesting to see how these general update rules can be integrated into our model. While they should improve the performance, would the computations of the predictive posterior remain as simple? Would this model still be justified from the perspective of some general de Finetti's theorem? Would such theorem exist at all? These and many other questions are yet to be explored.

Bayesian policies in meta-RL

At the end of Chapter 5, we asked a question whether Bayesian models can be used to define policies of the RL agents. Let us speculate on how BRUNO-like models might help us here. In BrunoSAC, our main assumption was the exchangeability of $(r, \boldsymbol{s}')$ conditionally on $(\boldsymbol{s}, \boldsymbol{a})$, and thus, with C-BRUNO we modelled the predictive distribution $p(r_{t+1}, \boldsymbol{s}'_{t+1} | \boldsymbol{s}_{1:t+1}, \boldsymbol{a}_{1:t+1}, r_{1:t}, \boldsymbol{s}'_{1:t})$ for $t \geq 0$. If we now wish to predict actions as well, we need to assume conditional exchangeability of $(\boldsymbol{a}, r, \boldsymbol{s}')$ given $\boldsymbol{s}$. We could then model $p(\boldsymbol{a}_{t+1}, r_{t+1}, \boldsymbol{s}'_{t+1} | \boldsymbol{s}_{1:t+1}, \boldsymbol{a}_{1:t}, r_{1:t}, \boldsymbol{s}'_{1:t})$ using C-BRUNO. Note that this distribution naturally factorizes into a policy and a model of the MDP:

$$p(\boldsymbol{a}_{t+1}, r_{t+1}, \boldsymbol{s}'_{t+1} | \boldsymbol{s}_{1:t+1}, \boldsymbol{a}_{1:t}, r_{1:t}, \boldsymbol{s}'_{1:t}) = \\ p(\boldsymbol{a}_{t+1} | \boldsymbol{s}_{1:t+1}, \boldsymbol{a}_{1:t}, r_{1:t}, \boldsymbol{s}'_{1:t})\, p(r_{t+1}, \boldsymbol{s}'_{t+1} | \boldsymbol{s}_{1:t+1}, \boldsymbol{a}_{1:t+1}, r_{1:t}, \boldsymbol{s}'_{1:t}).$$

Our experiments with BrunoSAC showed that modelling of $p(r_{t+1}, \boldsymbol{s}'_{t+1} | \boldsymbol{s}_{1:t+1}, \boldsymbol{a}_{1:t+1}, r_{1:t}, \boldsymbol{s}'_{1:t})$ is straightforward. Here, the tricky part is the policy $p(\boldsymbol{a}_{t+1} | \boldsymbol{s}_{1:t+1}, \boldsymbol{a}_{1:t}, r_{1:t}, \boldsymbol{s}'_{1:t})$. Its dependence on the history of previous transitions that include r and $\boldsymbol{s}'$ does not easily fit into the current BRUNO framework. Namely, with C-BRUNO we could only model $p(\boldsymbol{a}_{t+1} | \boldsymbol{s}_{1:t+1}, \boldsymbol{a}_{1:t})$. It leads to the conclusion that both model and policy parts need to be trained jointly, such that there is a flow of information about r and $\boldsymbol{s}'$ from one to the other. We leave the question of how this could be done for a future research.

Meta-learning

The ideas of meta-learning in various forms have been around since the 80s [107], however, only recently it started gaining a considerable amount of attention from the research community. This is due to the realization of a wide gap between how learning works in these models and in humans. Namely, given our prior experiences and our abilities for learning, we can master a new computer game or recognize a new type of bird just after playing the game or seeing the bird a couple of times. Having models that could do so too is an important milestone towards AGI. This "learning to learn" ability is what meta-learning is trying to achieve.

In our opinion, the past years in the meta-learning field were a time of picking low-hanging fruit, for instance, on the problem of few-shot classification. Even on the Omniglot dataset new splits of the training and testing alphabets were adopted so to make the task easier [73]. Moreover, many models are built with a single task in mind, which makes it difficult to generalize their principles across all the concept learning tasks. Finally, much of the knowledge that a researcher has about the task and the dataset gets encoded into the model during the process of its design. Doing so is typically beneficial when dealing with classical DNNs that aim at one application. However, we think that one should exercise caution when working in meta-learning settings, where the main purpose is to learn priors that can be quickly adjusted even when confronted with examples way out of the training distribution. For instance, in case of Omniglot, the model is sure to receive a grayscale image of some character at test time, and it might be utterly confused if the input is blank or has a drawing of a swoose.

On a related issue, improving DNN components of meta-learning models leads to better performance scores, but these gains are misleading when trying to evaluate the true function of these algorithms: their ability to adapt. Disentangling the progress in deep learning and meta-learning is thus becoming a hopeless task, and we cannot talk about advancing the field if we do not know where we stand. We have to admit that our models fell into

similar traps. At present, we think that the best way to proceed is to establish a set of tasks, datasets and perhaps DNN architectures so as to limit the influence of our choices on the aspects of models that are not directly involved in the process of learning to learn.

On the bright side, there is a lot of interest in meta-learning, and inevitably, it is paving the way for new challenges and approaches. While BRUNO models are not likely to play an important role in the grand scheme of things in meta-learning, we are quite certain that exchangeability is here to stay. It is a very basic property, which often goes unmentioned among the modelling assumptions. Nevertheless, it can serve as a starting point for inventing new types of models, and BRUNO is a tiny scratch on that surface.

A

Appendix

A.1 Proofs and derivations

A.1.1 Lemmas

Lemma. ***[1]*** *Given an exchangeable sequence $x_1, x_2, \ldots$ of random variables $x_i \in \mathcal{X}$ and a bijective mapping $f : \mathcal{X} \mapsto \mathcal{Z}$, the sequence $f(x_1), f(x_2), \ldots$ is exchangeable.*

Proof. Consider a vector function $\boldsymbol{g} : \mathbb{R}^n \mapsto \mathbb{R}^n$ such that $(x_1, \ldots, x_n) \mapsto (z_1 = f(x_1), \ldots, z_n = f(x_n))$. A change of variable formula gives:

$$p(x_1, x_2, \ldots, x_n) = p(z_1, z_2, \ldots, z_n) \left|\det \boldsymbol{J}\right|,$$

where $\det \boldsymbol{J} = \prod_{i=1}^{n} \frac{\partial f(x_i)}{\partial x_i}$ is the determinant of the Jacobian of $\boldsymbol{g}$. Since both the joint probability of $(x_1, x_2, \ldots, x_n)$ and the $|\det \boldsymbol{J}|$ are invariant to the permutation of sequence entries, so must be $p(z_1, z_2, \ldots, z_n)$. This proves that $z_1, z_2, \ldots$ is exchangeable. □

Lemma. ***[2]*** *Given two exchangeable sequence $\boldsymbol{x} = x_1, x_2, \ldots$ and $\boldsymbol{y} = y_1, y_2, \ldots$ of random variables, where x_i is independent from y_j for $\forall i, j$, the concatenated sequence $\boldsymbol{x}^\frown\boldsymbol{y} = (x_1, y_1), (x_2, y_2), \ldots$ is exchangeable as well.*

Proof. For any permutation π, as both sequences $\boldsymbol{x}$ and $\boldsymbol{y}$ are exchangeable we have:

$$\begin{aligned}&p(x_1, x_2, \ldots, x_n)p(y_1, y_2, \ldots, y_n)\\&= p(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})p(y_{\pi(1)}, y_{\pi(2)}, \ldots, y_{\pi(n)}).\end{aligned}$$

Independence between elements in $\boldsymbol{x}$ and $\boldsymbol{y}$ allows to write it as a joint distribution:

$$\begin{aligned}&p((x_1, y_1), (x_2, y_2) \ldots, (x_n, y_n)) =\\&p((x_{\pi(1)}, y_{\pi(1)}), (x_{\pi(2)}, y_{\pi(2)}), \ldots, (x_{\pi(n)}, y_{\pi(n)})),\end{aligned}$$

and thus the sequence $\boldsymbol{x}^\frown\boldsymbol{y}$ is exchangeable. □

This lemma justifies our construction with independent exchangeable processes in the latent space.

A.1.2 Derivation of recurrent updates for posterior predictive parameters in CS TPs and GPs

We assume that $\boldsymbol{x} = (x_1, x_2, \ldots x_n) \in \mathbb{R}^n$ follows a multivariate Student-t distribution $MVT_n(\nu, \boldsymbol{\mu}, \boldsymbol{K})$ with degrees of freedom $\nu \in \mathbb{R}_+ \backslash [0, 2]$, mean $\boldsymbol{\mu} \in \mathbb{R}^n$ and a positive definite $n \times n$ covariance matrix $\boldsymbol{K}$. Its density is given by:

$$\begin{aligned}p(\boldsymbol{x}) = &\frac{\Gamma(\frac{\nu+n}{2})}{((\nu-2)\pi)^{n/2}\Gamma(\nu/2)}|\boldsymbol{K}|^{-1/2}\\&\times\left(1 + \frac{(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{K}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}{\nu-2}\right)^{-\frac{\nu+n}{2}}.\end{aligned}$$

Note that this parameterization of the multivariate t-distribution as defined by Shah et al. [109] is slightly different from the commonly used one. We used this parametrization as it makes formulas simpler.

