On the Structure of Learning and Transfer in Machines

Janith Petangoda
Department of Computing
Imperial College London

a dissertation submitted for the degree of
Doctor of Philosophy

November 2022

This page was left intentionally blank.

Declaration

This dissertation and all research contained in it are the product of my original work unless stated otherwise.

I certify that, to the best of my knowledge, I have acknowledged the work, ideas and outcomes of others explicitly via citations, or in the main text. I have done my best to adhere to standard practices of academic integrity and discipline.

This page was left intentionally blank.

Copyright

This page was left intentionally blank.

Acknowledgments

I begin by thanking my examiners, David and Daniel. I know that this thesis was a dense read; I thank you for your time and insight. Your thoughts gave me confidence to believe that this work could, with some extra work, be practically useful.

Thank you to the members of Do Not Circulate; Kate, Arinbjörn, Daniel and Isak for the casual yet somehow intense discussions that eventually ended in a workshop paper.

Chatura, thank you listening to my (often) incoherent ramblings. These helped me consolidate what I understood as I tried to learn what was needed for this thesis.

Marc, thank you for allowing me to pursue the unusual direction that I took with this work, and for your eternal patience with me as I erratically tried different directions and failed to do anything in time. Thank you also for dealing with my horrendous writing, and teaching me how to write clearly and concisely. I am also grateful for the funding that you found me. It really helped me and my family during the previous 4 years, and gave me the peace of mind to focus on my work.

Nick, your friendship during the last 9 years has been an inspiration. Our chats were instrumental to the ideas that were written here. You showed me the value in my romantic thinking, and inspired me to pursue the joy in research and learning. Thank you. Also, I swear we will finish that paper soon!

Amma and Thaththi, I am eternally grateful for your support and love during this PhD. I wouldn't have been able to do it without your guidance.

Laura, your patience during the restless nights, and my bouts of stress was unimaginable. For this I am in your debt.

This page was left intentionally blank.

Abstract

Machine Learning (ML) is often described as a process of learning about patterns and relationships. Structure is an example of the relationship between spaces of *things*; in this work, we provide a definition of learning in machines written as the process of learning unknown structure. This produces a unified view of ML that affords us concrete notions of spaces of tasks, and how they relate to chosen models.

Using such a view, we define what it means to transfer between ML problems, and how to learn to transfer. Our definition embodies the notion that transfer is tightly coupled with biases, in that to transfer is to assume biases. Further, we define transfer in the same language of structure as we did vanilla learning; the key difference manifests as the structure that is learnt. This definition highlights differences between learning *to* transfer, and learning *by* transfer.

We provide a framework, based on the theory of foliations that expresses our notions of transfer in the context of structure. We express popular methods of transfer in ML using our framework, and discuss how our framework informs us about the benefits of transfer.

The primary goal of this thesis is to introduce the mathematical and philosophical frameworks by which learning and transfer in machines can be expressed and interpreted consistently in terms of structure.

This page was left intentionally blank.

Table of Contents

This page was left intentionally blank.

The chances of finding out what's really going on in the universe are so remote, the only thing to do is hang the sense of it and keep yourself occupied.

— Douglas Adams
A Hitchhiker's Guide to the Galaxy

Chapter 1

Introduction

As individuals capable of *general* learning, we intuitively know that *transfer* plays an important and necessary role in the process of learning. Instead of having to learn every minute detail necessary to live our lives, we are able to re-use those details that are shared. This is clearly a useful ability to have. Imagine if for every new terrain of ground we encounter, we would have to re-learn how to control our muscles; the number of trials needed would mean we would spend our lives doing little else.

Transfer is useful

Consider now that we needed to do the same intellectually. If we learn how to carry out addition over some set of numbers, we would need to re-learn how to carry out the same operation over any other, unseen combination of numbers. Instead, we are capable of *transferring* that knowledge and applying it to new numbers we had not computed addition over. We are able to apply the same set of rules to a new situation.

Transfer compresses the costs of learning, storing and representing *knowledge*, new or otherwise. If what we want to learn, store or represent is related to something new, transfer allows us to only learn, store or represent the *differences* between them, without wasting resources on what is already known, stored or represented.

Brief history of transfer in ML

Given such benefits of transfer, it comes as no surprise that we would like to enable learning machines to reap such benefits. Work in the field progressed through suggestions for lifelong learning [109, 110], multi-task learning [14, 94], and learning internal representations of biases, and inductive bias transfer [6, 7]. An idea of meta-learning, or learning to learn, was initially written by Schmidhuber in [89, 90]. More modern approaches that have built on these works include advances in deep learning based meta-learning [115, 43], few-shot learning [119], deep multitask learning [82] and neural architecture search [27]. In these, transfer allows us to learn using less data, computational resources, and time by leveraging relevant information. In the case of multitask learning, it can also lead to better generalisation [14].

The missing link

While the literature on transfer is rich with algorithmic and experimental contributions, it lacks a foundational framework that guides and unites the

exposition. The literature is vague about what it means for tasks to be related, and this relationship corresponds to transfer. It also writes it as a separate procedure to the learning of single tasks, rather than another learning procedure applied on a *different* kind of problem. This is partly due to the inextensibility of existing definitions of learning to accommodate transfer consistently. In this thesis, we aim to provide a framework that fills these gaps.

1.1 Contributions

Summary of the framework

The contributions of this thesis can be summarised as a framework that provides:

1. a definition of learning written in terms of *structure* (Part 1),
2. a generally applicable, philosophical definition of transfer that emphasises the structure of transfer (Chapter 6),
3. the application of our definition of transfer to the context of learning, written as the learning of the structure associated with transfer (Chapters 7 and 8), and
4. a distinction between *learning to transfer* and *learning by transfer* (Section 6.4) .

Through this framework, we show that learning, be it vanilla learning, or learning using transfer, is a process of identifying structure. The key differences between these can be characterised by the structure that is being identified. In learning related to transfer, the structure is often related to a particular space of problems that is carried out by some vanilla learning. Our framework unifies these points of views, and expresses their differences in terms of a hierarchy of structure. In other words, learning to transfer is the *same* as learning, but applied at a *higher* level.

Submitted papers

Contributions in the form of submitted papers are summarised below, in chronological order:

Disentangled Skill Embeddings for Reinforcement Learning [76]: We developed a novel multi-task reinforcement learning method that exploited known structure about how tasks would be sampled. Tasks varied according to their transition dynamics and reward functions; our methods used knowledge about the qualitative structure of the variations that made up the training tasks. The present author's contributions (DSE-REINFORCE) are summarised in Section 8.4.

GENNI: Visualising the geometry of equivalences for neural network identifiability [59]: Symmetries of the input-output map of a neural network can be important in understanding the optimisation that occurs during learning over the parameters of a network. However, understanding and accounting

for the symmetries of a network can be very difficult, even for the simplest of architectures. Section 3.4 summaries a technique we developed to help visualise potential symmetries near network models.

Learning to Transfer: A Foliated Theory [75]: This paper provides a summary of the initial results of this thesis, excluding those from the previous two contributions. It introduces the idea that a formal framework for transfer is needed, and that if done properly, we could combine techniques of some major fields of machine learning. In particular, we provide a group theoretic definition for transfer from first principles. This thesis expands on the ideas presented in this paper in greater detail.

1.2 Roadmap

The primary goal of this thesis is to introduce a complete and principled view of transfer in general, and apply this to transfer in the context of learning. Our definition casts transfer as something that can occur whenever *properties* can be shared between descriptions *objects*. For example, there is a sense in which transfer can occur between the knowledge of a cat and an owl (they are both animals), the storage of images, or the dynamics of different pendulums.

Components needed for Transfer

To define transfer in this way, we must identify some core components. The first of these is a set of objects. Here, *objects* can be anything, be it abstract notions, images, dynamical properties or others. We require that there exists a set of properties which can be used to uniquely identify each thing within the set of objects. These could, for example be the color and size of cats and owls, the RGB values of each pixel of an image, or the mass and length of a pendulum. Such properties are what can be transferred.

We can define a notion of transfer on this set of objects. A notion of transfer defines sets of related objects, which are defined such that transfer can occur between elements of these sets. Objects are related by their membership in such sets, and their relationships facilitate transfer. Intuitively, we know that transfer occurs between *related* objects; a notion of transfer captures this relationship. The exact transformation between objects is described by a notion of relatedness that can be defined on, or derived from a notion of transfer.

Components of Transfer in Learning

In the context of learning, we identify these components through the definitions of a task and spaces of tasks. A task defines what it is we want to learn. It is described as relationships (which we call structure maps) between chosen relata. These structure maps are the properties of tasks that can be used to define a notion of transfer. A space of tasks contains well defined variations of the structure maps of a task. Alternatively, we can first define a space of tasks as variations of structure maps, and a task is a particular instance of these variations. The space of tasks is the set of objects on which transfer can be defined for a learning problem.

In order to carry out machine learning however, we also require notions of models, loss functions, and learning algorithms. Together, these form learning problems and their solution. In a broad sense, we will describe machine learning as the process of finding descriptions of unknown structure. The unknown structure is defined on a learning task The model space is the language we can describe this unknown structure. A loss function characterises the suitability of each model in this space for representing the unknown structure of the task. A learning algorithm finds the most suitable model, and therefore description.

When machine learning is defined this way, it allows us to unify the problem of learning to transfer (and any other *higher order learning*), with what we typically think of as single task learning. In the structural view of machine learning, learning is a process of finding a representation that suitably preserves the structure of the learning task. Learning to transfer is a similar process carried out at a higher level. Instead of learning the structure of single learning problem, we want to find a certain structure (a notion of transfer) over a set of problems, thus allowing us to *learn by transfer*.

Summary of Part 1

Part 1 of this thesis looks at developing the formal definitions and theory of all these components for single task machine learning. Chapter 2 introduces structure maps, and how they relate to tasks. Further, in Section 2.6 we show how variations of structure maps can be used to identify a space of tasks. In Chapter 3 we introduce parametric models. We will show how we can identify a model space from a model architecture. We will discuss how symmetries in the input-output maps of a model can be accounted for. Chapter 4 defines the remaining components, which include the loss function, and learning algorithm. Finally, we conclude by unifying these notions in a framework of representations.

Summary of Part 2

Part 2 will be the discussion of transfer. We provide a general description of transfer in Chapter 6, and distinguish between *learning to transfer* and *learning by transfer*. Chapter 7 provides an introduction to foliations. The theory of foliations is used as the mathematical tool for concretely describing transfer on a space of tasks, and consequently on the space of models. We then adapt the definition of loss functions and learning algorithms to enable a mechanism for learning to transfer. In Chapter 8 we recontextualise existing methods in terms of our framework. We end with conclusions and potential future work.

1.3 Map of Definitions

To help the reader to parse this thesis, we provide maps that show the relationship between some definitions and examples are shown in Figures 1.1 - 1.6. These are organised by chapter; connections between chapters are shown by reusing the relevant definitions.

Note that the arrows in these diagrams should be read as 'used in'. For example, in Figure 1.1, *Relata* is used in the definition of *System*.

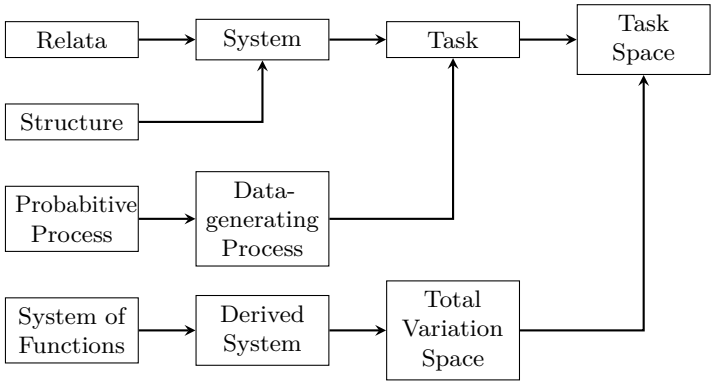


Figure 1.1. Map of Definitions for Chapter 2.

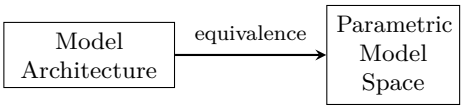


Figure 1.2. Map of Definitions for Chapter 3.

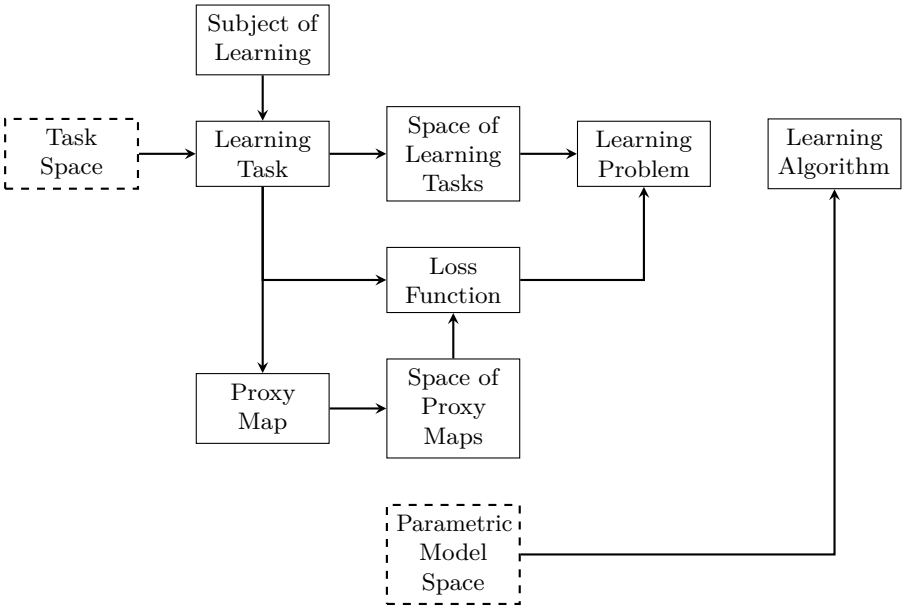


Figure 1.3. Map of Definitions for Chapter 4. Dashed boxes are definitions from other chapters. Relationships between these, if any, are not depicted here.