If we partition $\boldsymbol{x}$ into two consecutive parts $\boldsymbol{x}_a \in \mathbb{R}^{n_a}$ and $\boldsymbol{x}_b \in \mathbb{R}^{n_b}$:

$$\begin{bmatrix}\boldsymbol{x}_a\\\boldsymbol{x}_b\end{bmatrix} \sim MVT_n\left(\nu, \begin{bmatrix}\boldsymbol{\mu}_a\\\boldsymbol{\mu}_b\end{bmatrix}, \begin{bmatrix}\boldsymbol{K}_{aa} & \boldsymbol{K}_{ab}\\\boldsymbol{K}_{ba} & \boldsymbol{K}_{bb}\end{bmatrix}\right),$$

the conditional distribution $p(\boldsymbol{x}_b|\boldsymbol{x}_a)$ is given by [109]:

$$p(\boldsymbol{x}_b|\boldsymbol{x}_a) = MVT_{n_b}(\nu + n_a, \tilde{\boldsymbol{\mu}}_{\boldsymbol{b}}, \frac{\nu + \beta_a - 2}{\nu + n_a - 2}\tilde{\boldsymbol{K}}_{bb}), \tag{A.1}$$

where

$$\begin{aligned} \tilde{\boldsymbol{\mu}}_{\boldsymbol{b}} &= \boldsymbol{K}_{ba}\boldsymbol{K}_{aa}^{-1}(\boldsymbol{x}_a - \boldsymbol{\mu}_a) + \boldsymbol{\mu}_b \\ \beta_a &= (\boldsymbol{x}_a - \boldsymbol{\mu}_a)^T\boldsymbol{K}_{aa}^{-1}(\boldsymbol{x}_a - \boldsymbol{\mu}_a) \\ \tilde{\boldsymbol{K}}_{bb} &= \boldsymbol{K}_{bb} - \boldsymbol{K}_{ba}\boldsymbol{K}_{aa}^{-1}\boldsymbol{K}_{ab}. \end{aligned} \tag{A.2}$$

Let us now simplify these equations for the case of exchangeable sequences with $\boldsymbol{\mu} = (\mu, \mu \dots \mu)$ and the following covariance structure:

$$\boldsymbol{K} = \begin{pmatrix} v & \rho & \cdots & \rho \\ \rho & v & \cdots & \rho \\ \vdots & \vdots & \ddots & \vdots \\ \rho & \rho & \cdots & v \end{pmatrix}.$$

In our problem, we are interested in doing one-step predictions, i.e. computing a univariate density $p(x_{n+1}|x_{1:n})$ with parameters ν_{n+1}, μ_{n+1}, v_{n+1}. Therefore, in Eq. A.1 we can take: $n_b = 1$, $n_a = n$, $\boldsymbol{x}_a = x_{1:n} \in \mathbb{R}^n$, $\boldsymbol{x}_b = x_{n+1} \in \mathbb{R}$, $\boldsymbol{K}_{aa} = \boldsymbol{K}_{1:n,1:n}$, $\boldsymbol{K}_{ab} = \boldsymbol{K}_{1:n,n+1}$, $\boldsymbol{K}_{ba} = \boldsymbol{K}_{n+1,1:n}$ and $\boldsymbol{K}_{bb} = \boldsymbol{K}_{n+1,n+1} = v$.

Computing parameters of the predictive distribution requires the inverse of $\boldsymbol{K}_{aa}$, which we can find using the Sherman-Morrison formula:

$$\boldsymbol{K}_{aa}^{-1} = (\boldsymbol{A} + \boldsymbol{u}\boldsymbol{v}^T)^{-1} = \boldsymbol{A}^{-1} - \frac{\boldsymbol{A}^{-1}\boldsymbol{u}\boldsymbol{v}^T\boldsymbol{A}^{-1}}{1 + \boldsymbol{v}^T\boldsymbol{A}^{-1}\boldsymbol{u}},$$

with

$$\boldsymbol{A} = \begin{pmatrix} v - \rho & 0 & \cdots & 0 \\ 0 & v - \rho & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & v - \rho \end{pmatrix},$$

$$\boldsymbol{u} = \begin{pmatrix} \rho \\ \rho \\ \vdots \\ \rho \end{pmatrix}, \boldsymbol{v} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}.$$

After a few steps, the inverse of $\boldsymbol{K}_{aa}$ is:

$$\boldsymbol{K}_{aa}^{-1} = \begin{pmatrix} a_n & b_n & \cdots & b_n \\ b_n & a_n & \cdots & b_n \\ \vdots & \vdots & \ddots & \vdots \\ b_n & b_n & \cdots & a_n \end{pmatrix}$$

with

$$a_n = \frac{v + \rho(n-2)}{(v-\rho)(v+\rho(n-1))},$$
$$b_n = \frac{-\rho}{(v-\rho)(v+\rho(n-1))}.$$

Note that entries of $\boldsymbol{K}_{aa}^{-1}$ explicitly depend on n.

Equations for the mean and variance of the predictive distribution require the following term:

$$\boldsymbol{K}_{ba}\boldsymbol{K}_{aa}^{-1} = \begin{pmatrix} \rho & \rho & \cdots & \rho \end{pmatrix} \boldsymbol{K}_{aa}^{-1} = \left\{ \frac{\rho}{v + \rho(n-1)} \right\}_{1:n},$$

which is a $1 \times n$ vector.

We can now substitute this expression for $\boldsymbol{K}_{ba}\boldsymbol{K}_{aa}^{-1}$ into Eq. A.2 and write down the non-recurrent equations for parameters μ_{n+1} and v_{n+1} of the posterior predictive distribution:

$$\mu_{n+1} = \frac{\rho}{v + \rho(n-1)} \sum_{i=1}^{n} (x_i - \mu) + \mu$$
$$v_{n+1} = v - \frac{n\rho^2}{v + \rho(n-1)}$$

The two equations above are also valid in the GP case, i.e. when $p(\boldsymbol{x}) = \mathcal{N}_n(\boldsymbol{\mu}, \boldsymbol{K})$ and $p(x_{n+1}|x_{1:n}) = \mathcal{N}(\mu_{n+1}, v_{n+1})$. We could have obtained the same results using conjugate Bayesian analysis with the model discussed in Section 3.4. Let us now derive the recurrent updates for μ_{n+1} and v_{n+1}. In both cases, we will need the equations for the previous step:

$$\mu_n = \frac{\rho}{v + \rho(n-2)} \sum_{i=1}^{n-1} (x_i - \mu) + \mu$$
$$v_n = v - \frac{(n-1)\rho^2}{v + \rho(n-2)}$$

The common part between μ_{n+1} and μ_n is $\sum_{i=1}^{n-1}(x_i - \mu)$, thus we can rewrite μ_{n+1} as follows:

$$\begin{aligned}
\mu_{n+1} &= \frac{\rho}{v + \rho(n-1)} \left(\sum_{i=1}^{n-1} (x_i - \mu) + (x_n - \mu) \right) + \mu \\
&= \frac{\rho}{v + \rho(n-1)} \left(\frac{(\mu_n - \mu)(v + \rho(n-2))}{\rho} + (x_n - \mu) \right) + \mu \\
&= \frac{(\mu_n - \mu)(v + \rho(n-2))}{v + \rho(n-1)} + \frac{\rho}{v + \rho(n-1)} (x_n - \mu) + \mu \\
&= \mu_n (1 - d_n) + d_n x_n.
\end{aligned}$$

In the last line of this derivation, we used the fact that $\frac{v+\rho(n-2)}{v+\rho(n-1)} = 1 - \frac{\rho}{v+\rho(n-1)}$, and denoted $d_n = \frac{\rho}{v+\rho(n-1)}$. Using similar reasoning, we can derive the recurrence for the variance. Firstly, we will rewrite v_n in the following form:

$$\begin{aligned}
v_n &= v - \frac{n\rho^2}{v + \rho(n-1)} \frac{v + \rho(n-2)}{v + \rho(n-1)} + \frac{\rho^2}{v + \rho(n-2)} \\
&= v - \frac{n\rho^2}{v + \rho(n-1)} \frac{1}{1 - d_n} + \frac{\rho^2}{v + \rho(n-2)}
\end{aligned}$$

From this equation, we can express $\frac{n\rho^2}{v+\rho(n-1)}$ as a function of v_n and substitute it into v_{n+1}:

$$\begin{aligned}
v_{n+1} &= v - (v - v_n)(1 - d_n) - \frac{\rho^2}{v + \rho(n-1)} \\
&= v_n (1 - d_n) + d_n (v - \rho).
\end{aligned}$$

For TPs, we also need to derive a recurrent version for $\beta_{n+1} = (\boldsymbol{x}_a - \boldsymbol{\mu}_a)^T K_{aa}^{-1} (\boldsymbol{x}_a - \boldsymbol{\mu}_a)$.

Let $\tilde{\boldsymbol{x}} = \boldsymbol{x}_a - \boldsymbol{\mu}_a$, then:

$$\begin{aligned}
\beta_{n+1} &= \tilde{\boldsymbol{x}}^T K_{aa}^{-1} \tilde{\boldsymbol{x}} \\
&= (a_n \tilde{x}_1 + b_n \sum_{i \neq 1}^{n} \tilde{x}_i, a_n \tilde{x}_2 + b_n \sum_{i \neq 2}^{n} \tilde{x}_i, \ldots, a_n \tilde{x}_n + b_n \sum_{i \neq n} \tilde{x}_i)^T (\tilde{x}_1, \tilde{x}_2, \ldots \tilde{x}_n) \\
&= (a_n - b_n) \sum_{i=1}^{n} \tilde{x}_i^2 + b_n (\sum_{i=1}^{n} \tilde{x}_i)^2.
\end{aligned}$$

Similarly, β_n from $p(x_n | x_{1:n-1})$ is:

$$\beta_n = (a_{n-1} - b_{n-1}) \sum_{i=1}^{n-1} \tilde{x}_i^2 + b_{n-1} (\sum_{i=1}^{n-1} \tilde{x}_i)^2$$

We can now use the fact that $\sum_{i=1}^{n-1} \tilde{x}_i^2$ is a common part between β_{n+1} and β_n:

$$\begin{aligned}
\beta_{n+1} &= (a_n - b_n)(\sum_{i=1}^{n-1} \tilde{x}_i^2 + \tilde{x}_n^2) + b_n (\sum_{i=1}^{n} \tilde{x}_i)^2 \\
&= (a_n - b_n) \frac{\beta_n - b_{n-1} (\sum_{i=1}^{n-1} \tilde{x}_i)^2}{a_{n-1} - b_{n-1}} + (a_n - b_n) \tilde{x}_n^2 + b_n (\sum_{i=1}^{n} \tilde{x}_i)^2.
\end{aligned}$$

Given that $\frac{a_n - b_n}{a_{n-1} - b_{n-1}} = 1$, we write β_{n+1} recursively as:

$$\begin{aligned}
s_{n+1} &= s_n + \tilde{x}_n \\
\beta_{n+1} &= \beta_n + (a_n - b_n) \tilde{x}_n^2 + b_n (s_{n+1}^2 - s_n^2),
\end{aligned}$$

with $s_1 = 0$.

A.1.3 Derivation of the posterior in CS TPs

It is easiest to think about multivariate Student-t distribution as an infinite mixture of multivariate Gaussians where all Gaussians have the same means but their covariance is scaled with a different factor. It has also been shown that the scale parameter r follows an inverse gamma distibution, thus $\boldsymbol{x} \sim MVT_n(\nu, \boldsymbol{\mu}, \boldsymbol{K})$ when

$r \sim$ Inv-Gamma$(\frac{\nu}{2}, \frac{1}{2})$ and $\boldsymbol{x} \sim \mathcal{N}_n(\mu, r(\nu-2)\boldsymbol{K})$ [109, 136]. Let us write it as:

$$\begin{aligned} p(x_1, \ldots, x_n) &= MVT_n(\nu, \boldsymbol{\mu}, \boldsymbol{K}) \\ p(x_1, \ldots, x_n | r) &= \mathcal{N}_n(\boldsymbol{\mu}, r(\nu-2)\boldsymbol{K}) \\ p(r) &= \text{Inv-Gamma}(\frac{\nu}{2}, \frac{1}{2}) \end{aligned}$$

We can represent the multivariate Gaussian in de Finetti's form:

$$p(x_1, \ldots, x_n | r) = \int p(x_i | r, \phi) p(\phi | r) d\phi, \tag{A.3}$$

where given our usual assumptions, i.e. $\boldsymbol{\mu} = (\mu, \ldots \mu)$, $\boldsymbol{K}_{ii} = v$, and $\boldsymbol{K}_{ij, i \neq j} = \rho$, Eq. 3.3 implies that $p(\phi|r) = \mathcal{N}(\mu, r(\nu-2)\rho)$ and $p(x_i|\phi, r) = \mathcal{N}(\phi, r(\nu-2)(v-\rho))$.

If we write the marginal density as:

$$p(x_1, \ldots, x_n) = \int \int p(x_i | r, \phi) p(\phi, r) d\phi dr,$$

it becomes clear that the latent variable in de Finetti's theorem is $\boldsymbol{\theta} = \{\phi, r\}$. Therefore, we wish to find the posterior of ϕ and r given observations $x_{1:n}$. The posterior can be written as $p(\phi, r | x_{1:n}) = p(\phi | r, x_{1:n}) p(r | x_{1:n})$, where both factors are known. The first factor is a posterior from the model in Eq. A.3: $p(\phi | r, x_{1:n}) = \mathcal{N}(\mu_n, r(\nu-2)\sigma_n^2)$ with μ_n and σ_n^2 given in Eq. 3.4. The second one, $p(r|x_{1:n}) =$ Inv-Gamma$(\frac{\nu+n}{2}, \frac{1}{2}(1 + \frac{\beta_n}{\nu-2}))$ as derived by Shah et al. [109].

A.1.4 Derivation of recurrent updates for parameters of the posterior distribution in CS GPs

In Section 3.4 we derived the following equations for parameters of the posterior distribution $p(\theta | z_{1:n}) = \mathcal{N}(\mu_n, \sigma_n^2)$ in CS GPs:

$$\begin{aligned} \mu_n &= \left(\frac{1}{\rho} + \frac{n}{v-\rho} \right)^{-1} \left(\frac{\mu}{\rho} + \frac{\sum_{i=1}^n z_i}{v-\rho} \right) \\ \sigma_n^2 &= \left(\frac{1}{\rho} + \frac{n}{v-\rho} \right)^{-1} \end{aligned}$$

Instead of these closed-form equations, we would like to have the recurrent updates as we did for the posterior predictive distribution. Fortunately, we do not have to derive everything from scratch. Given that our likelihood is Gaussian with variance $v - \rho$, the conjugate Bayesian analysis [86] tells us how to relate parameters of the posterior with parameters of the predictive posterior. Namely, $p(z_{n+1}|z_{1:n}) = \mathcal{N}(\mu_n, \sigma_n^2 + (v - \rho))$. Thus, the means of the two distributions are equal, and their variances differ by a constant. We then immediately arrive to the following result:

$$\mu_{n+1} = (1 - d_t)\mu_n + d_n z_n$$
$$\sigma_{n+1}^2 = (1 - d_n)(\sigma_n^2 - v + \rho),$$

with $d_n = \frac{\rho}{v+\rho(n-1)}$, $\mu_0 = 0$ and $\sigma_0^2 = \rho$.

A.2 Optional materials

A.2.1 BRUNO

Implementation details for BRUNO models

For simple datasets, such as MNIST, we found it tolerable to use models that rely upon a general implementation of the Real NVP coupling layer similarly to Papamakarios et al. [91]. Namely, when scaling and translation functions s and t are fully-connected neural networks. In our model, networks s and t share the parameters in the first two dense layers with 1024 hidden units and ELU nonlinearity [19]. Their output layers are different: s ends with a dense layer with tanh and t ends with a dense layer without a nonlinearity. We stacked 6 coupling layers with alternating the indices of the transformed dimensions between odd and even as described by Dinh et al. [23]. For the first layer, which implements a logit transformation of the inputs, namely $f(x) = \text{logit}(\alpha + (1 - 2\alpha)x)$, we used $\alpha = 10^{-6}$. The logit transformation ensures that when taking the inverse mapping during sample generation, the outputs always lie within $(\frac{-\alpha}{1-2\alpha}, \frac{1-\alpha}{1-2\alpha})$.

In Omniglot, Fashion MNIST and CIFAR-10 experiments, we built upon a Real NVP model originally designed for CIFAR-10 by Dinh

et al. [24]: a multi-scale architecture with deep convolutional residual networks in the coupling layers. Our main difference was the use of coupling layers with fully-connected s and t networks (as described above) placed on top of the original convolutional Real NVP model. We found that adding these layers allowed for faster convergence and improved results. This is likely due to better mixing of the information before the output of the Real NVP gets into the Student-t layer. We also found that using weight normalisation [105] within every s and t function was crucial for successful training of large models.

The model parameters were optimized using RMSProp [120] with a decaying learning rate starting from 10^{-3}. Trainable parameters of a TP or GP were updated with a 10x smaller learning rate and were initialized as following: $\nu^d = 1000$, $v^d = 1.$, $\rho^d = 0.1$ for every dimension d. The mean μ^d was fixed at 0. For the Omniglot model, we used a batch size of 32, the sequence length of 20 and trained for 200K iterations. The other models were trained for a smaller number of iterations, i.e. ranging from 50K to 100K updates.

Parameters analysis

In addition to the analysis of correlations in the Omniglot model from Section 3.5.1, we provide the results for CIFAR-10 and MNIST models in Figure A.1.

For TP-based models, degrees of freedom ν^d were initialized to 1000, thus making TPs close to a GPs. After training, most of the dimensions retain fairly high degrees of freedom, but some can have small ν's. One can notice from Figure A.2 that dimensions with high correlation tend to have smaller degrees of freedom.

We noticed that CS TPs and GPs can behave differently for certain settings of hyperparameters even when TPs have high degrees of freedom. Figure A.3 gives one example when this is the case.

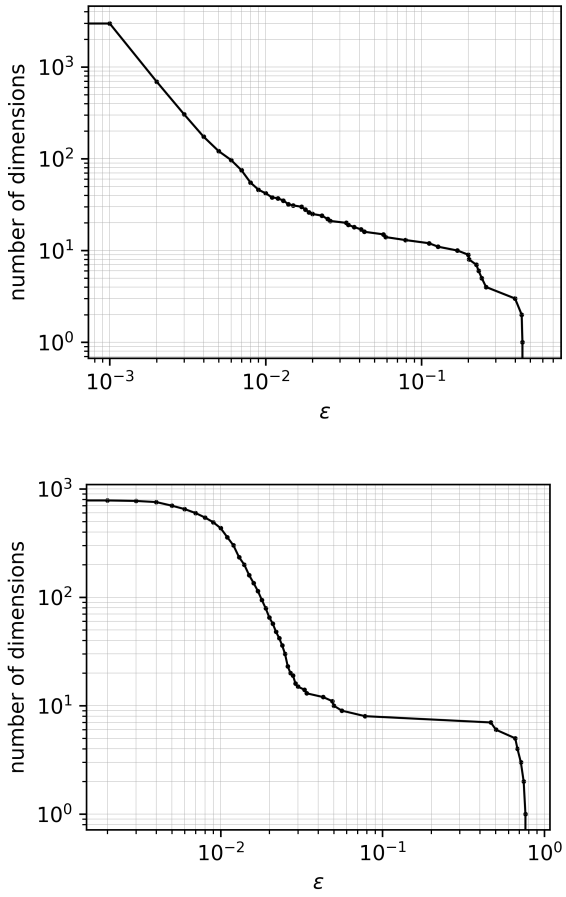


Figure A.1 Number of dimensions where $\rho^d/v^d > \epsilon$ plotted on a double logarithmic scale. *Top*: CIFAR-10 model *Bottom*: Non-convolutional MNIST model.