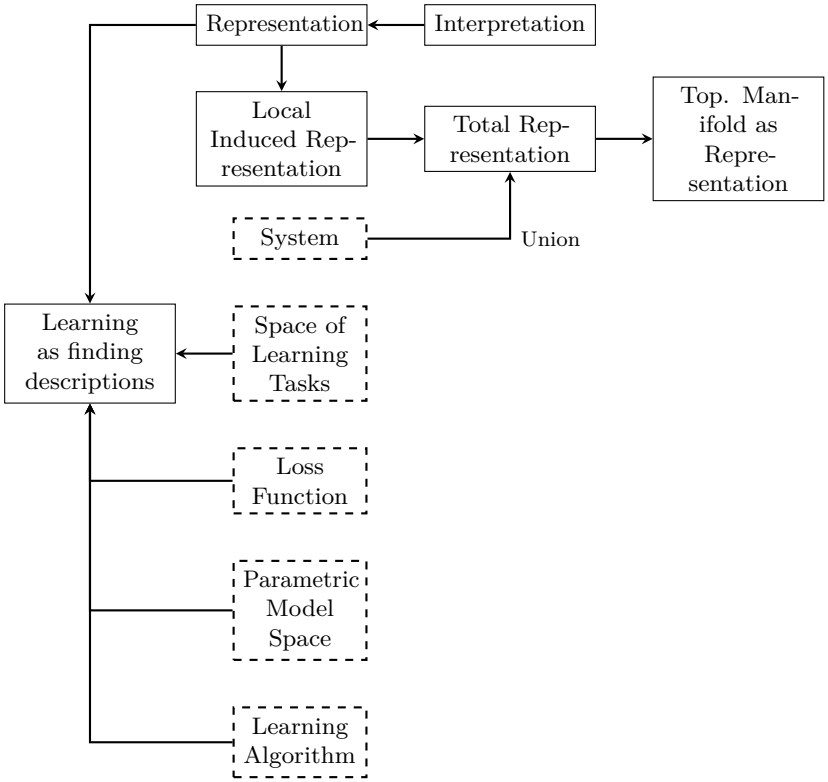


Figure 1.4. Map of Definitions for Chapter 5. Dashed boxes are definitions from other chapters. Relationships between these, if any, are not depicted here.

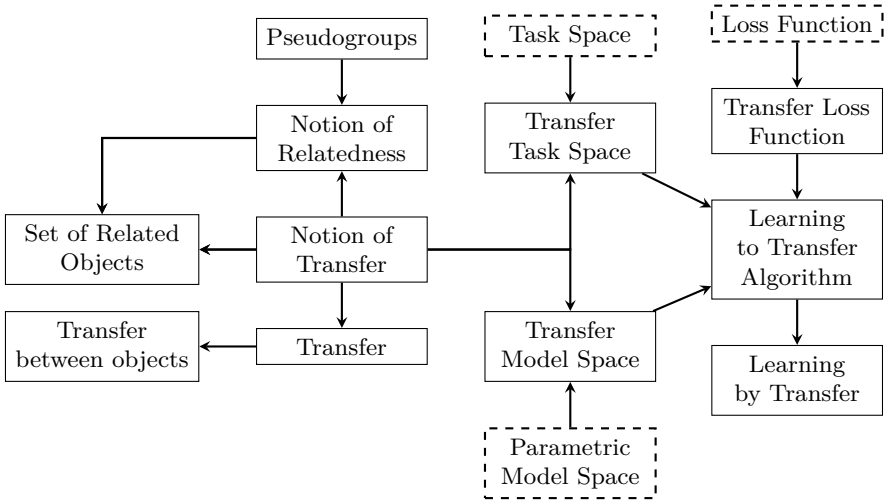


Figure 1.5. Map of Definitions for Chapter 6.

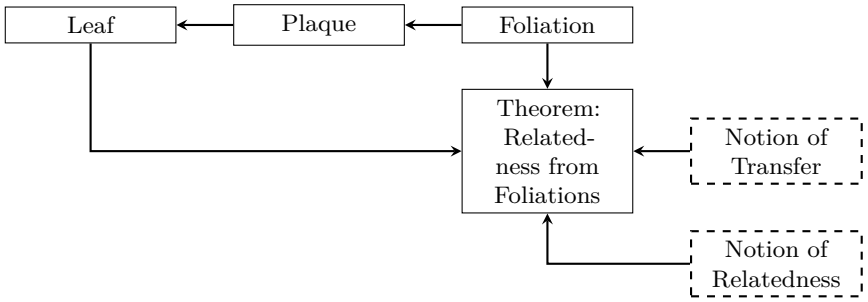


Figure 1.6. Map of Definitions for Chapter 7. Dashed boxes are definitions from other chapters. Relationships between these, if any, are not depicted here.

1.4 Notes on notation

The notation in this thesis can be unwieldy at times. For this reason, we summarise the following notational rules that we used throughout this thesis, unless otherwise stated.

- $\mathbb{R}$ refers to the real space, endowed with the standard topology, and the usual vector space structure. It is also assumed that $\mathbb{R}^+ = [0, \infty)$.
- Sets will be denoted by upper case letters, such as X. The complement of a set is denoted by X^{c}, and the closure of a set as $\overline{X}$. Note that this notation only makes sense when X is a set. For more clarity, we will say the *closure* $\overline{X}$.
- We will use the letters U_X and V_X to refer to subsets of a set X; the subscript won't be specified if the superset if obvious.
- A topology on a set X will be denoted as $\mathcal{O}_X$.
- Maps and functions will be denoted using f and g.
- Vectors are denoted by v. We use $\mathbf{v}$ if it is necessary to distinguish a vector from a scalar.
- Special spaces, elements of these spaces and maps that are key to this thesis will be denoted using the *franktur* font from the AMSFONTS package [46] of LaTeX.
- When denoting objects of a certain type, such as vectors, we will consistently use the notation given above, and will differentiate between different objects of the same type using subscripts, and superscripts (when necessary for clarity).
- Remarks, Examples and Proofs conclude with •, ▲ and ■ on the bottom right, respectively.
- Where necessary, we provide informational boxes that describe complicated notation.

This page was left intentionally blank.

I

The Structure of Learning

This page was left intentionally blank.

Chapter 2

Tasks and Spaces of Tasks

When one thinks of a machine learning (ML) problem, they would inevitably think of it terms such as supervised learning, reinforcement learning (RL) and unsupervised learning. Such names imply major differences between each ML problem; these differences can include their description, their formulation, or the techniques that are used to solve them. The setting of supervised learning for example, where one tries to find a function that best fits a given set of labelled data, is contrasted with unsupervised learning, which deals with unlabelled data. In turn, they differ wildly to the problem of reinforcement learning, which is akin to an optimal control problem where environments, agents and policies are involved[1].

These differences make the field of ML rich with knowledge and techniques inspired from mathematics, physics, computer science and evolutionary biology. There is however a common, underlying thread between these problems that allows us to make a definition of ML. Mitchell [67, Chapter 1] defines learning in machines as the process by which a computer program improves itself (relative to a given task and performance measure) with experience. From a slightly different point of view, Bishop [11, Chapter 1] implies that ML is a form of pattern recognition, where *automatic* (i.e. algorithmic) methods are used to discover *regularities* in data. It should be noted that these definitions are not mutually exclusive, but they put emphasis on different aspects.

What is structure

In this thesis, we want to place emphasis on the latter view. ML, in its broadest sense, is the problem of using machines to find *useful representations* of *structure*. Structure can be thought of as a set of rules, relationships or restrictions that can be said about a set of *things*. It allows us to identify such a set of things as the *particular* set (as opposed to other sets of things) that satisfies the structure imposed on it. In mathematical logic, structure is precisely defined as a *vocabulary* of function and relation symbols which have particular *interpretations* when applied to a set [16][2]. In this way, a *field* has the structure given by 2 operations (often symbolised as $+$ and $\times$) and the set over which they act on.

[1] It should be noted that we can combine these methods into a learning pipeline.

[2] We have simplified the description here for clarity and brevity.

Structure, described as *relationships* between things that constrain or identify sets of things, can be seen in a variety of examples. Take the function $f : \mathbb{R} \to \mathbb{R}$, such that,

$$f(x) = ax + b, \tag{2.1}$$

where $x, a, b \in \mathbb{R}$. For particular values of a and b, f is a relationship that is satisfied between some values $x, y \in \mathbb{R}$; if one were to pick two elements of $\mathbb{R}$ at random, they do not necessarily satisfy the relationship given. Thus, the structure here allows us to *identify* the subspace $\mathbb{R}^2$ of the particular straight line.

Structure can also be found in nature, where the regularities of honeycombs (see Figure 2.1) and crystalline objects, of the cyclic motions of planets and pendulums, and of the behaviours of flocks of birds and schools of fish can be construed from relationships that the constituent components satisfy between them. In the philosophical theory of Ontic Structural Realism, it is even posited that structure itself is the *true* and most fundamental nature of reality [30, 28]![3]

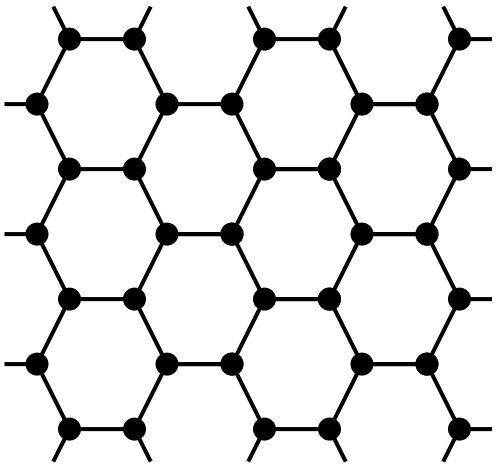


Figure 2.1. The regular structure of a honeycomb pattern. The structure of a honeycomb can be described as collection of points in $\mathbb{R}^2$ that form regular hexagons (with a constant side length), and have all sides adjacent to another hexagon. An alternative structure is that its the points of $\mathbb{R}^2$ that can be constructed using the repeated boundary of a regular polygon (with constant side length), such that they maximise the packing density of the area enclosed.

When looked at from this point of view, knowledge of structure allows us to identify the inclusion of elements in the set of things that obey said structure.

[3](Epistemic) Structural Realism (SR) was initially described by Worrall [126] as a way of reconciling Scientific Realism with the properties of Scientific Revolutions: if scientific theories can change so drastically, then surely they cannot be describing the fundamental nature of reality. SR noted that structural relationships remained consistent however, and thus stated that the scientific method led us to valid relationships between our self-defined objects that were good enough. Ontic Structural Realism took this further by claiming that structure was in fact the true nature of reality.

For example, given Equation (2.1), we can, for any value of $(x, y) \in \mathbb{R}^2$, test whether they satisfy the rule given by straight line. Alternatively, given any $x, y \in \mathbb{R}$, we can find the corresponding y, x respectively, that satisfies this structure. If we are given a portion of a regular crystalline lattice, and are told to extend it, we can easily generate new points and connection that satisfy the structure of the lattice. In mechanics, knowing that the motion of an object is governed by a particular differential equation allows us to trace its future path, after knowing only its initial conditions.

ML aims at using machines to identify such structure, and to then describe this structure using a useful representation. In the case of Equation (2.1), one could describe the structure as the literal sentence:

> the set of real numbers x, y where the difference between y and the product ax is equal to a constant b.

Such a representation of the structure is quite useless, even for a human, who would have to translate this to an equation of the form given by Equation (2.1); even then, for unwieldy numbers, they would often have to translate this into a form that could be understood by a machine (such as a calculator, or a general purpose computing device). Of course, the usefulness of a description is contextual (for example, if one wanted to waste another's time, then the statement above would be quite useful!)

Summary of the chapter

From this view, a ML problem has to refer to its structure and how it is going to be represented. This chapter focuses on the former. We formally define a system and its structure in Section 2.1. Then, Section 2.2 describes how a *task* can be defined in terms of structure. In this thesis, we will mostly be working from a deterministic point of view. For completeness, in Section 2.3, we will be restating our main definitions probabilistically. Following examples of tasks in Section 2.5, we will end with a discussion of how we can formally think about variations of systems and tasks, and consequently spaces of tasks.

2.1 Structure and systems

Tasks as systems with structure

If machine learning tries to find structure, a description of the task of a machine learning problem must include the system the structure of which it tries to find. Structure is a relationship between relata; a system is a collection containing relata, and relationships between them. When a task is described, the relata is often known, or assumed, and the a subset of the structure is unknown, prompting us to search for it (in this case, using machine learning).

Given a system, there can then be a *process* by which we interact with it. This is the mechanism that allows us to gather *data*, which are glimpses of the unknown structure that we wish to learn. Such a process takes a particular system, and presents us with some information that is *described* in some

representation that is accessible[4] to us (see Section 5.1 for a more in-depth discussion of representations). Since the structure of a system can be learned only if we can interact with it, a task must necessarily be described with a data-generating process. In addition, since we expect machines to do the learning, the data-generating process must describe the data in a representation that is accessible to machines.

Definition of a Structured System

Definition 2.1.1 (Structured System)**.** *A structured system* $\mathfrak{z}$ *(or system) is the tuple* $(\mathfrak{R}, \mathfrak{S})$ *of relata* $\mathfrak{R}$ *and associated structure* $\mathfrak{S}$*.*

$\mathfrak{R}$ corresponds to a set of sets, spaces, or systems themselves (see Section 2.1.1) that can admit relationships between each other. The structure $\mathfrak{S}$ corresponds to these relationships. That is,

Definition of a Structure

Definition 2.1.2 (Structure)**.** *Suppose we are given a set of sets or spaces, denoted by* $\mathfrak{R}$*. A structure* $\mathfrak{S}$ *on* $\mathfrak{R}$*, consists of:*

a) a set of domains and codomains derived from $\mathfrak{R}$*. These can be subsets or sets of subsets. The elements of* $\mathfrak{R}$ *must be included in this set.*

b) a set of maps between possible products of elements of the set of domains and codomains. This set must contain the identity maps for all sets in $\mathfrak{R}$*.*

Notation. When writing out definitions of structures, we will omit the identity maps for brevity; however, it should be assumed that they are present.

Furthermore, the set of domains and codomains will typically just be $\mathfrak{R}$. Therefore, unless necessary, we will not specify it when defining a structure. As with the identity maps, when we do specify it, we will not be repeating the elements of $\mathfrak{R}$.

Finally, if we write $f \in \mathfrak{S}$ where f is stated to be a map, then we mean that f belongs to the set defined by (b) in Definition 2.1.2. Similarly, if D is a set, then $D \in \mathfrak{S}$ is a domain or codomain from (a).

Thus, a system is a set of spaces that satisfy relationships between them that are specified in its structure[5]. It should be noted that this definition does not preclude relata or structures that are singletons; in fact, this is necessary to allow for tasks that are learned by unsupervised learning, and reinforcement learning (see Section 2.5.2).

Example 2.1.1 Linear Equation: Consider Equation (2.1); here, the relata would be,

$$\mathfrak{R} = \{\mathbb{R}\}, \tag{2.2}$$

[4] By accessible, we mean that it can be manipulated by some party, be it a human, or a machine.

[5] Such a definition of a system is applicable only in the context of the present work, and is not meant to used generally.

where $\mathbb{R}$ has the standard topology and the expected vector space structure[6]. The structure is,

$$\mathfrak{S} = \{f : \mathbb{R} \to \mathbb{R}; f(x) = ax + b\}, \tag{2.3}$$

where f and $\mathrm{id}_{\mathbb{R}}$ are continuous maps. ▲

Example 2.1.2 Topological Space: A topological space (see Appendix A.1 for more details) is a tuple $(X, \mathcal{O}_X)$, where $\mathcal{O}_X$ is a set of subsets of X that must satisfy the conditions given in Definition A.1.1. That is, a topological system $\mathfrak{z}_{\text{top}}$ is,

$$\mathfrak{z}_{\text{top}} = (\mathfrak{R}_{\text{top}}, \mathfrak{S}_{\text{top}}), \tag{2.4}$$

where,

$$\begin{aligned}
\mathfrak{R}_{\text{top}} &= \{X\},\\
\mathfrak{S}_{\text{top}} &= \{\{\mathcal{O}_X\},\\
&\quad \{\bigcup : ... \times \mathcal{O}_X \times ... \to \mathcal{O}_X; (..., U, ...) \mapsto \bigcup(..., U, ...) = ... \cup U \cup ...,\\
&\quad \cap : \mathcal{O}_X \times \mathcal{O}_X \to \mathcal{O}_X; (U_1, U_2) \mapsto \cap(U_1, U_2) = U_1 \cap U_2,\\
&\quad f_\varnothing : \mathcal{O}_X \to \mathcal{O}_X; U \mapsto f_\varnothing(U) = \varnothing,\\
&\quad f_X : \mathcal{O}_X \to \mathcal{O}_X; U \mapsto f_X(U) = X\}\}.
\end{aligned}$$

We have specified the set of domains and codomains, which in this case is the topology on X. The structure maps define the axioms that the elements of the topology must satisfy. The first structure equation denotes arbitrary unions, whereas the second set of structure maps denotes finite intersections. The final equations ensure that the null set and the full set X are in $\mathcal{O}_X$.

Remark 2.1.1: As per Definition A.3.1, a topological manifold is a topological space which satisfies certain conditions. Therefore, a topological manifold can be written as a system. Similarly, we can extend this to a smooth manifold, if in addition to the components in 2.4, we also include $\mathbb{R}^d$ and the domains of the charts in the relata, and the chart maps are included in the structure. •

▲

Example 2.1.3 Vector Space: A vector space of dimension d over a field $\mathbb{R}$ can be written as a system;

$$\mathfrak{z}_V = (\{V, \mathbb{R}\}, \{f_+ : V \times V \to V, f_\cdot : \mathbb{R} \times V \to V), \tag{2.5}$$

where f_+ is commutative and associative, and $f_\cdot$ is associative and distributive over addition in the field $\mathbb{R}$ and f_+. Further, V and $\mathbb{R}$ contain neutral elements for f_+ and $f_\cdot$ respectively, along with appropriate inverse elements. ▲

[6]This allows use to *add* and *multiply* with elements of $\mathbb{R}$.

2.1.1 Hierarchies of systems

In Definition 2.1.1, the relata can be spaces of objects, and the structure are relationships, denoted by maps, between such spaces. By being a *space*, each relatum can have intrinsic structure that can affect the structure. For example, it is often useful to make the relata *topological* spaces (see Appendix A.1), which allows us to make the structure maps *continuous*. As we will see in Section 2.3, it can be useful for elements of the relata to be probability spaces (see Appendix A.2 for preliminaries about measure theory).

The structure of a system corresponds to the relationships that affect the structure maps that we are interested in learning. If we are interested in learning about any intrinsic structure, then we could re-write a space X in terms of the structure in Definition 2.1.2. In Examples 2.1.2 and 2.1.3, we did just that for a topological and a vector space, respectively.

From this, it would be beneficial if systems and their structure can be *composed*, creating *hierarchical* descriptions of them. Formally;

Definition of a Inheritence of systems

Definition 2.1.3 (Inheritence of systems)**.** *Given a system* $\mathfrak{z} = (\mathfrak{R}, \mathfrak{S})$*, the system* $\mathfrak{z}_\mathrm{i} = (\mathfrak{R}_\mathrm{i}, \mathfrak{S}_\mathrm{i})$ *inherits* $\mathfrak{z}$ *if,*

$$X \in \mathfrak{R}_\mathrm{i}, \forall X \in \mathfrak{R},$$
$$f \in \mathfrak{S}_\mathrm{i}, \forall f \in \mathfrak{S}.$$

Notation. If we define a system $\mathfrak{z} = (\mathfrak{R}, \mathfrak{S})$, where $X \in \mathfrak{R}$ is itself a system $(\mathfrak{R}_X, \mathfrak{S}_X)$, then we mean that $\mathfrak{z}$ inherits X.

The relata can constitute systems, and the structure maps in this system are available to be used in the new system. This system has inherited the systems in the relata. Consider the system in Example 2.1.1. It was necessary that the relata $\{\mathbb{R}\}$ are at least vector spaces. In Equation (2.3), the key structure map is $f : \mathbb{R} \to \mathbb{R}; x \mapsto f(x) = ax + b$, for a, b in $\mathbb{R}$. When writing this, we had invoked the structure maps $f_a : \mathbb{R} \to \mathbb{R}; f(x) = ax$, and $f_{1,1} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}; f(x, b) \mapsto f(x, b) = x + b$ from the vector space system in Equation (2.5). By currying the second parameter of the latter, we can write f as the composition,

$$\begin{aligned} f : \mathbb{R} &\to \mathbb{R} \\ x &\mapsto f(x) = f_{1,1}(b) \circ f_a(x). \end{aligned}$$

Suppose, we further want to discuss *continuity* of structure maps. Then, the domains and codomains of such structure maps must be topological systems. Given topological systems,

$$\mathfrak{z}^1_\mathrm{top} = \{\{X_1\}, \{\mathcal{O}_{X_1}, \mathfrak{S}^1_\mathrm{top}\}\},$$
$$\mathfrak{z}^2_\mathrm{top} = \{\{X_2, \{\mathcal{O}_{X_2}, \mathfrak{S}^2_\mathrm{top}\}\},$$

we can say that a system $\mathfrak{z} = (\{\mathfrak{z}^1_{\text{top}}, \mathfrak{z}^2_{\text{top}}\}, \{f : X_1 \to X_2\})$ has a continuous structure map f if for $U \in \mathcal{O}_{X_2}$, then $\text{preim}_f(U) \in \mathcal{O}_{X_1}$. Given the structure maps of the topological spaces, this also means the f must satisfy this condition under compositions with such structure maps.

2.1.2 Subsystems

Given a system $(\mathfrak{R}, \mathfrak{S})$, a subsystem of this system is a subset of it in the following way,

Definition 2.1.4 (Subsystem)**.** *Given a system $\mathfrak{z} = (\mathfrak{R}, \mathfrak{S})$, a subsystem $\widetilde{\mathfrak{z}} = (\mathfrak{R}_{\widetilde{\mathfrak{z}}}, \mathfrak{S}_{\widetilde{\mathfrak{z}}})$ is another system where,* Definition of a Subsystem

$$\mathfrak{R}_{\widetilde{\mathfrak{z}}} = \{S \mid (S \neq \varnothing) \subseteq U, \forall U \in \mathfrak{R}\},$$
$$\mathfrak{S}_{\widetilde{\mathfrak{z}}} = \mathfrak{S}|_{\mathfrak{R}_{\widetilde{\mathfrak{z}}}}.$$

Notation. $\mathfrak{S}|_{\mathfrak{R}_{\widetilde{\mathfrak{z}}}}$ denotes

a) that for a domain or codomain an element of $U^X \in D^X \in \mathfrak{S}$, is a set of subsets derived from $X \in \mathfrak{R}$,

$$(U^X_{\widetilde{\mathfrak{z}}} \in D^X_{\widetilde{\mathfrak{z}}}) = U^X \cap X_{\widetilde{\mathfrak{z}}},$$

b) and that structure maps to the subsets of the appropriate domains and codomains as in $\mathfrak{R}_{\widetilde{\mathfrak{z}}}$. That is, if say $\mathfrak{z} = (\{X, Y\}, \{f : X \to Y\})$, then a subsystem could be $\widetilde{\mathfrak{z}} = (\{U_X, U_Y\}, \{f|_{\{U_X, U_Y\}} : U_X \to U_Y\})$.

Thus, a subsystem consists of relata that are subsets of the relata of the original system, and its structure maps are similar, except that they are constrained to the appropriate subsets.

Operations on subsystems. Unions and intersections of subsystems can be defined as the following:

Definition 2.1.5 (Union of subsystems)**.** *Suppose we are given subsystems $\widetilde{\mathfrak{z}}1 = (\mathfrak{R}_{\widetilde{\mathfrak{z}}1}, \mathfrak{S}_{\widetilde{\mathfrak{z}}1})$ and $\widetilde{\mathfrak{z}}2 = (\mathfrak{R}_{\widetilde{\mathfrak{z}}2}, \mathfrak{S}_{\widetilde{\mathfrak{z}}2})$ of a system $\mathfrak{z} = (\mathfrak{R}, \mathfrak{S})$. Then,* Definition of a Union of subsystems

$$\widetilde{\mathfrak{z}}1 \cup \widetilde{\mathfrak{z}}2 = (\mathfrak{R}_{\widetilde{\mathfrak{z}}1 \cup \widetilde{\mathfrak{z}}2}, \mathfrak{S}_{\widetilde{\mathfrak{z}}1 \cup \widetilde{\mathfrak{z}}2}),$$

where,

$$\mathfrak{R}_{\widetilde{\mathfrak{z}}1 \cup \widetilde{\mathfrak{z}}2} = \{U_1 \cup U_2 \mid \forall (U_1 \in \mathfrak{R}_{\widetilde{\mathfrak{z}}1}) \subseteq (U \in \mathfrak{R})\}, \forall (U_2 \in \mathfrak{R}_{\widetilde{\mathfrak{z}}2}) \subseteq (U \in \mathfrak{R})\},$$
$$\mathfrak{S}_{\widetilde{\mathfrak{z}}1 \cup \widetilde{\mathfrak{z}}2} = \mathfrak{S}|_{\mathfrak{R}_{\widetilde{\mathfrak{z}}1 \cup \widetilde{\mathfrak{z}}2}}.$$

$\mathfrak{R}_{\widetilde{\mathfrak{z}}1 \cup \widetilde{\mathfrak{z}}2}$ is the relata obtained by taking the union of a relata $U_1 \in \mathfrak{R}_{\widetilde{\mathfrak{z}}1}$ with the corresponding relata $U_2 \in \mathfrak{R}_{\widetilde{\mathfrak{z}}2}$ where both U_1 and U_2 are subsets of the same relata $U \in \mathfrak{R}$ of the system they are subsystems of.

Union of subsystems is a subsystem

Theorem 2.1.1. *Suppose we are given subsystems $\widetilde{\mathfrak{z}}_1$ and $\widetilde{\mathfrak{z}}_2$ of a system $\mathfrak{z}$. Then $\widetilde{\mathfrak{z}}_1 \cup \widetilde{\mathfrak{z}}_2$ is also a subsystem of $\mathfrak{z}$.*

Proof. The proof follows by inspection. The elements of $\mathfrak{R}_{\widetilde{\mathfrak{z}}_1 \cup \widetilde{\mathfrak{z}}_2}$ are, by definition subsets of elements in $\mathfrak{R}$. Furthermore, since none of the relata in $\widetilde{\mathfrak{z}}_1$ and $\widetilde{\mathfrak{z}}_2$ are null sets, $\mathfrak{R}_{\widetilde{\mathfrak{z}}_1 \cup \widetilde{\mathfrak{z}}_2}$ does not contain any null sets.

Finally, the restriction of a valid structure to a subset of its relata is also a valid structure. ■

Similarly, the intersection of subsystems can be defined, as in Appendix B.1.

2.2 A Structural Definition of Tasks

The key difference between a task and a system is that, while a system is defined abstractly, a task has a mechanism by which we can interact with it, as we will see in this section.

2.2.1 Interacting with systems

For the purposes of learning, we must be able to interact with systems. This can be done using *probabitive processes*;

Definition of a Probative Process

Definition 2.2.1 (Probative Process)**.** *Given a system $\mathfrak{z} = (\mathfrak{R}, \mathfrak{S})$, a probative process $\mathfrak{P}_{\mathfrak{z}}$ that can be* applied *to $(\mathfrak{R}, \mathfrak{S})$ is a map,*

$$\mathfrak{P}_{\mathfrak{z}} : \mathfrak{U}^m \to \mathfrak{U}^n,$$

where $\mathfrak{U} \in \mathfrak{R}$, $(m, n \in \mathbb{N}) \leq |\mathfrak{R}|$, and $\mathfrak{U}^m = \mathfrak{U}_1 \times ... \times \mathfrak{U}_m$, and $\mathfrak{U}^n = \mathfrak{U}_1 \times ... \times \mathfrak{U}_n$ are product spaces of some or all elements of $\mathfrak{R}$.