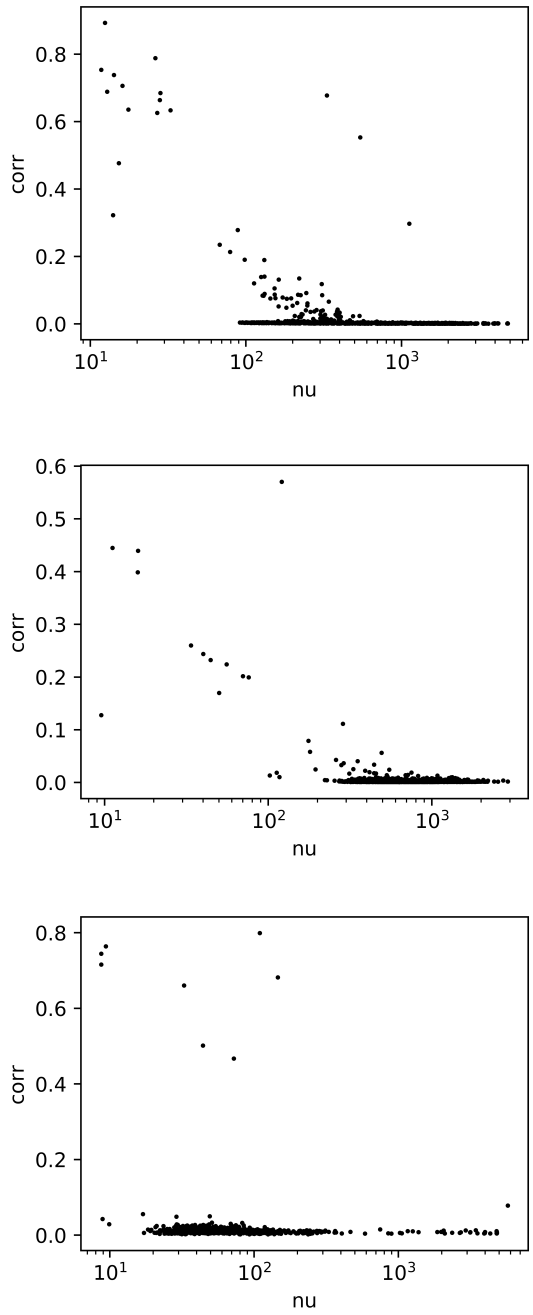


Figure A.2 Correlation ρ^d/v^d versus degrees of freedom ν^d for every d. Degrees of freedom on the x-axis are plotted on a logarithmic scale. *Top*: Omniglot model. *Middle*: CIFAR-10 model *Bottom*: Non-convolutional version of BRUNO trained on MNIST.

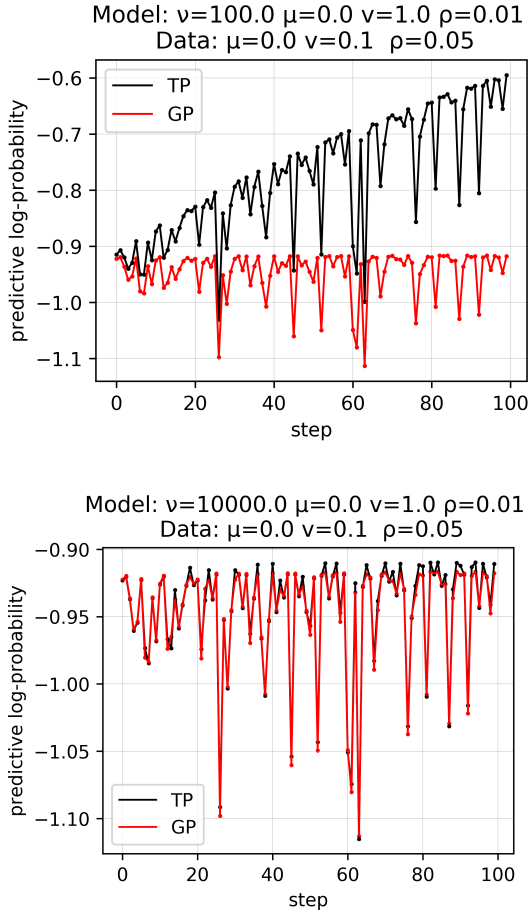


Figure A.3 A toy example which illustrates how degrees of freedom ν affect the behaviour of a TP compared to a GP. Here, we generate one sequence of 100 observations from an multivariate normal disribution with parameters $\mu = 0.$, $v = 0.1$, $\rho = 0.05$ and evaluate predictive probabilities under CS TP and GP models with parameters $\mu = 0.$, $v = 1.$, $\rho = 0.01$ and different ν for TP in the top and bottom plots.

Sampling algorithm for a Student-t distribution

Algorithm 4 Efficient sampling on a GPU from a univariate t-distribution with mean μ, variance v and degrees of freedom ν

$a, b \leftarrow \mathcal{U}(0, 1)$
$c \leftarrow \min(a, b)$
$r \leftarrow \max(a, b)$
$\alpha \leftarrow \frac{2\pi c}{r}$
$t \leftarrow \cos(\alpha)\sqrt{(\nu/r^2)(r^{-4/\nu} - 1)}$
$\sigma \leftarrow \sqrt{v\left(\frac{\nu-2}{\nu}\right)}$
return $\mu + \sigma t$

BRUNO samples

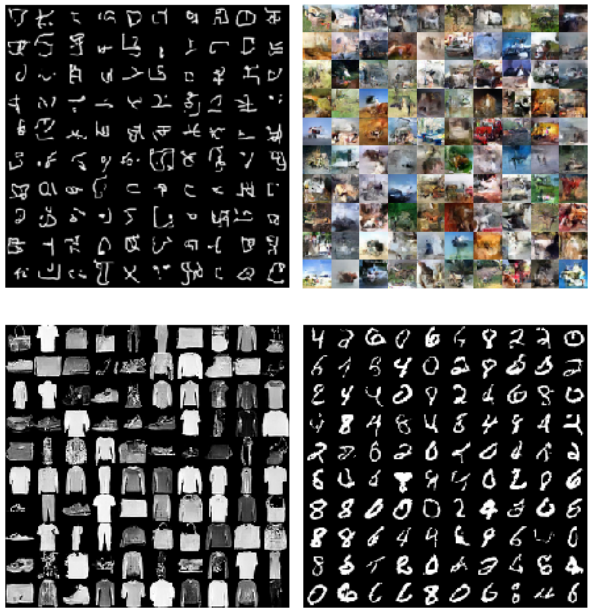


Figure A.4 Samples from the prior for Omniglot, CIFAR-10, Fashion MNIST and MNIST models. CIFAR-10 and Fashion MNIST models were trained on examples from all 10 classes, MNIST was only trained on even digits.

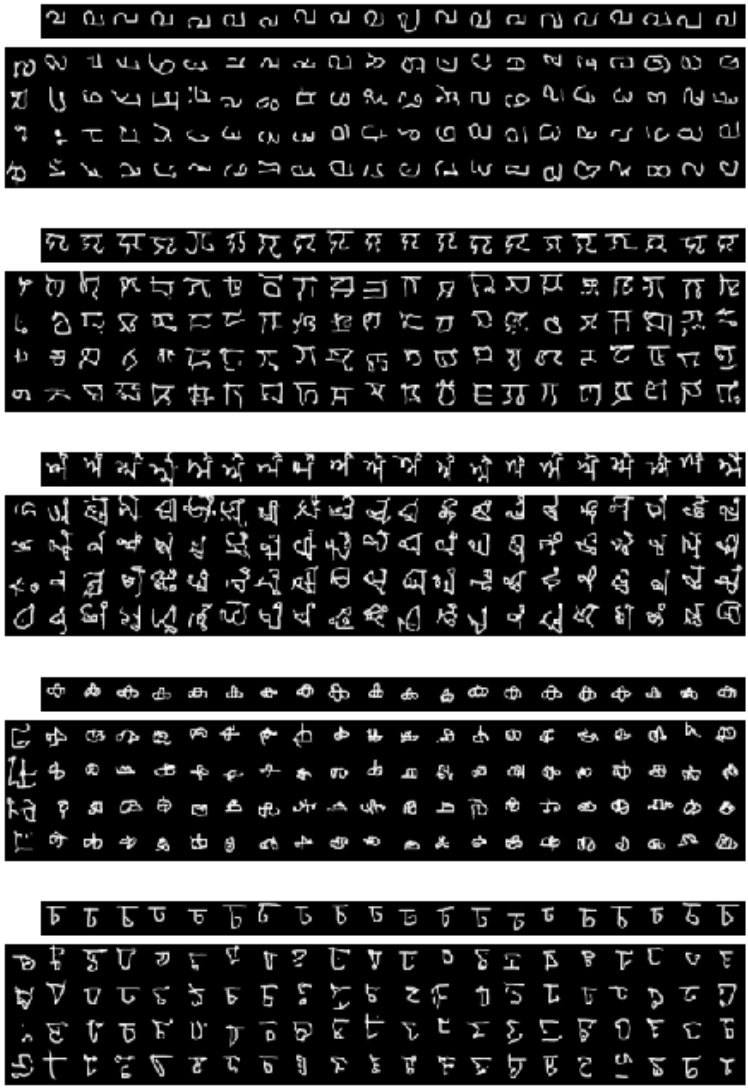


Figure A.5 Samples from a model trained on Omniglot. Conditioning images come from character classes that were not used during training, so when n is small, the problem is equivalent to a few-shot generation.

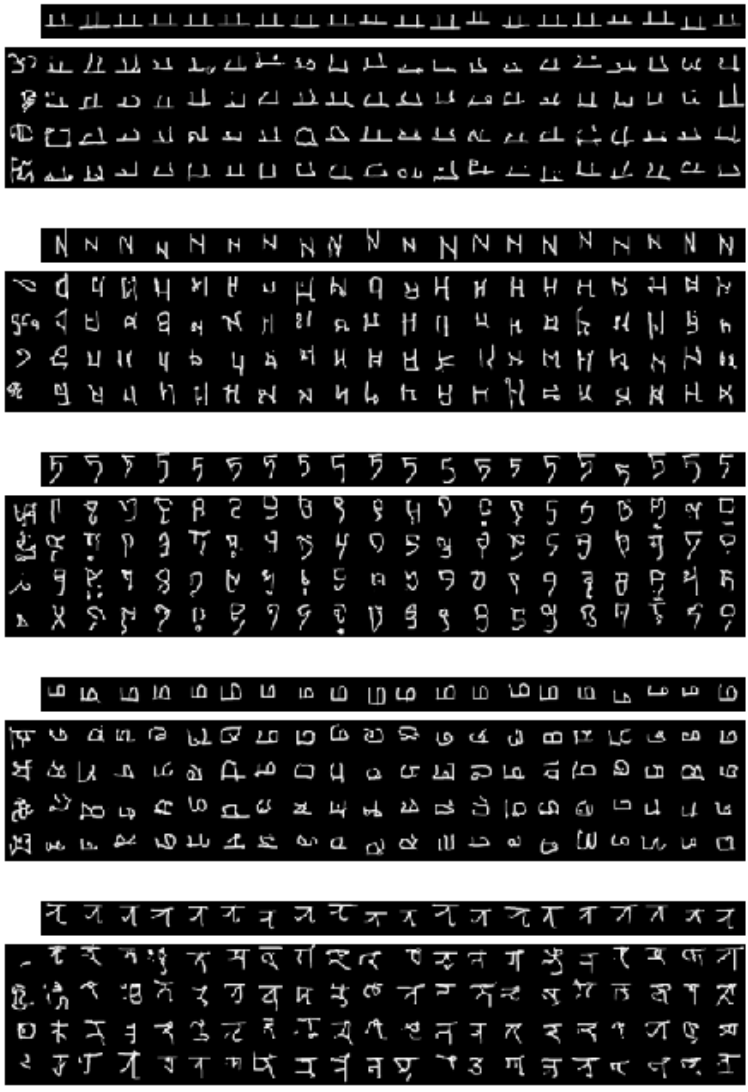


Figure A.6 Continuation of Figure A.5.

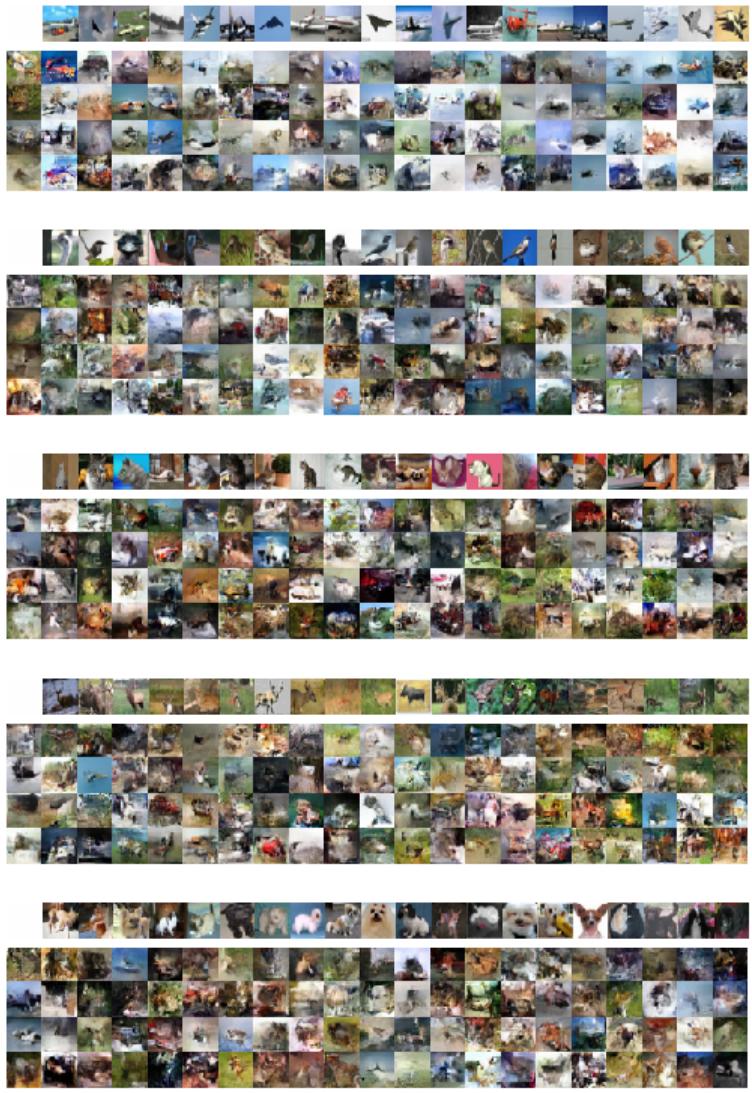


Figure A.7 Samples from a model trained on CIFAR-10. The model was trained on the dataset with 10 classes. Conditioning images in the top row of each subplot come from the test set.

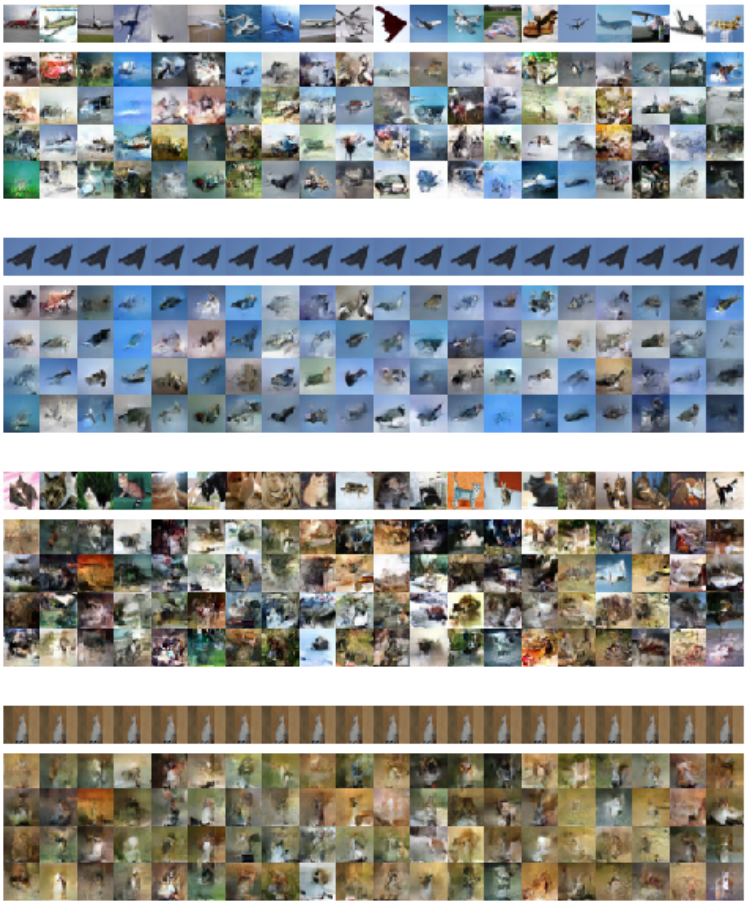


Figure A.8 CIFAR-10 samples from $p(\mathbf{x}|\mathbf{x}_{1:n})$ for every $n = 480, \ldots, 500$. *Left*: input sequence (given in the top row of each subplot) is composed of random same-class test images. *Right*: same image is given as input at every step. In both cases, input images come from the test set of CIFAR-10 and the model was trained on all of the classes.