$\mathfrak{P}_{\mathfrak{z}}$ must also satisfy that,

$$\mathfrak{P}_{\mathfrak{z}} = f_1 \circ ... \circ f_s,$$

where $f_i \in \mathfrak{S}$ and $(s \in \mathbb{N}) \leq |\mathfrak{S}|$.

Notation. Henceforth, the notation for $\mathfrak{P}_{(\mathfrak{R},\mathfrak{S})}$ will be suppressed to $\mathfrak{P}$, where the system it corresponds should be deemed from context, unless otherwise specified.

Since there can be possible probative processes that are functionally different, but have the same domain, we will denote by $\{\mathfrak{P}_i\}_{i\in\mathcal{I}}^{\mathfrak{U}^m}$, for $\mathfrak{U}^m = \mathfrak{U}_1 \times ... \times \mathfrak{U}_m$ where $\mathfrak{U}_i \in \mathfrak{R}$, a set of probative processes that have the same domain $\mathfrak{U}^m$. It should be noted that this set does not contain *every* such probative process, but a subset which satisfies the stated condition.

Probative processes are used to generate datasets. The process that generates data is called a data-generating process, which is defined as:

Definition 2.2.2 (Data-generating Process)**.** *Given a system* $(\mathfrak{R}, \mathfrak{S})$*, and a collection of probative processes* $\{\mathfrak{P}_i\}_{i\in\mathcal{I}}^{\mathfrak{U}^m}$*, a data-generating process* $\mathfrak{P}^{\mathfrak{D}}$ *is the map,* Definition of a Data-generating Process

$$\begin{aligned}\mathfrak{P}^{\mathfrak{D}} : \mathfrak{U}^m &\to \mathfrak{U}_1^{n_1} \times ... \times \mathfrak{U}_{|\mathcal{I}|}^{n_{|\mathcal{I}|}} \\ r &\mapsto \mathfrak{P}^{\mathfrak{D}}(r) = (\mathfrak{P}_1(r), ..., \mathfrak{P}_{|\mathcal{I}|}(r)).\end{aligned}$$

A data-generating process is created from probative processes. We will call the output of a data-generating process a *data point*. Then, a *dataset* is a collection of data points.

Definition 2.2.3 (Dataset)**.** *For a system* $(\mathfrak{R}, \mathfrak{S})$ *and a data-generating process* $\mathfrak{P}^{\mathfrak{D}}$*, a dataset* $\mathfrak{D}$ *of size* n *is a set of the results of* n *applications of* $\mathfrak{P}^{\mathfrak{D}}$*. Each such output is a data point.* Definition of a Dataset

Notation. We will use $\mathfrak{P}$ to denote a data-generating process, unless specified otherwise. This is because we won't be using probabitive processes that aren't data-generating processes.

Finally, we can present the following definition of a task,

Definition 2.2.4 (Task)**.** *A task, denoted by* $\mathfrak{t}$*, is a collection of a system* $(\mathfrak{R}, \mathfrak{S})$ *that can admit one or more probative processes, and a data-generating process* $\mathfrak{P}$ *that is constructed using such processes. Thus,* Definition of a Task

$$\mathfrak{t} = ((\mathfrak{R}, \mathfrak{S}), \mathfrak{P}) \tag{2.6}$$

In this definition, for a given system, we could have various tasks that differ in their data-generating processes. A task, therefore, can be interpreted as a system, and particular aspects of the system's structure that we wish to learn about; the data-generating process, which consists of various probative processes that are composed of structure maps, embodies these aspects. Thus, there can be structure maps that are not utilised in the data-generating process, indicating that these structure maps cannot be directly interacted with. However, these maps are important, nonetheless, and any ML solution that tries to solve a ML problem based on a task must also satisfy these structure maps.

2.3 Probability and Tasks

So far, the definitions that we had made were written in terms of *deterministic* maps. However, it is also possible to make these definitions in probabilistic terms, where instead of deterministic maps, we use either probability measures, measurable maps or stochastic processes (see Appendix A.2). Probability theory is useful in problem of ML, particularly because we often deal with limited data, and datasets that aren't necessarily generated deterministically.

Thus, we can really only talk about *averages*, expressed with measures of *uncertainty*.

In a probabilistic interpretation of Definition 2.2.4, the relata of a system would be either probability spaces, or measurable spaces. Furthermore, the structure maps would be measurable maps.

Definition of a Structure (probabilistic)

Definition 2.3.1 (Structure (probabilistic))**.** *Suppose we are given a set of spaces of things, denoted by* $\mathfrak{R}$*, where for* $X \in \mathfrak{R}$*,* X *is either a probability space, or a measurable space. A structure (probabilistic) on* $\mathfrak{R}$*,* $\mathfrak{S}_{\mathfrak{R}}$*, is a set of measurable maps between appropriate elements of* $\mathfrak{R}$*.*

$\mathfrak{S}$ *must also contain the identity maps for all spaces in* $\mathfrak{R}$*.*

The structure maps could contain operations with other random variables, such as white noise. Where the domains of a structure map is a product space, the product measure space is assumed.

Definition of a System (probabilistic)

Definition 2.3.2 (System (probabilistic))**.** *A system (probabilistic) is a tuple* $(\mathfrak{R}, \mathfrak{S})$ *of relata, where for* $X \in \mathfrak{R}$*,* X *is either a probability space, or a measurable space, and associated structure (probabilistic).*

Then, a probative process (probabilistic) is as defined in Definition 2.2.1, where now, we have compositions of measurable maps. The representation maps would also be measurable. Where there are measurable maps, we would take *samples*. Then, a data-generating process would produce samples of the joint measure of the codomain that is induced by the composition of measurable maps.

Definition of a Probative Process (probabilistic)

Definition 2.3.3 (Probative Process (probabilistic))**.** *Given a system (probabilistic)* $(\mathfrak{R}, \mathfrak{S})$*, a probative process (probabilistic)* $\mathfrak{P}_{(\mathfrak{R},\mathfrak{S})}$ *that can be* applied *to* $(\mathfrak{R}, \mathfrak{S})$ *is a measurable map,*

$$\mathfrak{P}_{(\mathfrak{R},\mathfrak{S})} : \mathfrak{U}^m \to \mathfrak{U}^n,$$

where $\mathfrak{U} \in \mathfrak{R}$*,* $(m, n \in \mathbb{N}) \leq |\mathfrak{R}|$*, and* $\mathfrak{U}^{m,n} = \mathfrak{U}_1 \times ... \times \mathfrak{U}_{m,n}$ *is a product space of some or all elements of* $\mathfrak{R}$*.*

$\mathfrak{P}$ *must also satisfy that,*

$$\mathfrak{P} = f_1 \circ ... \circ f_s,$$

where $f_i \in \mathfrak{S}$ *and* $(s \in \mathbb{N}) \leq |\mathfrak{S}|$*.*

Definition of a Data-generating Process (probabilistic)

Definition 2.3.4 (Data-generating Process (probabilistic))**.** *Given a system (probabilistic)* $(\mathfrak{R}, \mathfrak{S})$*, and collection of probative processes (probabilistic)* $\{\mathfrak{P}_i\}_{i\in\mathcal{I}}^{\mathfrak{U}^m}$*, then a data-generating process (probabilistic)* $\mathfrak{P}^{\mathfrak{D}}$ *is measurable map,*

$$\begin{aligned} \mathfrak{P}^{\mathfrak{D}} : \mathfrak{U}^m &\to \mathfrak{U}_1^{n_1} \times ... \times \mathfrak{U}_{|\mathcal{I}|}^{n_{|\mathcal{I}|}} \\ r &\mapsto \mathfrak{P}^{\mathfrak{D}}(r) = (\mathfrak{P}_1(r), ..., \mathfrak{P}_{|\mathcal{I}|}(r)). \end{aligned}$$

Finally,

Definition 2.3.5 (Task (probabilistic))**.** *A task (probabilistic), denoted by* $\mathfrak{t}$*, is a collection of a system (probabilistic)* $(\mathfrak{R}, \mathfrak{S})$ *that can admit one or more probative processes (probabilistic), and a data-generating process (probabilistic)* $\mathfrak{P}$ *which is constructed using such processes (probabilistic). Thus,*

Definition of a Task (probabilistic)

$$\mathfrak{t} = ((\mathfrak{R}, \mathfrak{S}), \mathfrak{P}) \tag{2.7}$$

Notation. In the section, we did not make a notational distinction between the deterministic and probabilistic definitions of systems, structures, probabitive processes, data-generating processes, or tasks. Neither will we do so in the rest of this thesis, because we will be using the deterministic definitions throughout this thesis, unless specified otherwise.

2.4 Arguments for a Structural Definition of Tasks

A natural question that arises is whether there is a need for such a definition of a task; why is it useful to use a definition of a task that explicitly exposes the underlying structure of a system. In statistical learning theory, a learning task (of a supervised learning problem) is defined as a measure (or a probability density function) on a joint input-output space [92].

Definition 2.4.1 (Task (Statistical learning theory))**.** *Given a measurable space* $(X \times Y, \Sigma_{X\times Y})$*, a task is a measure* τ *on this space.*

Definition of a Task (Statistical learning theory)

Such a definition has a few advantages. For one, it is quite simple to understand; the same structure that we had described is still present implicitly in the measure, but it is hidden from the practitioner. Furthermore, this definition directly gives us the data-generating process; it is the process of sampling from the task. Thus, from such a definition, we are saying that the data was generated from some probability distribution; we do not care much about the internal mechanisms of this distribution, but expect that it contains the structure that we are interested in learning about. In fact, our definition (if written probabilistically) subsumes this definition.

Due to its simplicity, Definition 2.4.1 is also practical. From this, we can derive such things as the Probably Approximately Correct (PAC) learning scheme, from which wonderful and useful ideas have been bourn. PAC learning captures the uncertainty that is encountered when dealing with datasets that are sampled, and therefore aren't deterministic, and the error that is incurred when we make approximations of the underlying distribution. These include the Vapnik-Chernovenkis (VC-) dimension [114] which gives us a tool for measure data complexity; that is, how quickly does the uncertainty in our learning problem reduce as we increase the size of the dataset.

Explicit structure is key to transfer. A key benefit to Definition 2.2.4 is when thinking about problems of transfer. Often in the literature, problems of transfer are relegated to a separate subfield, with many names such as transfer learning, meta learning, domain adaptation, etc. As we will see in Chapter 6, problems of transfer are, in essence the application of ML to single task problems, but where the structure is on a higher, hierarchical level. In Section 2.6, we see that structure maps can be used to define a strict notion of variations between tasks. These are particularly useful for interpreting transfer in ML problems.

Elegance of philosophical expression. We must also confess that our primary motivations were, among others, philosophical and ontological. Definition 2.2.4 was designed to invoke a different point of view regarding problems of ML, where more emphasis was put into reasoning about *what* we wanted to learn, rather than *how* we can learn it. Of course, these notions aren't mutually exclusive, and one isn't more or less important than the other. Traditionally however, ML has dealt mostly with, to great success, the latter. Thus, we wished to fill in this gap. We expect that emphasizing a different aspect of the problem would help in gaining novel insights, most likely in the long run. A particular area that we believe will greatly benefit from approaching ML through our lens is in the interpretability of learned solutions.

Despite this long term view, there are some immediate practical merits to Definition 2.2.4, especially when working with problems involving transfer. In [7], we see a mathematical analysis of transfer that applies the methods of PAC learning to problems involving multiple tasks. In this, several bounds on the VC-dimension were derived without explicitly handling the relationships between tasks. Then in [10, 8, 9], these bounds were tightened by considering the structure of the learning task explicitly. The authors described a relationship between tasks in terms of an action of a group (see Appendix A.4).

Laying a concrete foundation. These definitions provide us with a robust and elementary foundation from which to build analyses of ML problems and techniques. The structural view expresses key components of a ML problem as maps that we can vary against, when choosing and developing models, loss functions and learning algorithms.

In addition to aiding the development of these ideas, we expect that this view will also help in communicating such ideas. When written in this way, the key arguments and assumptions are laid bare, for anyone subscribing to this view to easily understand and scrutinize.

Understanding the structure of learning is important. Recent trends in research also show a growing interest, and benefit of being more explicit about the structure of ML problems. Perhaps the best example of this is in Convolutional Neural Networks (CNNs). CNNs are foundational to successes in computer vision. As a model, a CNN is quite simple, which allowed it to be attempted in a variety of problems. Recently, it was understood that the

key property that made a CNN so useful in problems involving images was a convolutional layer's property of being translation equivariant [20].

This, in retrospect, is an obvious notion; if we were to translate a object in an image, we shouldn't expect any salient changes in the interpretation of the image (this is especially true in classification problems). Once this was identified, it became possible to apply convolutional methods to more exotic spaces, starting with spheres [21], where the equivariance was w.r.t. a different group (SO(3)), acting on a sphere. The changes here aren't trivial; for one, the topology of a sphere is very different to the topology of $\mathbb{R}^2$. More recent work [18] has generalised this notion to a convolution on any (homogeneous) space w.r.t. any appropriate action of a group.

[87] shows how to explicitly model properties that are experienced by physical problems that fall under classical (lets say Newtonian) physics, namely the fact that they obey the Principle of Least Actions [4]. In our previous work [76], we did learning on multi-task reinforcement learning problems which were known to have a particular structure regarding how tasks were distributed.

These examples show that explicit expressions of the structure has been useful. Being explicit about the components of any theory or idea is always useful for understanding, if one ignores that being overbearingly so can hurt the barrier to entry. Often the discovery of new ideas, particularly in science, is initiated by attempts at trying to find the explicit structural components that can explain some phenomena. In ML, the ideas of implicit gradient regularisation [5, 96], and flat minima [41, 42, 24, 17] were identified when studying the generalisation properties of deep learning. Thus, in the present thesis, we wish to explore the structure of ML, with a particular emphasis on transfer.