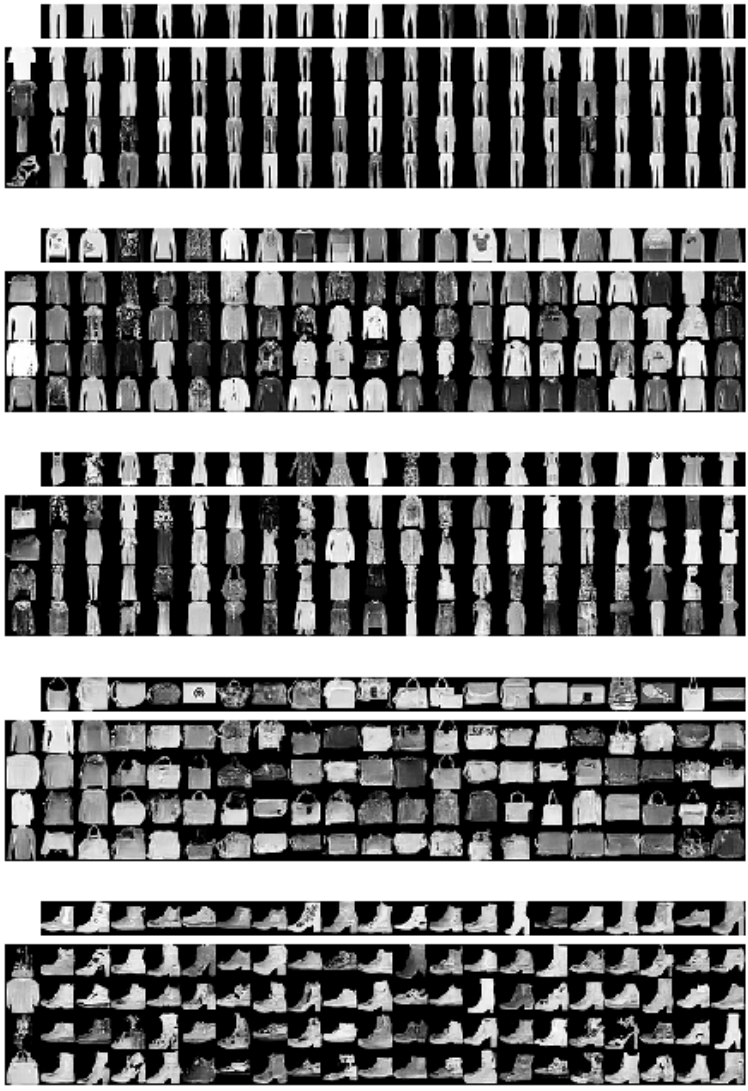


Figure A.9 Samples from a convolutional BRUNO model trained on Fashion MNIST. The model was trained on the set with 10 classes. Conditioning images in the top row of each subplot come from the test set.

A.2.2 C-BRUNO

ShapeNet view generation

In our experiments, we closely follow the setup of Gordon et al. [40]. Namely, we use the same train-test split of 37,108 objects from 12 ShapeNet classes: airplane, bench, cabinet, car, phone, chair, display, lamp, speaker, sofa, table, boat. Each object is available in 36 views spaced 10 degrees in azimuth around the object. This dataset and the code to reproduce Versa's results are provided by the authors at `github.com/Gordonjo/versa`. We train C-BRUNO in an episodic manner with 4 random tasks per batch. In each task, the model is given one random view of an object and it is required to assign likelihoods to other 16 random views of the same object. Below, we provide additional 1-shot samples from the two models.

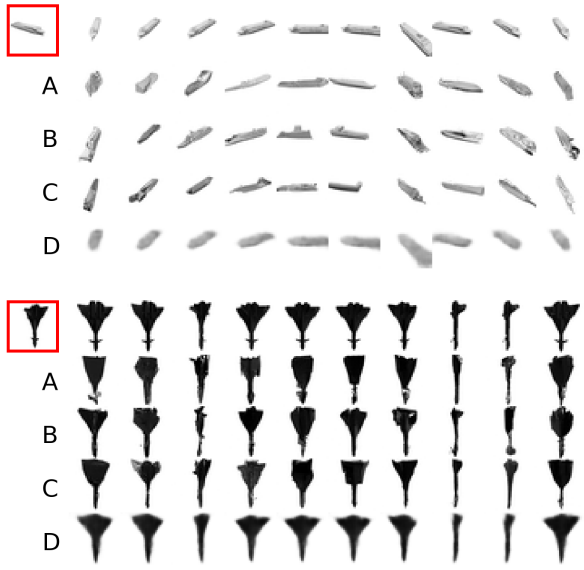


Figure A.10 One-shot C-BRUNO samples in rows A-C and Versa samples in row D for the unseen test objects. Here, we condition on a single view $(\boldsymbol{x}_1, \boldsymbol{h}_1)$ marked in red. The top row of each plot contains ground truth images, whereas the three rows A to C are the C-BRUNO samples from $p(\boldsymbol{x}|\boldsymbol{h}, \boldsymbol{x}_1, \boldsymbol{h}_1)$ conditioned on a different angle $\boldsymbol{h}$ in each column.

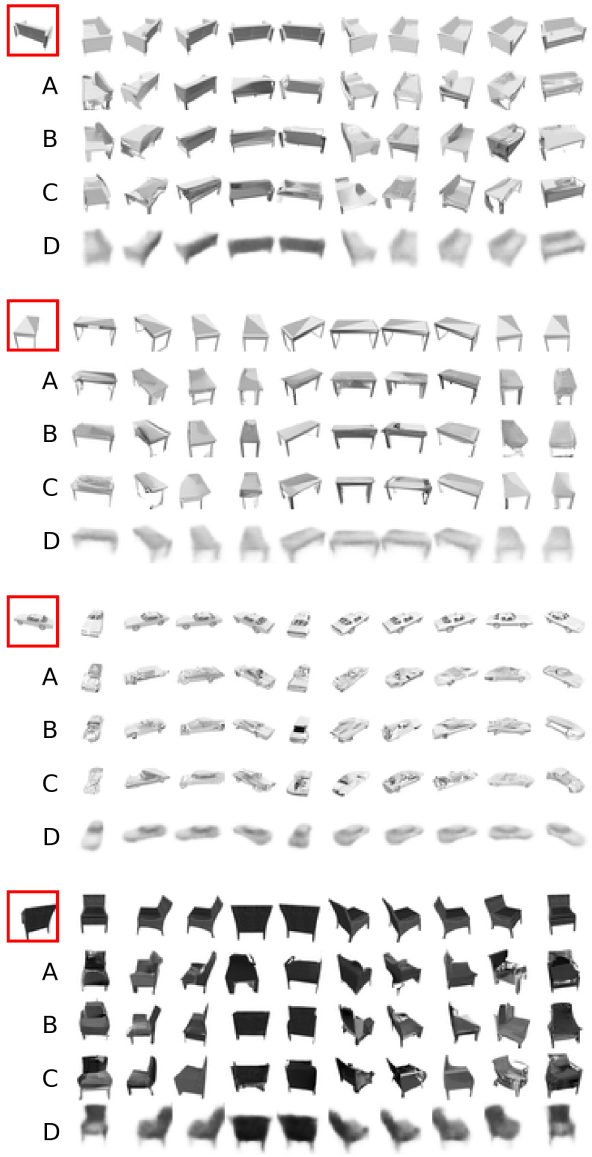


Figure A.11 Continuing Figure A.10.

1D regression

For the toy 1D regression tasks we use C-BRUNO and NPs with a 2-dimensional latent space. The MAF architecture of C-BRUNO consists of 7 autoregressive layers, in which scale and translation networks share 3 hidden layers of 32 units. Two of these layers are responsible for computing the features of $\boldsymbol{h}$. The code for NPs is adapted from `github.com/deepmind/neural-processes`. The encoder network of NPs is 4-layer MLP with 128 units per layer, and the decoder has 2 hidden layers of the same size. We train both models for 100K update steps on batches of 16 sequences. Each sequence is of length 50, and the context is sampled randomly per batch.

A.2.3 BrunoSAC

Model hyperparameters

N_{iter}	2500
N_{updates}	200
N_{batch}	10
N_{pretrain}	250
γ	0.99
$\lambda_V, \lambda_Q, \lambda_{\boldsymbol{\phi}}$	0.001
trajectory length	200
training sequence length	100
context length k	$\sim \mathcal{U}(25, 75)$
MAF architecture	6 autoregressive layers with 2 hidden layers of 128 units (one layer for inputs $\boldsymbol{x}$ and one for the condition $\boldsymbol{h}$), ReLU [87] activation
dimensionality of $\mathcal{Z}$	5
Policy network architecture	2 hidden layers of 128 units with leaky ReLU activation
Value and action-value network achitectures	1 hidden layer of 128 units with leaky ReLU activation

Bibliography

[1] D.J. Aldous et al. *Ecole d'Ete de Probabilites de Saint-Flour XIII, 1983*. Lecture Notes in Mathematics. Springer Berlin Heidelberg, 1985. ISBN: 9783540152033.

[2] R. W. Bailey. "Polar generation of random variates with the t-distribution". In: *Mathematics of Computation* 62.206 (1994), pp. 779–781.

[3] S. Bartunov and D. Vetrov. "Few-shot Generative Modelling with Generative Matching Networks". In: *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics*. Vol. 84. 2018.

[4] S. Bartunov et al. "Meta-Learning Deep Energy-Based Memory Models". In: *International Conference on Learning Representations*. 2020.

[5] C. Bender et al. "Exchangeable Generative Models with Flow Scans." In: *Association for the Advancement of Artificial Intelligence*. 2019.

[6] J.M. Bernardo and A.F.M. Smith. *Bayesian Theory*. Wiley Series in Probability and Statistics. Wiley, 2007.

[7] M. Betancourt. "A Conceptual Introduction to Hamiltonian Monte Carlo". In: *ArXiv* abs/1701.02434 (2017).

[8] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, 2006.

[9] C. M. Bishop and J. Lasserre. “Generative or Discrimative? Getting the Best of Both Worlds”. In: *Bayesian Statistics 8*. International Society for Bayesian Analysis. Oxford University Pres, 2007, pp. 3–24.

[10] P. G. Bissiri, C. C. Holmes, and S. G. Walker. “A general framework for updating belief distributions”. In: *Journal of the Royal Statistical Society. Series B, Statistical methodology* 78.5 (2016), pp. 1103–1130.