2.5 Examples of tasks in Machine Learning

In order to aide in the understanding of Definition 2.2.4 of a task, we will be restating the tasks provided by typical ML problems in the following. It should be noted that the term *ML problem* is used to classify names such as *supervised learning*, *unsupervised learning*, *reinforcement learning*, and the like. The task of such a problem encompasses a system that contains some unknown structure map that we wish to learn. As expected, the task does not tell us anything about the models, loss functions nor learning algorithms used to solve these problems.

2.5.1 Task of Supervised Learning

Supervised learning is perhaps the easiest to express in our definition. Typically, it is defined for ML problems where *labels* are known; this is best contrasted with unsupervised learning. More formally the task of a supervised learning problem is,

Definition of a Supervised Learning Task (Typical)

Definition 2.5.1 (Supervised Learning Task (Typical))**.** *A supervised learning problem is a ML task where the data is assumed to have been generated from a map of the form,*

$$f : X \to Y, \tag{2.8}$$

where X and Y are the input and output spaces respectively. Furthermore, the data are of the form (x, y), where $x \in X$ and $y \in Y$, and $y = f(x)$.

Supervised learning includes problems of regression and classification, where the differences are in the output space Y; Y is discrete or categorical in classification. A typical regression problem was given in Equation (2.1), where $X, Y = \mathbb{R}$. A classification problem can be where X is a space of matrices $\mathbb{R}^{m \times n}$ corresponding to images, Y is $\{\text{cat}, \text{dog}\}$, and f is a map that identifies whether the subject of an image is a cat or a dog.

To cast the task of supervised learning in terms of Definition 2.2.4, we must identify the system and data-generating process it composes of.

Definition of a Supervised Learning Task

Definition 2.5.2 (Supervised Learning Task)**.** *A supervised learning (SL) task, denoted by $\mathfrak{t}_{\text{SL}}$, is a task that, the components of which at least satisfy the tuple $((\mathfrak{R}_{\text{SL}}, \mathfrak{S}_{\text{SL}}), \mathfrak{P}_{\text{SL}})$, where,*

$$\mathfrak{R}_{\text{SL}} = \{X, Y\},$$

$$\mathfrak{S}_{\text{SL}} = \{f : X \to Y\},$$

and,

$$\begin{aligned} \mathfrak{P}_{\text{SL}} : X \times X &\to X \times Y \\ (x, x) &\mapsto \mathfrak{P}_{\text{SL}}(x, x) = (\text{id}_X(x), f(x)). \end{aligned}$$

Thus, we see that the structure that we are trying to learn, as encoded in the data-generating process, is the relationship between X and Y, as given by f. As we mentioned in Section 2.1, a task could contain structure that is not utilised in the data-generating process, indicating the intent of the learning problem. Therefore, a supervised learning task *could* contain additional structure that isn't interesting.

The system to which Equation (2.1) belonged to could have been Equation (2.3). It's data-generating process would have simply been,

$$\mathfrak{P}_{\text{reg}} : X \to X \times Y \tag{2.9}$$

$$x \mapsto \mathfrak{P}_{\text{reg}}(x) = (x, ax + b). \tag{2.10}$$

2.5.2 Task of Reinforcement Learning

Reinforcement learning (RL) is a genre of ML that deals with the problem of decision making [104]. A typical scenario is the pendulum swing-up problem that is depicted in Figure 2.2. The pendulum of a fixed mass and length starts at its lowest position. Then, given some constrained torque that can be applied

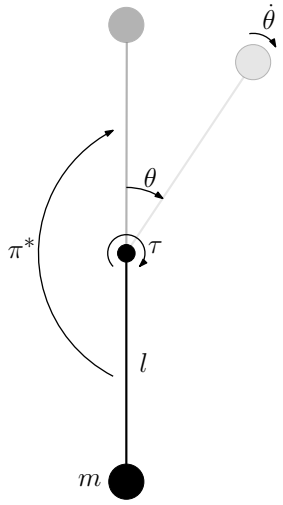


Figure 2.2. The pendulum problem is to find an optimal policy π^* that applies torque τ at the pivot of a pendulum of mass m and length l to move it from its lowest position to its highest position, and to balance it there. The optimality of the policy is determined by some reward function.

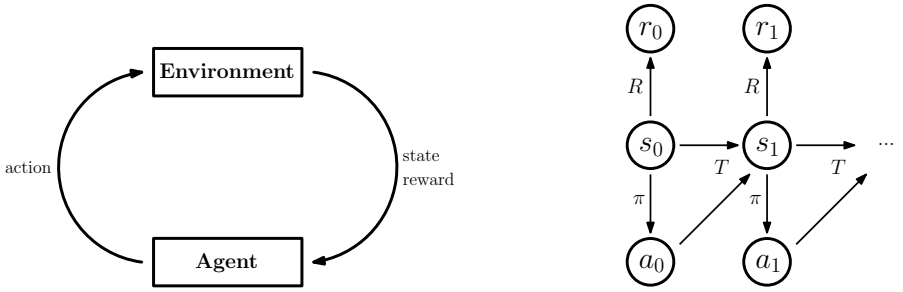
at the pivot, the goal of RL is to find a strategy of applying a sequence of torques to move the pendulum to its highest position, and then balance it there. The constraint on the torque is limited to ensure that the pendulum must be swung up. Of course, the pendulum is assumed to act under gravity, and is usually not affected by any resistances (such as friction or air resistance). A RL problem is typically modelled in terms of a Markov Decision Process (MDP). An example graph of a MDP is shown in Figure 2.3b. A MDP in RL is typically specified by a tuple $\mathfrak{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, which consists of a state space $\mathcal{S}$, action space $\mathcal{A}$, transition dynamics $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, reward function $\mathcal{R} : \mathcal{S} \to \mathbb{R}$[7] and a discount factor $\gamma \in [0, 1]$[8].

Components of RL

The state space denotes the possible states that the environment can be in, and the action space tell us the possible actions that the agent can make in this environment. The transition dynamics tell us how the present state changes given a particular action. The action is chosen using a policy $\pi : \mathcal{S} \to \mathcal{A}$ that describes the strategy that is taken, since it tells us the action to carry out given the current state. Finally, the reward function tells us how good arriving at the current state of the environment is. A key assumption that is made here, which is also the etymology of *Markov* in MDP, is that the system is setup such that all relevant information is given in the current state of the environment. That is, knowledge of previous states does not make a difference. Furthermore, *time* is often assumed to be discrete.

[7]Some works denote the reward function as $\mathcal{R} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. In the present work, we will use the simpler form written in the main text, as it doesn't change the more abstract discussions made.

[8]If the MDP is defined to be *finite*, the discount factor doesn't need to be specified.



(a) The typical reinforcement learning loop. The environment provides information regarding its state and reward to an agent.

(b) An example of a possible Markov Decision Process that is used in reinforcement learning.

Figure 2.3. Graphical representations of reinforcement learning.

In the case of the pendulum problem, the state space is often the angle made by the pendulum and the angular velocity of the pendulum. The angle can be measured from the goal, as in Figure 2.2. The action space is the size of the torque. The transition dynamics are the dynamics of the pendulum. However, since time is often discretised, these dynamics are often the Euler approximations of the true differential equations that describe the motion of a pendulum, and tell us how the angle and angular velocity change given a particular torque. The reward function can be the sum of the current angle and angular velocity (that is, the norm of the current state vector). The goal of the RL problem is to find an optimal policy π^* that solves the decision making problem posed by the MDP. This decision making process is defined w.r.t. a value function, which is defined as,

Definition of a Value Function

Definition 2.5.3 (Value Function)**.** *Given a MDP, $\mathfrak{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, and a policy π, the value function V_π is a map:*

$$V_\pi : \mathcal{S} \to \mathbb{R}$$
$$s_0 \mapsto V_\pi s_0 = \sum_{t=0}^{\infty} \mathcal{R} \circ \mathcal{T}(s_t, \pi(s_t))\gamma^t$$

Remark 2.5.1: RL is typically defined probabilistically; in the present work, we will be making definitions that are purely deterministic. •

The value function $V_\pi(s_0)$ evaluates how good a current state s_0 is w.r.t. the potential total discounted reward that could be achieved following a particular policy π. The total discounted reward is also the *return*. The optimality of a policy is determined by whether it maximises its associated value function. That is,

$$\pi^* = \arg\max_\pi V_\pi(s_0). \tag{2.11}$$

Thus, in the case of the pendulum problem and its reward function, we see that the optimal policy must also minimise the time that it takes to swing up.

The typical data of an RL problem is given in the form of a trajectory. A trajectory is a rollout of the MDP of a RL problem, and is expressed in terms of a collection of transitions, which is defined as,

Definition 2.5.4 (Transition)**.** *For a given MDP $\mathfrak{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, and policy π, a transition is a tuple $(s, a, s', r) \in \mathcal{S} \times \mathcal{A} \times \mathbb{R} \times \mathcal{S}$, where,* Definition of a Transition

$$\begin{aligned} a &= \pi(s), \\ s' &= \mathcal{T}(s, a), \\ r &= \mathcal{R}(s). \end{aligned}$$

The product space $\mathcal{S} \times \mathcal{A} \times \mathbb{R} \times \mathcal{S}$ is denoted by $\mathcal{O}_{\text{RL}}$.

A transition gives us information regarding a single *step* in the MDP. As such, a transition can be indexed by $\mathbb{N}$, where for a time step $t \in \mathbb{N}$, there is a transition (s_t, a_t, s_{t+1}, r_t). A collection of transitions that are ordered by time is called a trajectory. In order to formalise a trajectory, it is useful to define a map as follows,

Definition 2.5.5 (Single rollout)**.** *Given a MDP $\mathfrak{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, and policy π, a single rollout is a map $f^{\pi,1}_{\text{roll}}$ that is defined as;* Definition of a Single rollout

$$\begin{aligned} f^{\pi,1}_{\text{roll}} &: \mathcal{S} \to \mathcal{O}_{\text{RL}} \\ &s \mapsto f^{\pi,1}_{\text{roll}}(s) = (s, a = \pi(s), \mathcal{R}(s), \mathcal{T}(s, a)). \end{aligned}$$

A modified single rollout $\overline{f^{1}_{\text{roll}}}$ can also be defined as,

$$\begin{aligned} &\overline{f^{\pi,1}_{\text{roll}}} : \mathcal{O}_{\text{RL}} \to \mathcal{O}_{\text{RL}} \\ &(s, a, r, s') \mapsto \overline{f^{\pi,1}_{\text{roll}}} : (s, a, r, s') = (s', a = \pi(s'), \mathcal{R}(s'), \mathcal{T}(s', a)), \end{aligned}$$

to take in transitions and output the next transition.

Notation. Note that the notation $(s, \boldsymbol{a = \pi(s)}, \mathcal{R}(s), \boldsymbol{\mathcal{T}(s, a)})$ here implies that when we invoke $\mathcal{T}(s, a)$, we are using the same output of $\pi(s)$ that was created at $a = \pi(s)$, rather than invoking the policy again. This isn't important when the MDP is deterministic. However, when it is probabilistic, this means that we aren't sampling from the induced measure over actions twice, but rather recording and using the same single sample.

Such a rollout can be composed to create a $t-$rollout, which records rolling out the MDP up to a time step $t \in \mathbb{N}$. That is,

Definition 2.5.6 ($t-$rollout)**.** *Given a MDP $\mathfrak{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, and policy π, a $t-$rollout, for $t \in \mathbb{N}$ is a map $f^{\pi,t}_{\text{roll}}$ that is defined as,* Definition of a $t-$rollout

$$\begin{aligned} f^{\pi,t}_{\text{roll}} &: \mathcal{S} \to \mathcal{O}^{\intercal}_{\text{RL}} \\ &s_0 \mapsto f^{\pi,t}_{\text{roll}}(s) = ((s_0, a_0, r_0, s_1), (s_1, a_1, r_1, s_2), ..., (s_t, a_t, r_t, s_{t+1})), \end{aligned}$$

where,

$$
\begin{aligned}
(s_0, a_0, r_0, s_1) &= f_{\text{roll}}^{\pi,1}(s_0), \\
(s_1, a_1, r_1, s_2) &= \overline{f_{\text{roll}}^{\pi,1}}(s_0, a_0, r_0, s_1), \\
(s_t, a_t, r_t, s_{t+1}) &= \overline{f_{\text{roll}}^{\pi,1}}(s_{t-1}, a_{t-1}, r_{t-1}, s_t).
\end{aligned}
$$

A rollout then is the application of a single rollout *ad infinitum*,

Definition of a Rollout

Definition 2.5.7 (Rollout)**.** *Given a MDP $\mathfrak{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, and policy π, a rollout is a map f_{roll}^{π}, which is a $\infty-$rollout.*

It should be noted that while the transition dynamics and reward functions are explicitly described in its formulation, RL assumes that we are not given access to these[9]. That is, the optimal policy must be identified merely using data given in the form of trajectories of states, actions and rewards. This leads to many different ways of solving the RL problem. Model free methods, such as Q-learning [120] for example, immediately try to learn the policy, while model-based methods, such as PILCO [22] learn a model of the transition dynamics, and uses that to learn a policy through simulation. Off-policy methods [120, 104] use data generated from any policy, whereas on-policy methods [3] use data generated from the current policy. RL can be further classified into value-based methods (such as Q-learning) which try to learn a form of the problem's optimal Value function explicitly, or policy-search methods [105], such as REINFORCE [123], explicitly model and optimise the policy. In value-based methods, the policy is derived from the value function.

A structural definition of the task of a RL problem must include the ability to generate trajectories. Furthermore, its relata must contain the action space, state space, and $\mathbb{R}$ since transition dynamics and reward function depend on these. Thus,

Definition of a Task of Reinforcement Learning

Definition 2.5.8 (Task of Reinforcement Learning)**.** *A reinforcement learning (RL) task, denoted by $\mathfrak{t}_{\text{RL}}$ is a task $((\mathfrak{R}_{\text{RL}}, \mathfrak{S}_{\text{RL}}), \mathfrak{P}_{\text{RL}})$, where,*

1. *$\mathcal{S}, \mathcal{A}, \mathbb{R}, \{\gamma\}, \Pi \in \mathfrak{R}_{\text{RL}}$, for $\Pi := \{\pi | \pi : \mathcal{S} \to \mathcal{A}\}$,*
2. *$\mathcal{T}, \mathcal{R} \in \mathfrak{S}_{\text{RL}}$ such that,*

$$
\begin{aligned}
\mathcal{T} &: \mathcal{S} \times \mathcal{A} \to \mathcal{S} \\
\mathcal{R} &: \mathcal{S} \to \mathbb{R},
\end{aligned}
$$

3. *and, for the MDP defined by $\mathfrak{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, the data-generating process can be written as,*

$$
\begin{aligned}
\mathfrak{P}_{\text{RL}} &: \mathcal{S} \times \Pi \to \mathcal{O}_{\text{RL}}^{\infty} \\
&(s_0, \pi) \mapsto \mathfrak{P}_{\text{RL}}(s_0, \pi) = f_{\text{roll}}^{\pi}(s_0).
\end{aligned}
$$

[9] If these are known, the setup is called a *planning problem.*

Π is a space of possible policies. For example, this could be the space of all maps from the state space to the action space. From this definition, we see that the particular structure of a RL task is given by the transition dynamics, the reward function and the discount factor.

As a final note, the typical definition of a RL problem is made probabilistically. The value function is defined as an expectation over possible trajectories.

2.6 Spaces of Tasks

An important aspect of the structural definition of a task is that it explicitly outlines the maps that are interesting to us, regardless of the type of a ML problem. From this, we can describe *variations* to a task in terms of variations to the structure maps. All such variations can be collected together in a *space of tasks*.

As an example, consider the regression task $\mathfrak{t}$ that was described in Section 2.5.1, the definitions of which we repeat below.

$$\mathfrak{t}_{\text{reg}} = ((\mathfrak{R}_{\text{lin}}, \mathfrak{S}_{\text{lin}}), \mathfrak{P}_{\text{reg}}), \tag{2.12}$$

where,

$$\begin{aligned}
&\mathfrak{R}_{\text{reg}} = \{\mathbb{R}\},\\
&\mathfrak{S}_{\text{reg}} = \{f : \mathbb{R} \to \mathbb{R}; f(x) = ax + b, \text{id}_{\mathbb{R}} : \mathbb{R} \to \mathbb{R}\},\\
&\mathfrak{P}_{\text{reg}} : \mathbb{R} \to \mathbb{R} \times \mathbb{R}\\
&\qquad x \mapsto \mathfrak{P}_{\text{reg}}(x) = (x, f(x)),
\end{aligned}$$

where $a, b \in \mathbb{R}$. Since the structure map f is specifically defined for an a and b, we can vary the task by changing these values. That is, we can, as an example, define a task $\mathfrak{t}_{a_1,b_2}$ with a structure map $f = a_1x + b_1$, and a task $\mathfrak{t}_{a_2,b_2}$ with a structure map $f = a_2x + b_2$, where $a_1 \neq a_2$ and $b_1 \neq b_2$. We can do this for any value of a, b from $\mathbb{R}$. In varying this way, the collection of all such maps is the space of affine maps between $\mathbb{R}$ and $\mathbb{R}$.

We can formalise notions of variations of systems, and by extension tasks using a structural theory too.

Variations of Systems. Consider the variations to the structure maps above. Since we varied the structure maps affinely, the collection of structure maps is a space of affine maps $\mathcal{F}_{\text{aff}} = \{f_{a,b} : \mathbb{R} \to \mathbb{R}; f_{a,b}(x) = ax + b \,\big|\, a, b \in \mathbb{R}\}$. We can describe $\mathcal{F}_{\text{aff}}$ in a system:

$$\mathfrak{z}_{\text{aff}} = (\mathfrak{R}_{\text{aff}}, \mathfrak{S}_{\text{aff}}), \tag{2.13}$$

where,

$$\begin{aligned}
&\mathfrak{R}_{\text{aff}} = \{\mathbb{R}, \mathcal{F}_{\text{aff}}\},\\
&\mathfrak{S}_{\text{aff}} = \{f : \mathcal{F}_{\text{aff}} \times \mathbb{R} \to \mathbb{R}; (f_{a,b}, x) \mapsto f(f_{a,b}, x) = f_{a,b}(x) = ax + b\}.
\end{aligned}$$

f is an evaluation functional that evaluates its first argument at the point given by its second argument. $\mathbb{R}$ is the appropriate vector space system. A system with this format is a *system of functions.*

Definition of a System of functions

Definition 2.6.1 (System of functions)**.** *A system* $(\mathfrak{R}, \mathfrak{S})$ *is called a system of functions if:*

a) $\mathfrak{R}$ *contains at most one collection of* $\mathcal{F}, X, Y$*, where any* $f \in \mathcal{F}$ *is a map,*

$$f : X \to Y,$$

b) for $\mathcal{F}, X, Y \in \mathfrak{R}$*, as above, there exists an evaluation functional* $f_{\mathcal{F}} \in \mathfrak{S}$*, where,*

$$\begin{aligned} f_{\mathcal{F}} : \mathcal{F} \times X &\to Y \\ (f, x) &\mapsto f_{\mathcal{F}}(f, x) = f(x). \end{aligned}$$

A system of functions can contain additional structure maps. As another example, consider a system of continuous functions between topological systems $((X, \mathcal{O}_X), \mathfrak{S}_X)$ and $((Y, \mathcal{O}_Y), \mathfrak{S}_Y)$;

$$\mathfrak{z}_{\text{con}} = (\mathfrak{R}_{\text{con}}, \mathfrak{S}_{\text{con}}), \tag{2.14}$$

where,

$$\begin{aligned} \mathfrak{R}_{\text{con}} &= \{X, Y, \mathcal{F}_{\text{con}}\}, \\ \mathfrak{S}_{\text{con}} &= \{f_{\mathcal{F}} : \mathcal{F} \times X \to Y, \\ &\qquad \text{con}_{\mathcal{F}_{\text{con}}} : \mathcal{F}_{\text{con}} \times \mathcal{O}_Y \to \{1\} \\ &\qquad\qquad (f, U_Y) \mapsto \text{con}_{\mathcal{F}}(f, U_Y) = \mathbb{1}_{\mathcal{O}_X} \circ \text{preim}_f(U_Y) = 1. \end{aligned}$$

$\text{con}_{\mathcal{F}_{\text{con}}}$ constrains the functions in $\mathcal{F}_{\text{con}}$ to satisfy the requirements of a continuous function.

We constrain a system of functions to contain a single function space for simplicity. Furthermore, X and Y generally describe the domain and codomains of the functions in $\mathcal{F}$; we will assume that a system of functions can *always* be defined such that these spaces can be written in this way.

Systems of functions can be used to derive systems with structures that are contained in the function spaces $\mathcal{F}$ of the systems of functions;

Definition of a Derived system

Definition 2.6.2 (Derived system)**.**

A system $(\mathfrak{R}_{\text{der}}, \mathfrak{S}_{\text{der}})$ *is derived from systems of functions* $\{(\mathfrak{R}_i, \mathfrak{S}_i)\}_{i \in \mathcal{I}}$ *if:*

a) $\forall (\mathfrak{R}_i, \mathfrak{S}_i) \in \{(\mathfrak{R}_i, \mathfrak{S}_i)\}_{i \in \mathcal{I}}$*, with* $\mathcal{F}_i, X_i, Y_i \in \mathfrak{R}_i$*, where* $(f_i \in \mathcal{F}_i) : X_i \to Y_i$*,*

$$X_i, Y_i \in \mathfrak{R}_{\text{der}},$$

and,

b) $\forall(\mathfrak{R}_i, \mathfrak{S}_i) \in \{(\mathfrak{R}_i, \mathfrak{S}_i)\}_{i\in\mathcal{I}}$, *with* $\mathcal{F}_i, X_i, Y_i \in \mathfrak{R}_i$, *there exists a single* $f_i \in \mathcal{F}_i$, *and a single, not necessarily unique*[10] $f_{\text{der}} \in \mathfrak{S}_{\text{der}}$ *that can be decomposed as,*

$$f_{\text{der}} = ... \circ f_i \circ ...$$

Further, we say that $(\mathfrak{R}_i, \mathfrak{S}_i)$ affects *the derived system at* $f_{\text{der}} \in \mathfrak{S}$ *by* $f_i \in \mathcal{F}_i$.

Thus, a derived system is a tuple,

$$((\mathfrak{R}_{\text{der}}, \mathfrak{S}_{\text{der}}), \{(\mathfrak{R}_i, \mathfrak{S}_i)\}_{i\in\mathcal{I}}).$$

Condition (b) means that the system of functions is used to construct the structure of the derived system via function composition. A system can be derived from multiple systems of functions, even repeatedly by the same system of functions, as long as each affects a unique position of the decomposition of a single structure map. For example, the system $(\mathfrak{R}_{\text{lin}}, \mathfrak{S}_{\text{lin}})$ is *derived* from the system of functions $\mathfrak{z}_{\text{aff}}$, since one of its structure maps is an element of the system in $\mathfrak{R}_{\text{aff}}$.

Crucially, a derived system provides a principled mechanism for defining *variations* of a system.

Definition 2.6.3 (Variation of a Derived system)**.** *Suppose we are given a derived system* $((\mathfrak{R}_{\text{der}}, \mathfrak{S}_{\text{der}}), \{(\mathfrak{R}_i, \mathfrak{S}_i)\}_{i\in\mathcal{I}})$, *with system* $\mathfrak{z} = (\mathfrak{R}_{\text{der}}, \mathfrak{S}_{\text{der}})$ *that is derived from systems of functions* $\{(\mathfrak{R}_i, \mathfrak{S}_i)\}_{i\in\mathcal{I}}$. Definition of a Variation of a Derived system

Further, suppose that $(\mathfrak{R}_i, \mathfrak{S}_i) \in \{(\mathfrak{R}_i, \mathfrak{S}_i)\}_{i\in\mathcal{I}}$ *affects* $\mathfrak{z}$ *at* $f_{\text{der}} \in \mathfrak{S}$ *by* $f_i \in \mathcal{F}_i$. *That is,*

$$f_{\text{der}} = f^{\text{pre}} \circ f_i \circ f^{\text{suf}},$$

for some prefix and suffix functions f^{pre} *and* f^{suf} *respectively.*

Then, a variation $\mathfrak{z}'_{(\mathfrak{R}_i, \mathfrak{S}_i)}$ *of* $\mathfrak{z}$ *w.r.t.* $(\mathfrak{R}_i, \mathfrak{S}_i) \in \{(\mathfrak{R}_i, \mathfrak{S}_i)\}_{i\in\mathcal{I}}$ *is a new derived system* $(\mathfrak{R}'_{\text{der}}, \mathfrak{S}'_{\text{der}})$ *where,*

a) $\mathfrak{R}'_{\text{der}} = \mathfrak{R}_{\text{der}}$,

b) $\exists f'_{\text{der}} \in \mathfrak{S}'_{\text{der}}$, *where for* $(f'_i \in \mathcal{F}_i) \neq f_i$, *can be decomposed as,*

$$f'_{\text{der}} = f^{\text{pre}} \circ f'_i \circ f^{\text{suf}},$$

and,

c) $\mathfrak{S}'_{\text{der}} \backslash \{f'_i\} = \mathfrak{S}_{\text{der}} \backslash \{f_i\}$.

In words, a variation of a derived system is a change to its structure, relative to a particular system of functions that was used to derive it. When we described the affine space of functions above, there was a sense of structural similarity

[10] We mean that a single structure map could be affected by different systems of functions. However, each system of functions affects a single structure map.

between each particular variation. This definition formalises the notion of structural similarity in terms of the system of functions; in Equation 2.13, we could have varied $f(x) = ax + b$ as $f'(x) = ax + b + sin(c)$, where $c \in (0, \pi)$. In this case, the system of functions used is very different.

A derived system can be derived from many systems of functions, and each such system can vary the derived system. Thus, a product space $\overline{\mathcal{V}}_{\mathfrak{z}_{\text{der}}}$ of all possible variations of a derived system $\mathfrak{z}_{\text{der}}$ is,

$$\overline{\mathcal{V}}_{\mathfrak{z}_{\text{der}}} = ... \times \mathcal{F}_i \times ...$$

for $i \in \mathcal{I}$.

For $\overline{v} \in \overline{\mathcal{V}}_{\mathfrak{z}_{\text{der}}}$, a system that has this particular variation is denoted by $\mathfrak{z}_{\text{der}}^{\overline{v}}$. $\mathfrak{z}_{\text{der}}^{\overline{v}}$ is the result of a combination of individual variations w.r.t. each system of functions it is derived from. However, there can be situations where two such total variations result in equal derived systems, in that their structures are equal to each other. Thus, an equivalence relation $\sim_{\mathfrak{z}_{\text{der}}}$ can be defined, where for $\overline{v_1}, \overline{v_1} \in \overline{\mathcal{V}}_{\mathfrak{z}_{\text{der}}}$,

$$\overline{v_1} \sim_{\mathfrak{z}_{\text{der}}} \overline{v_2} \iff \mathfrak{z}_{\text{der}}^{\overline{v_1}} = \mathfrak{z}_{\text{der}}^{\overline{v_2}}. \tag{2.15}$$

Then,

Definition of a Total Variation Space of a Derived System

Definition 2.6.4 (Total Variation Space of a Derived System)**.** *Given a derived system* $\mathfrak{z}_{\text{der}} = \big((\mathfrak{R}_{\text{der}}, \mathfrak{S}_{\text{der}}), \{(\mathfrak{R}_i, \mathfrak{S}_i)\}_{i \in \mathcal{I}}\big)$*, the total variation space* $\mathcal{V}_{\mathfrak{z}_{\text{der}}}$ *is the quotient space,*

$$\mathcal{V}_{\mathfrak{z}_{\text{der}}} = \overline{\mathcal{V}}_{\mathfrak{z}_{\text{der}}} / \sim_{\mathfrak{z}_{\text{der}}} .$$

A particular $v \in \mathcal{V}$ *is called a total variation*[11]*, and the system derived from* v *is denoted as* $\mathfrak{z}_{\text{der}}^{v}$*.*

Variations of Tasks. The additional property that a task has, compared to a system is the data-generating process. Furthermore, the structure of a task is stripped to only contain the structure maps that are used in its data-generating process. Thus a task can be derived from systems of functions in the same way as a system, if the systems of functions under consideration affect the structure of the task in a similar manner. We will denote a derived task as the tuple,

$$\big(\big((\mathfrak{R}_{\text{der}}, \mathfrak{S}_{\text{der}}), \{(\mathfrak{R}_i, \mathfrak{S}_i)\}_{i \in \mathcal{I}}\big), \mathfrak{P}\big). \tag{2.16}$$

That is, it is composed of a derived system $\mathfrak{z} = \big((\mathfrak{R}_{\text{der}}, \mathfrak{S}_{\text{der}})$, and a data-generating process $\mathfrak{P}$ that is applicable to this system.