[11] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. “Variational Inference: A Review for Statisticians”. In: *ArXiv* abs/1601.00670 (2017).

[12] D. M. Blei, A. Y. Ng, and M. I. Jordan. “Latent Dirichlet Allocation”. In: *Journal of Machine Learning Research* 3 (2003), pp. 993–1022.

[13] M. Botvinick et al. “Reinforcement Learning, Fast and Slow”. In: *Trends in Cognitive Sciences* 23.5 (2019), pp. 408–422.

[14] Y. Burda, R. B. Grosse, and R. Salakhutdinov. “Importance Weighted Autoencoders”. In: *arXiv* abs/1509.00519 (2016).

[15] A. X. Chang et al. “ShapeNet: An Information-Rich 3D Model Repository”. In: *ArXiv* abs/1512.03012 (2015).

[16] J. Chen et al. “VFlow: More Expressive Generative Flows with Variational Data Augmentation”. In: *Proceedings of the 37th International Conference on Machine Learning*. 2020.

[17] Wei-Yu Chen et al. “A Closer Look at Few-shot Classification”. In: *International Conference on Learning Representations*. 2019.

[18] K. Cho et al. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”. In: *Empirical Methods in Natural Language Processing*. 2014.

[19] D. Clevert, T. Unterthiner, and S. Hochreiter. “Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)”. In: *International Conference on Learning Representations*. 2016.

[20] L. Clouâtre and M. Demers. “FIGR: Few-shot Image Generation with Reptile”. In: *ArXiv* abs/1901.02199 (2019).

[21] G. Deco and W. Brauer. “Higher Order Statistical Decorrelation without Information Loss”. In: *Advances in Neural Information Processing Systems 7*. 1995.

[22] J. Deng et al. “ImageNet: A Large-Scale Hierarchical Image Database”. In: *CVPR09*. 2009.

[23] L. Dinh, D. Krueger, and Y. Bengio. “NICE: Non-linear Independent Components Estimation”. In: *arXiv* abs/1410.8516 (2014).

[24] L. Dinh, J. Sohl-Dickstein, and S. Bengio. “Density Estimation Using Real NVP”. In: *International Conference on Learning Representations*. 2017.

[25] Y. Duan et al. “RL2: Fast Reinforcement Learning via Slow Reinforcement Learning”. In: *ArXiv* abs/1611.02779 (2017).

[26] M. Duff and A. Barto. “Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes”. PhD thesis. Univ of Massachusetts at Amherst, 2002.

[27] E. Dupont, A. Doucet, and Yee Whye Teh. “Augmented Neural ODEs”. In: *ArXiv* abs/1904.01681 (2019).

[28] H. Edwards and A. Storkey. “Towards a Neural Statistician”. In: *International Conference on Learning Representations*. 2017.

[29] L. Elliott. *AI Art Gallery. NeurIPS Workshop on Machine Learning for Creativity and Design*. `http://www.aiartonline.com/`. 2019.

[30] S. Ermon and A. Grover. *Deep Generative Models CS236 Lecture notes*. `https://deepgenerativemodels.github.io`. 2019.

[31] S. M. A. Eslami et al. “Neural scene representation and rendering”. In: *Science* 360.6394 (2018), pp. 1204–1210.

[32] B. Eysenbach and S. Levine. “If MaxEnt RL is the Answer, What is the Question?” In: *ArXiv* abs/1910.01913 (2019).

[33] B. de Finetti. “Foresight: Its Logical Laws, Its Subjective Sources”. In: *Breakthroughs in Statistics: Foundations and Basic Theory*. Springer New York, 1992, pp. 134–174.

[34] C. Finn, P. Abbeel, and S. Levine. “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks”. In: *Proceedings of the 34th International Conference on Machine Learning*. 2017.

[35] S. Fortini and S. Petrone. “Predictive construction of priors in Bayesian nonparametrics”. In: *Brazilian Journal of Probability and Statistics* 26.4 (2012), pp. 423–449.

[36] A. Galashov et al. “Meta-Learning surrogate models for sequential decision making”. In: *ArXiv* abs/1903.11907 (2019).

[37] M. Garnelo et al. “Neural Processes”. In: *Theoretical Foundations and Applications of Deep Generative Models, ICML workshop* (2018).

[38] Z. Ghahramani and K. A Heller. “Bayesian Sets”. In: *Advances in Neural Information Processing Systems 18*. 2006.

[39] X. Glorot, A. Bordes, and Y. Bengio. “Deep Sparse Rectifier Neural Networks”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 2011.

[40] J. Gordon et al. “Meta-Learning Probabilistic Inference for Prediction”. In: *International Conference on Learning Representations*. 2019.

[41] W. Grathwohl et al. “Scalable Reversible Generative Models with Free-form Continuous Dynamics”. In: *International Conference on Learning Representations*. 2019.

[42] Klaus Greff et al. “LSTM: A Search Space Odyssey”. In: *IEEE Transactions on Neural Networks and Learning Systems* 28 (2017), pp. 2222–2232.

[43] D. Ha, A. M. Dai, and Q. V. Le. “HyperNetworks”. In: *International Conference on Learning Representations*. 2017.

[44] T. Haarnoja et al. “Latent Space Policies for Hierarchical Reinforcement Learning”. In: *Proceedings of the 35th International Conference on Machine Learning*. 2018.

[45] T. Haarnoja et al. “Soft Actor-Critic Algorithms and Applications”. In: *ArXiv* abs/1812.05905 (2018).

[46] T. Haarnoja et al. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”. In: *Proceedings of the 35th International Conference on Machine Learning*. 2018.

[47] D. Hafner et al. “Learning Latent Dynamics for Planning from Pixels”. In: *Proceedings of the 36th International Conference on Machine Learning*. 2019.

[48] N. Heess et al. “Learning Continuous Control Policies by Stochastic Value Gradients”. In: *Advances in Neural Information Processing Systems 28*. 2015.

[49] K. A. Heller and Z. Ghahramani. “A Simple Bayesian Framework for Content-Based Image Retrieval”. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2006.

[50] L. B. Hewitt et al. “The Variational Homoencoder: Learning to learn high capacity generative models from few examples”. In: *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence*. 2018.

[51] I. Higgins et al. “Darla: Improving zero-shot transfer in reinforcement learning”. In: *Proceedings of the 34th International Conference on Machine Learning*. 2017.

[52] S. Hochreiter and J. Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[53] S. Hochreiter, A. S. Younger, and P. R. Conwell. “Learning to Learn Using Gradient Descent”. In: *Artificial Neural Networks — ICANN 2001*. 2001, pp. 87–94.

[54] E. Hoogeboom et al. “Integer Discrete Flows and Lossless Compression”. In: *Advances in Neural Information Processing Systems 32*. 2019.

[55] B. Hou et al. “Flexible Conditional Image Generation of Missing Data with Learned Mental Maps”. In: *Machine Learning for Medical Image Reconstruction*. 2019, pp. 139–150.

[56] C. Huang, L. Dinh, and A. C. Courville. “Augmented Normalizing Flows: Bridging the Gap Between Generative Flows and Latent Variable Models”. In: *ArXiv* abs/2002.07101 (2020).

[57] G. Huang, H. Larochelle, and S. Lacoste-Julien. “Are Few-shot Learning Benchmarks Too Simple ?” In: *ArXiv* abs/1902.08605 (2019).

[58] J. F. Humplík et al. “Meta reinforcement learning as task inference”. In: *ArXiv* abs/1905.06424 (2019).

[59] F. Huszár. *Exchangeable Models via Recurrent Neural Networks?* `https://www.inference.vc/exchangeable-processes-via-neural-networks/`. Blog. 2015.

[60] A. K. Jain, M. N. Murty, and P. J. Flynn. “Data Clustering: A Review”. In: *ACM Computing Surveys* 31.3 (1999), pp. 264–323.

[61] R. Józefowicz et al. “Exploring the Limits of Language Modeling”. In: *ArXiv* abs/1602.02410 (2016).

[62] T. Karras, S. Laine, and T. Aila. “A Style-Based Generator Architecture for Generative Adversarial Networks”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2018), pp. 4396–4405.

[63] H. Kim et al. “Attentive Neural Processes”. In: *International Conference on Learning Representations*. 2019.

[64] D. P. Kingma and J. Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations*. 2015.

[65] D. P. Kingma and P. Dhariwal. “Glow: Generative Flow with Invertible 1x1 Convolutions”. In: *Advances in Neural Information Processing Systems 31*. 2018.

[66] D. P. Kingma and M. Welling. “Auto-Encoding Variational Bayes”. In: *International Conference on Learning Representations*. 2014.

[67] D. P. Kingma et al. “Improved Variational Inference with Inverse Autoregressive Flow”. In: *Advances in Neural Information Processing Systems 29*. 2016.

[68] I. Korshunova et al. “Discriminative Topic Modeling with Logistic LDA”. In: *Advances in Neural Information Processing Systems 32*. 2019.

[69] A. Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

[70] A. Krizhevsky, I. Sutskever, and G. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. 2012.

[71] A. Kumar et al. "Consistent Generative Query Networks". In: *ArXiv* abs/1807.02033 (2018).

[72] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. "Human-level concept learning through probabilistic program induction". In: *Science* (2015).

[73] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. "The Omniglot challenge: a 3-year progress report". In: *Current Opinion in Behavioral Sciences* 29 (2019), pp. 97–104.

[74] B. M. Lake et al. "Building Machines That Learn and Think Like People". In: *The Behavioral and brain sciences* 40 (2016).