Variations of derived tasks are trickier. From a machine learning perspective, the key aspects of a task are captured by its data-generating process. Thus, two tasks are equal to each other, if their data-generating processes are functionally equal[12]. Simply comparing the structures of tasks is insufficient, since the

[11] This is not the statistical notion of a total variation.

[12] That is, their outputs match for all values of their inputs (in the deterministic case), or their induced measures are functionally equal (in the probabilistic case).

composition of structure maps could lead to additional equivalences. As with derived systems, the changes to variations of tasks are in their structures, which in turn affect the data-generating processes. There are no changes made to the relata or their representations. We write the product space of all task variations $\overline{\mathfrak{T}}_{\mathfrak{z}}$ as,

$$\overline{\mathfrak{T}}_{\mathfrak{z}} = ... \times \mathcal{F}_i \times ...$$

for $i \in \mathcal{I}$. If $\mathfrak{t}^{\bar{t}}$ denotes the derived task for the variation $\bar{t} \in \overline{\mathfrak{T}}_{\mathfrak{z}}$, then the equivalence $\sim_{\mathfrak{t}}$ between two such variations $\overline{t_1}, \overline{t_2} \in \overline{\mathfrak{T}}_{\mathfrak{z}}$ is,

$$\overline{t_1} \sim_{\mathfrak{t}} \overline{t_2} \iff \mathfrak{P}^{\overline{t_1}} = \mathfrak{P}^{\overline{t_2}}, \tag{2.17}$$

where $\mathfrak{P}^{\overline{t_i}}$ denotes the data-generating process of the derived task $\mathfrak{t}^{\overline{t_i}}$. Then,

Definition of a Task Space

Definition 2.6.5 (Task Space)**.** *Given a derived system,*

$$\mathfrak{z}_{\text{der}} = ((\mathfrak{R}_{\text{der}}, \mathfrak{S}_{\text{der}}), \{(\mathfrak{R}_i, \mathfrak{S}_i)\}_{i \in \mathcal{I}}),$$

and a valid data-generating process $\mathfrak{P}$, the task space $\mathfrak{T}_{\mathfrak{z},\mathfrak{P}}$ is the quotient space,

$$\mathfrak{T}_{\mathfrak{z},\mathfrak{P}} = \overline{\mathfrak{T}}_{\mathfrak{z}} / \sim_{\mathfrak{t}} .$$

The key difference between a task space and the total variation space of a derived system is that the equivalence relation is different.

Consider that we are given a task space $\mathfrak{T}_{\mathfrak{z},\mathfrak{P}}$. Each variation in $\mathfrak{T}_{\mathfrak{z},\mathfrak{P}}$ uniquely identifies a task. This is because of the quotient we took in Definition 2.6.5. Thus, If we agree on $\mathfrak{z}$ and $\mathfrak{P}$, then in this context, a task $\mathfrak{t}$ can be defined as being an element of $\mathfrak{T}_{\mathfrak{z},\mathfrak{P}}$.

2.6.1 Task spaces as Smooth Manifolds

The intrinsic structure of a space of tasks must depend on the intrinsic structures of the systems of functions that were used to derive it, as well as the induced equivalence relation. For example, consider again the task $\mathfrak{t}_{\text{reg}}$, as per Equation (2.12). If this task was derived and varied w.r.t. the affine system in Equation (2.13), then the task space $\mathfrak{T}_{\text{reg}}$ itself is given by $\mathbb{R}^2$, since any $(a, b) \in \mathbb{R}^2$ defines a unique structure map, and for any $(a_1, b_1), (a_2, b_2) \in \mathbb{R}^2$, where at least $a_1 \neq a_2$ or $b_1 \neq b_2$, we can derive a unique task.

By saying this, we also imply that $\mathfrak{T}_{\text{reg}}$ also carries with it the topological properties of $\mathbb{R}^2$. As a counterexample, consider the supervised learning task of a shifted sinusoid $\mathfrak{t}_{\text{sin}}$,

$$\mathfrak{t}_{\text{sin}} = ((\mathfrak{R}_{\text{sin}}, \mathfrak{S}_{\text{sin}}), \mathfrak{P}_{\text{sin}}),$$

where,

$$\begin{aligned}
&\mathfrak{R}_{\text{sin}} = \{\mathbb{R}, [-1, -1]\}, \\
&\mathfrak{S}_{\text{sin}} = \{f_{\text{sin}} : \mathbb{R} \to [-1, -1]; f_{\text{sin}}(x) = \sin(x + a), \\
&\mathfrak{P}_{\text{sin}} : \mathbb{R} \to \mathbb{R} \times [-1, -1] \\
&\qquad x \mapsto \mathfrak{P}(x) = (x, f_{\text{sin}}(x)).
\end{aligned}$$

Due to the periodicity of the sin function, for any $a \in \mathbb{R}$, we have that the task for which $a_k = a + 2k\pi$, for $k \in \mathbb{Z}$, is equivalent to the original task, since their respective data-generating processes cannot distinguish between them. The task space $\mathfrak{T}_{\text{sin}}$ is the topological space given by the circle $\mathbb{S}$.

In both cases, the resulting task space can also be represented by a topological manifold. In the present work, we consider task spaces that can be represented by smooth manifolds; these are topological manifolds that have a smooth structure. Our reasons for making this assumption are:

Topological and Smooth Structures. When we start discussing problems of transfer, it will become evident that having a notion of a topology on a space of tasks is useful. A topology allows us to describe what neighbourhoods of tasks would look like. In addition, it will give us a notion of *continuous* transformations of tasks. The topological structure of a task space can be derived from its definition; that is the quotient topology of the product topology over the functions spaces that derive it; this is assuming that the function spaces themselves are equipped with topological structures.

In later sections, we will see that it might be necessary to take derivatives, particularly when discussing algorithms for learning to transfer. The smooth structure of a smooth manifold affords us this luxury.

Finiteness. The dimensionality of the manifold describes the *finite* number of degrees of freedom we have to move in to describe different tasks in the task space. This means that we will be considering task spaces that can be varied along a finite number of directions. This is opposed to a universal set of tasks that contains *all possible* learning tasks; we believe that this is too general to consider[13].

The finiteness of the task space could be thought of as pathological; for example, it is possible (albeit unwieldy) to consider the space of all continuous functions as a task space. However, we propose that given some useful structure, and therefore bias, we will often find subspaces of this full set which are indeed finite dimensional, and form a topological manifold. Certainly, it is possible to find such subspaces, and to therefore restrict our attention to solving learning tasks from such spaces.

We mentioned in Section 2.6 that the elements of a task space themselves provide the most fundamental level of structure, since they are chosen from the set of all possible tasks to belong to the given task space $\mathfrak{T}$. For example, if we define the task space to be all regression tasks generated from scalar valued truncated Fourier series on $\mathbb{R}$, we immediately note there is a constraint (and therefore structure) that defines the task space.

[13]There does exist an obvious counter example to this however: a reasonable set of tasks that one could consider in a regression task is the set of all continuous functions. This is technically *infinite* dimensional. While we believe that it is possible that our theory could be extended to this case, we will restrict ourselves to the finite dimensional scenario for the time being.

Independence of representation. In Example 5.1.5, we show that a smooth (topological) manifold is a natural representation of certain topological spaces. An important note here is that any theories that we would write w.r.t. this space are independent of this representation. This is good for us, since we can then think about task spaces abstractly, and then realise them using an appropriate atlas, when necessary.

Remark 2.6.1: From Remark 2.1.1, the space of tasks can itself be seen as a system, since we have now assumed that it is a smooth manifold. •

This page was left intentionally blank.

Chapter 3

Models and Symmetries

A task is a structured system in a space of tasks with some unknown structure that we wish to learn. To do so, we need to *describe* this structure meaningfully. A parametric model[1] defines a class of input-output maps in terms of a model architecture and a parameter space. Together, these provide us with a language to describe our structure.

In this chapter, we will be defining these terms. In addition, we will see how to create a model space which accounts for any symmetries in the generated input-output maps. We will investigate the symmetries of a simple neural network, and conclude with a brief discussion of a novel tool that can help us visualise the symmetries of a neural network.

3.1 Model Architectures

Parametric models are etymologically based on a space of parameters Θ. Along with a parameter space, a parametric model is equipped with a model architecture.

Definition 3.1.1 (Model Architecture)**.** *Given a parameter space Θ, and spaces X and Y, a model architecture κ is a map,* Definition of a Model Architecture

$$\begin{aligned} \kappa : \Theta \times X &\to Y \\ (\theta, x) &\mapsto \kappa(\theta, x). \end{aligned}$$

For a particular $\theta \in \Theta$, the map,

$$\begin{aligned} \kappa_\theta : X &\to Y \\ x &\mapsto \kappa_\theta(x) = \kappa(\theta, x), \end{aligned}$$

is called the architecture map of θ.

The model architecture provides a way of constructing an input-output map between two spaces, using a parameter space Θ. For a particular $\theta \in \Theta$, the

[1] Non-parametric models are trickier to describe, and will therefore be deferred to future work.

architecture map κ_θ is the particular input-output map that is induced by the model architecture.

Example 3.1.1 Linear architecture: Consider $\Theta = \mathbb{R}^d$. For input space $\mathbb{R}^d$ and output space $\mathbb{R}$, we can define an architecture κ_{lin} which creates linear maps as,

$$\begin{aligned} \kappa_{\text{lin}} : \mathbb{R}^d \times \mathbb{R}^d &\to \mathbb{R} \\ (\theta, x) &\mapsto \kappa_{\text{lin}}(\theta, x) = \theta^\intercal x. \end{aligned} \tag{3.1}$$

▲

Example 3.1.2 Sinusoidal architecture: Consider $\Theta_{\text{sin}} = \mathbb{R}^2$. An example architecture on this space, for input space $\mathbb{R}$ and output space $\mathbb{R}$ is the architecture κ_{sin},

$$\begin{aligned} \kappa_{\text{sin}} : \mathbb{R}^2 \times \mathbb{R} &\to \mathbb{R} \\ ((a, b), x) &\mapsto \kappa_{\text{sin}}((a, b), x) = a + \sin(x + b). \end{aligned} \tag{3.2}$$

▲

Example 3.1.3 Neural Network architecture: In this example, we will construct a single layer, 2-node neural network architecture. The non-linearity we choose is the Rectified Linear Unit (ReLU), which is given by:

$$\begin{aligned} \sigma_{\text{ReLU}} : \mathbb{R} &\to \mathbb{R} \\ x \mapsto \sigma_{\text{ReLU}}(x) &= \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}. \end{aligned} \tag{3.3}$$

Notation. If we write $\sigma_{\text{ReLU}}(v)$, where $\mathbf{v} \in \mathbb{R}^d$, then σ_{ReLU} is applied point-wise to each coordinate. That is, for $(v_1, ..., v_d) = \mathbf{v} \in \mathbb{R}^d$, where $v_i \in \mathbb{R}$,

$$\sigma_{\text{ReLU}}((v_1, ..., v_d)) = (\sigma_{\text{ReLU}}(v_1), ..., \sigma_{\text{ReLU}}(v_d)).$$

Suppose that $X = \mathbb{R}^d$ and $Y = \mathbb{R}$. Then, $\Theta_{\text{ReLU}} = \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}$, giving κ_{ReLU} as,

$$\begin{aligned} \kappa_{\text{ReLU}} : \Theta_{\text{ReLU}} \times \mathbb{R}^d &\to \mathbb{R} \\ (\mathbf{v_1}, \mathbf{v_2}, w_1, w_2, x) &\mapsto \kappa_{\text{ReLU}}(\mathbf{v_1}, \mathbf{v_2}, w_1, w_2, x) \\ &= \underbrace{w_1 \sigma_{\text{ReLU}}(\mathbf{v_1}^\intercal x)}_{\text{node 1}} + \underbrace{w_2 \sigma_{\text{ReLU}}(\mathbf{v_2}^\intercal x)}_{\text{node 2}}. \end{aligned} \tag{3.4}$$

Other neural network architectures can be defined by varying the number of layers, nodes, and the non-linearity that is used; standard non-linearities include the tanh function and the sigmoid function. These networks are classified as fully-connected networks; more specialised neural network architectures include the Convolutional Neural Network (CNNs) [57], Residual Networks (ResNet) [38], and Recurrent Neural Networks (RNNs) [83]. ▲

3.2 Symmetries in Architectures

Given a model architecture, with its associated input and output spaces, and its parameter space, a question that can be asked is whether each parameter $\theta \in \Theta$ *uniquely* identifies an input-output map.

Definition of a Equivalence of Parameters

Definition 3.2.1 (Equivalence of Parameters)**.** *Given a model architecture $\kappa : \Theta \times X \to Y$, an equivalence relation $\sim_\kappa$ is defined between $\theta_1, \theta_2 \in \Theta$. That is,*

$$\theta_1 \sim_\kappa \theta_2 \iff \kappa_{\theta_1}(x) = \kappa_{\theta_2}(x),\ \forall x \in X.$$

The question of whether architecture maps are equivalent is called the identifiability problem, and has been studied for neural networks [103, 59, 116]. In example Example 3.1.1, each parameter vector produces a unique input-output map; this follows from basic linear algebra. However, looking at Examples 3.1.2 and 3.1.3, this is clearly not true. In Example 3.1.2, for $(a, b) \in \Theta_{\text{sin}}$, we know by the periodicity of the sin function that,

$$\kappa_{\text{sin}}((a, b + 2k\pi), \cdot) = \kappa_{\text{sin}}((a, b), \cdot),$$

for $k \in \mathbb{Z}$.