[75] A. Lapidoth. *A Foundation in Digital Communication*. 2nd ed. Cambridge University Press, 2017.

[76] S. Lauritzen. *Exchangeability and de Finetti's Theorem*. Lecture notes. 2007.

[77] Y. LeCun, Y. Bengio, and G. Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[78] Y. LeCun, C. Cortes, and C. JC Burges. *The MNIST database of handwritten digits*. 1998.

[79] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE*. 1998, pp. 2278–2324.

[80] K. Li and J. Malik. "Learning to Optimize". In: *ArXiv* abs/1606.01885 (2016).

[81] Y. Li. "Approximate Inference: New Visions". PhD thesis. University of Cambridge, 2018.

[82] P. McCullagh. "Exchangeability and regression models". In: *Celebrating Statistics*. Oxford: Oxford University Press, 2005.

[83] S. Mohamed et al. "Monte Carlo Gradient Estimation in Machine Learning". In: *ArXiv* abs/1906.10652 (2019).

[84] A. Mordvintsev, C. Olah, and M. Tyka. *Inceptionism: Going Deeper into Neural Networks*. 2015. URL: https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html.

[85] T. Munkhdalai and H. Yu. "Meta Networks". In: *Proceedings of the 34th International Conference on Machine Learning*. 2017.

[86] K. P. Murphy. *Conjugate bayesian analysis of the gaussian distribution*. Tech. rep. 2007.

[87] V. Nair and G. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. 2010.

[88] D. Nielsen and O. Winther. "Closing the Dequantization Gap: PixelCNN as a Single-Layer Flow". In: *ArXiv* abs/2002.02547 (2020).

[89] B. O'Neill. "Exchangeability, Correlation, and Bayes' Effect". In: *International Statistical Review* 77.2 (2009), pp. 241–250.

[90] B. Øksendal. *Stochastic Differential Equations: An Introduction with Applications*. Hochschultext / Universitext. Springer, 2003.

[91] G. Papamakarios, T. Pavlakou, and I. Murray. "Masked Autoregressive Flow for Density Estimation". In: *Advances in Neural Information Processing Systems 30*. 2017.

[92] G. Papamakarios et al. "Normalizing Flows for Probabilistic Modeling and Inference". In: *ArXiv* abs/1912.02762 (2019).

[93] G. I. Parisi et al. "Continual lifelong learning with neural networks: A review". In: *Neural Networks* 113 (2019), pp. 54–71.

[94] K. Rakelly et al. "Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables". In: *Proceedings of the 36th International Conference on Machine Learning*. 2019.

[95] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. ISBN: 026218253X.

[96] S. Reed et al. “Few-shot Autoregressive Density Estimation: Towards Learning to Learn Distributions”. In: *International Conference on Learning Representations*. 2018.

[97] D. J. Rezende, S. Mohamed, and D. Wierstra. “Stochastic Backpropagation and Approximate Inference in Deep Generative Models”. In: *Proceedings of the 31st International Conference on Machine Learning*. 2014.

[98] D. J. Rezende et al. “Normalizing Flows on Tori and Spheres”. In: *ArXiv* abs/2002.02428 (2020).

[99] D. Rezende et al. “One-Shot Generalization in Deep Generative Models”. In: *Proceedings of The 33rd International Conference on Machine Learning*. 2016.

[100] J. Rothfuss et al. “ProMP: Proximal Meta-Policy Search”. In: *International Conference on Learning Representations*. 2019.

[101] S. Ruder. “An Overview of Multi-Task Learning in Deep Neural Networks”. In: *ArXiv* abs/1706.05098 (2017).

[102] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323 (1986).

[103] S. Sæmundsson, K. Hofmann, and M. P. Deisenroth. “Meta Reinforcement Learning with Latent Variable Gaussian Processes”. In: *Conference on Uncertainty in Artificial Intelligence (UAI)*. 2018.

[104] H. Sak et al. “Fast and accurate recurrent neural network acoustic models for speech recognition”. In: *ArXiv* abs/1507.06947 (2015).

[105] T. Salimans and D. P. Kingma. “Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks”. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. 2016.

[106] A. Santoro et al. “Meta-Learning with Memory-Augmented Neural Networks”. In: *Proceedings of The 33rd International Conference on Machine Learning*. 2016.

[107] J. Schmidhuber. “Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook”. PhD thesis. Inst. f. Inf., Tech. Univ. Munich, 1987.

[108] J. Schulman et al. “Proximal Policy Optimization Algorithms”. In: *ArXiv* abs/1707.06347 (2017).

[109] A. Shah, A. G. Wilson, and Z. Ghahramani. “Student-t Processes as Alternatives to Gaussian Processes”. In: *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics*. 2014.

[110] I. Simon and S. Oore. *Performance RNN: Generating Music with Expressive Timing and Dynamics*. `https://magenta.tensorflow.org/performance-rnn`. Blog. 2017.

[111] J. Snell, K. Swersky, and R. Zemel. “Prototypical Networks for Few-shot Learning”. In: *Advances in Neural Information Processing Systems 30*. 2017.

[112] E. Snelson, Z. Ghahramani, and C. E. Rasmussen. “Warped Gaussian Processes”. In: *Advances in Neural Information Processing Systems 16*. 2004, pp. 337–344.

[113] K. Sohn, H. Lee, and X. Yan. “Learning Structured Output Representation using Deep Conditional Generative Models”. In: *Advances in Neural Information Processing Systems 28*. 2015.

[114] B. Stadie et al. “The Importance of Sampling in Meta-Reinforcement Learning”. In: *Advances in Neural Information Processing Systems 31*. 2018.

[115] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018.

[116] Z. Szabo et al. “Learning Theory for Distribution Regression”. In: *Journal of Machine Learning Research* 17.152 (2016).

[117] E. G. Tabak and Cristina V. Turner. “A Family of Nonparametric Density Estimation Algorithms”. In: *Communications on Pure and Applied Mathematics* 66.2 (2013), pp. 145–164.

[118] L. Theis, A. van den Oord, and M. Bethge. “A note on the evaluation of generative models”. In: *International Conference on Learning Representations*. 2016.

[119] S. Thrun and L. Pratt, eds. *Learning to Learn*. USA: Kluwer Academic Publishers, 1998. ISBN: 0792380479.

[120] T. Tieleman and G. Hinton. *Lecture 6.5 - RmsProp: Divide the gradient by a running average of its recent magnitude*. Coursera: Neural Networks for Machine Learning. 2012.

[121] E. Todorov, T. Erez, and Y. Tassa. "MuJoCo: A physics engine for model-based control." In: *IROS*. 2012.

[122] D. Tran et al. "Discrete Flows: Invertible Generative Models of Discrete Data". In: *Advances in Neural Information Processing Systems 32*. 2019.

[123] E. Triantafillou et al. "Meta-Dataset: A Dataset of Datasets for Learning to Learn from Few Examples". In: *International Conference on Learning Representations*. 2020.

[124] B. Uria, I. Murray, and H. Larochelle. "RNADE: The real-valued neural autoregressive density-estimator". In: *Advances in Neural Information Processing Systems 26*. 2013.

[125] L. Van Der Maaten, E Postma, and J. Van den Herik. "Dimensionality reduction: a comparative review". In: *Journal of Machine Learning Research* 10 (2009), pp. 66–71.

[126] O. Vinyals. *Contrasting Model- and Optimization-based Metalearning*. `http://metalearning-symposium.ml/files/vinyals.pdf`. Talk. 2017.

[127] O. Vinyals, S. Bengio, and M. Kudlur. "Order matters: Sequence to sequence for sets". In: *International Conference on Learning Representations*. 2016.

[128] O. Vinyals et al. "Matching Networks for One Shot Learning". In: *Advances in Neural Information Processing Systems 29*. 2016.

[129] E. Wagstaff et al. "On the Limitations of Representing Functions on Sets". In: *ArXiv* abs/1901.09006 (2019).

[130] J. X. Wang et al. "Learning to reinforcement learn". In: *ArXiv* abs/1611.05763 (2016).

[131] K. Weiss, T. M. Khoshgoftaar, and D. Wang. "A survey of transfer learning". In: *Journal of Big Data* 3.1 (2016).

[132] F. Wenzel et al. "How Good is the Bayes Posterior in Deep Neural Networks Really?" In: *ArXiv* abs/2002.02405 (2020).

[133] D. Wertheimer and B. Hariharan. “Few-Shot Learning With Localization in Realistic Settings”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019).

[134] H. Xiao, K. Rasul, and R. Vollgraf. “Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms”. In: *ArXiv* abs/1708.07747 (2017).

[135] M. Yang et al. “Energy-Based Processes for Exchangeable Data”. In: *Proceedings of the 37th International Conference on Machine Learning*. 2020.

[136] S. Yu, V. Tresp, and K. Yu. “Robust Multi-Task Learning with t-Processes”. In: *Proceedings of the 24th International Conference on Machine Learning*. 2007.

[137] M. Zaheer et al. “Deep Sets”. In: *Advances in Neural Information Processing Systems 30*. 2017.

[138] A. Zhou et al. “Watch, Try, Learn: Meta-Learning from Demonstrations and Rewards”. In: *International Conference on Learning Representations*. 2020.

[139] B. D. Ziebart et al. “Maximum Entropy Inverse Reinforcement Learning”. In: *Proceedings of the 23rd National Conference on Artificial Intelligence*. 2008.

[140] L. Zintgraf et al. “VariBAD: A Very Good Method for Bayes-Adaptive Deep RL via Meta-Learning”. In: *International Conference on Learning Representations*. 2020.