In the case of κ_{ReLU}, non-identifiability occurs in three respects. Firstly, σ_{ReLU} obeys *non-negative homogeneity* [24]. That is, for $c \in \mathbb{R}^+ \backslash \{0\}$,

$$\sigma_{\text{ReLU}}(cx) = c\sigma_{\text{ReLU}}(x),\ \forall x \in \mathbb{R}. \tag{3.5}$$

Thus, in the case of Θ_{ReLU}, we have that for any $c_1, c_2 \in \mathbb{R}^+ \backslash \{0\}$,

$$\begin{aligned}
\kappa_{\text{ReLU}}(c_1\mathbf{v_1}, c_2\mathbf{v_2}, w_1, w_2, x) &= w_1\sigma_{\text{ReLU}}((c_1\mathbf{v_1})^\intercal x) + w_2\sigma_{\text{ReLU}}((c_2\mathbf{v_2})^\intercal x) \\
&= w_1\sigma_{\text{ReLU}}(c_1(\mathbf{v_1}^\intercal x)) + w_2\sigma_{\text{ReLU}}(c_2(\mathbf{v_2}^\intercal x)) \\
&= c_1 w_1\sigma_{\text{ReLU}}(\mathbf{v_1}^\intercal x) + c_2 w_2\sigma_{\text{ReLU}}(v_\intercal x) \\
&= \kappa_{\text{ReLU}}(\mathbf{v_1}, \mathbf{v_2}, c_1 w_1, c_2 w_2, x),\ \forall x \in X.
\end{aligned}$$

Equivalently,

$$\kappa_{\text{ReLU}}(\mathbf{v_1}, \mathbf{v_2}, w_1, w_2, x) = \kappa_{\text{ReLU}}(c_1\mathbf{v_1}, c_2\mathbf{v_2}, c_1^{-1}w_1, c_2^{-1}w_2, x),\ \forall x \in X.$$

Notation. Since for $i \in \{1, 2\}$, $c_i \in \mathbb{R}$, the notation $c_i v_i$ denotes an elementwise multiplication of a scalar and a vector.

The second source of non-identifiability is due to the fact that addition is associative. That is, we have that,

$$\kappa_{\text{ReLU}}(\mathbf{v_1}, \mathbf{v_2}, w_1, w_2, x) = \kappa_{\text{ReLU}}(\mathbf{v_2}, \mathbf{v_1}, w_2, w_1, x),\ \forall x \in X$$

since,

$$\begin{aligned}\kappa_{\mathrm{ReLU}}(\mathbf{v_1}, \mathbf{v_2}, w_1, w_2, x) &= w_1{}^{\intercal}\sigma_{\mathrm{ReLU}}(\mathbf{v_1}{}^{\intercal}x) + w_2{}^{\intercal}\sigma_{\mathrm{ReLU}}(\mathbf{v_2}{}^{\intercal}x) \\ &= w_2{}^{\intercal}\sigma_{\mathrm{ReLU}}(\mathbf{v_2}{}^{\intercal}x) + w_1{}^{\intercal}\sigma_{\mathrm{ReLU}}(\mathbf{v_1}{}^{\intercal}x) \\ &= \kappa_{\mathrm{ReLU}}(\mathbf{v_2}, \mathbf{v_1}, w_2, w_1, x),\ \forall x \in X.\end{aligned}$$

Thus, nodes can be permuted.

Finally, the third source occurs when any of $\mathbf{v_1}, \mathbf{v_2}, w_1$, or w_2 is equal to zero. If $w_1 = 0$, then the input-output map for any $\mathbf{v_1} \in \mathbb{R}^d$ is non-identifiable. This is similarly true for any $w_1 \in \mathbb{R}$ if $\mathbf{v_1} = 0$. This can be extended to the pair $\mathbf{v_2}$ and w_2. This source is due to the *reducibility* [103] of the network when any of $\mathbf{v_1}, \mathbf{v_2}, w_1$, or w_2 is equal to zero, or when $\mathbf{v_1} = c\mathbf{v_2}$, for $c \in \mathbb{R}^+\backslash\{0\}$.

A network is said to be reducible if there is a *smaller* network (in the sense of the number of nodes) that is equivalent to it. For example, when $w_1 = 0$ or $\mathbf{v_1} = 0$, we can effectively remove node 1[2] since,

$$\begin{aligned}\kappa_{\mathrm{ReLU}}(\mathbf{v_1}, \mathbf{v_2}, 0, w_2, x) &= \kappa_{\mathrm{ReLU}}(0, \mathbf{v_2}, w_1, w_2, x) \\ &= w_2\sigma_{\mathrm{ReLU}}(\mathbf{v_2}{}^{\intercal}x),\ \forall x \in X.\end{aligned}$$

Furthermore, if $\mathbf{v_1} = c\mathbf{v_2}$, for $c \in \mathbb{R}^+\backslash\{0\}$, then, due to the non-negative homogeneity,

$$\begin{aligned}\kappa_{\mathrm{ReLU}}(\mathbf{v_1}, \mathbf{v_2}, w_1, w_2, x) &= \kappa_{\mathrm{ReLU}}(c\mathbf{v_2}, \mathbf{v_2}, w_1, w_2, x) \\ &= cw_1\sigma_{\mathrm{ReLU}}(\mathbf{v_2}{}^{\intercal}x) + w_2\sigma_{\mathrm{ReLU}}(\mathbf{v_2}{}^{\intercal}x) \\ &= (cw_1 + w_2)\sigma_{\mathrm{ReLU}}(\mathbf{v_2}{}^{\intercal}x),\ \forall x \in X,\end{aligned}$$

where we can now have a single node with parameters $(\mathbf{v_2}, cw_1 + w_2)$. Similarly, we could also have a single node $(\mathbf{v_1}, w_1 + c^{-1}w_2)$.

Some of these symmetries can be visualised in Figure 3.1.

Knowing the *unique* architecture maps of a given model architecture is useful when attempting to study models for machine learning. This becomes especially clear when we wish to place additional structures on model architecture, as we will see in Section 7.1. Thus,

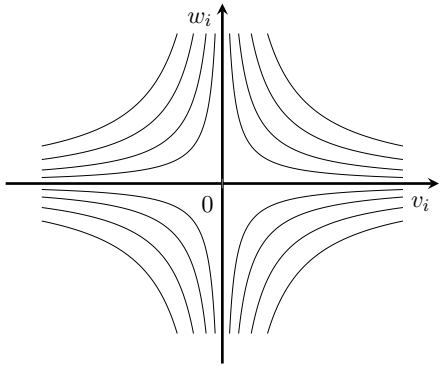
Definition of a Parametric Model Space

Definition 3.2.2 (Parametric Model Space)**.** *Given a model architecture $\kappa : \Theta \times X \to Y$, the parametric model space Θ_κ is,*

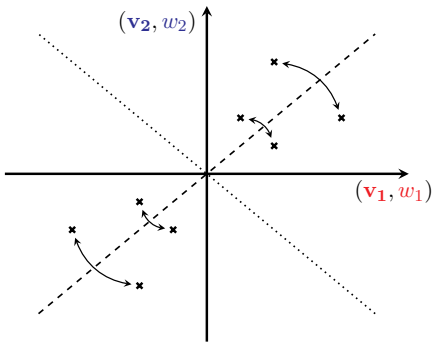
$$\Theta_\kappa = \Theta/\sim_\kappa,$$

where $\sim_\kappa$ is the equivalence between parameters in Θ, w.r.t. the architecture.

[2] It is interesting to note that in a sense, residual networks are specially designed to allow this. A ResNet makes it easier for nodes to *learn* identity maps [38], by allowing non-linear components to drop to zero, thus effectively removing this node, and allowing only the residual component to carry on.



(a) $i \in \{1, 2\}$. Each connected line depicts a set of equivalent parameter values due to non-negative homogeneity. Both axes represent the reducibility symmetry due to any of $\mathbf{v_1}, \mathbf{v_2}, w_1, w_2 = 0$.



(b) Swapping weights along the dashed line yields equivalent parameter values. Furthermore, both dotted and dashed lines denote reducibility symmetries due to $(\mathbf{v_1}, w_1) = (\mathbf{v_2}, w_2)$.

Figure 3.1. Visualisation of the symmetries of a single layer, 2 node, ReLU neural network, as in Example 3.1.3.

A parametric model space is a space of parameters that correspond to *unique*, *identifiable* architecture maps. As with spaces of tasks, in the present work, we will express parametric model spaces as smooth manifolds.

The question of whether this is possible is closely tied to symmetries in the architecture. When there are no symmetries, such as in the parametric model space of Example 3.1.1 is simply $\mathbb{R}^d$, the parameter space and parametric model space are isomorphic. If they are not, it becomes more complicated, as we will see.

3.3 Parametric model spaces as smooth manifolds

It is typical to consider symmetries in terms of groups and group actions. A symmetry group of space X is a group $\mathcal{G}_X$, and a (left) group action $\rho_{\mathcal{G}_X} : \mathcal{G}_X \times X \to X$ on X that, for each $g \in \mathcal{G}_X$ specifies a transformation,

$$\begin{aligned} \rho_g : X &\to X \\ x &\mapsto \rho_g(x) = \rho_{\mathcal{G}_X}(g, x) \end{aligned}$$

on X that obeys the consistency requirements of a group action. That is, for $g_1, g_2 \in \mathcal{G}_X$,

$$\rho_{g_2} \circ \rho_{g_1}(x) = \rho_{g_2 g_1}(x).$$

Points on X are said to be symmetric to each other if there exists such a transformation that maps them to each other; they are said to be in the same

orbit. Thus, the group action defines an equivalence relation which is equivalent to the equivalence relation $\sim_\kappa$.

The parametric model space of a model architecture is then the orbit space of the group action of the symmetry group. Thus, the question of whether the parametric model space is a smooth manifold is a question of whether this orbit space is a smooth manifold. The following theorem, from [58, *Chapter* 21][3] can help.

Quotient Manifold Theorem (Theorem 21.10 [58])

Theorem 3.3.1. *Suppose $\mathcal{G}$ is a Lie group acting smoothly, freely and properly on a smooth manifold X. Then the orbit space $X/\mathcal{G}$ is a topological manifold of dimension equal to* $\dim X - \dim \mathcal{G}$*, and has a unique smooth structure.*

This theorem implies that if we can show that the action of the symmetry group is smooth, free and proper (see Appendix A.4 for definitions), then the resulting orbit space can be given the structure of a smooth manifold. For general model architectures, showing this could prove difficult, perhaps impossible. However, it might be more reasonable to assume that there is some submanifold of the parameter space on which the appropriate quotient manifold exists, where the elements that are ignored have zero consequences[4] to the purpose of learning. Furthermore, it is expected that the resulting space will contain architecture maps that are very *close* in practical behaviour[5] to the points that we removed.

Symmetry group of the Sinusoidal Architecture

The symmetry group of Example 3.1.2 is the Lie group of the additive integers, the action of which translates the parameter b by $2\pi k$. That is,

$$\begin{aligned}&\mathcal{G}_{\text{sin}} = \mathbb{Z},\\ &\rho_{\mathcal{G}_{\text{sin}}} : \mathcal{G}_{\text{sin}} \times \Theta_{\text{sin}} \to \Theta_{\text{sin}}\\ &\qquad\qquad (k,(a,b)) \mapsto \rho_{\mathcal{G}_{\text{sin}}}(k,(a,b)) = (a, b+2\pi k).\end{aligned}$$

Each map $\rho_k : \mathbb{R} \to \mathbb{R}; b \mapsto \rho_k(b) = b+2\pi k$, for $k \in \mathbb{Z}$ is clearly a diffeomorphism on $\mathbb{R}$. In addition, since $\mathbb{Z}$ is discrete, $\rho_{\mathcal{G}_{\text{sin}}}$ is also smooth. According to the definition of a proper action (Definition A.4.4), it suffices to show that the map $(k,(a,b)) \mapsto (((a,b+2\pi k)),(a,b))$ is proper. By Proposition A.53(d) in [58], because $\mathbb{R}^2 \times \mathbb{R}^2$ is Hausdorff, and ρ_k^{-1} is continuous, $\rho_{\mathcal{G}_{\text{sin}}}$ is a proper action. It is also clear that this action is free. Thus, the parametric model space for κ_{sin} is a smooth manifold; in fact, this manifold is the cylinder space $\mathbb{R} \times \mathbb{S}^1$. Since the dimension of $\mathcal{G}_{\text{sin}}$ is 0, the dimension of the parametric model space should be 2, which is true for $\mathbb{R} \times \mathbb{S}^1$.

Symmetry group of a Neural Network Architecture

In Example 3.1.3, each symmetry we described forms its own group. In order to account for pathological cases, we will remove all points in Θ_{ReLU} where any

[3] We have reproduced only the relevant parts of the theorem, and have slightly changed the notation to match the present work.

[4] In the sense that, relative to stochastic gradient descent, they have zero measure.

[5] By practical behaviour, we mean that the differences, in the domain of X that we are interested is negligible. Further, this means that for any arbitrary measure of closeness, we can find an appropriate parameter value.

of $\mathbf{v_1}, \mathbf{v_2}, w_1, w_2 = 0$. Furthermore, we will remove all points where $\mathbf{v_1} = c\mathbf{v_2}$ for some constant $c \in \mathbb{R}^+\backslash\{0\}$. The following theorem is useful for the ensuing discussion.

Removing points in Θ_{ReLU} maintains diffeomorphisms

Theorem 3.3.2. *Given $\Theta_{\text{ReLU}} = \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}$, let us denote,*

$$\overline{\Theta}_{\text{ReLU}} = \Theta_{\text{ReLU}}\backslash(\Theta_0 \cup \Theta_=),$$

where,

$$\Theta_0 = \{(\mathbf{v_1}, \mathbf{v_2}, w_1, w_2) \mid \mathbf{v_1}, \mathbf{v_2}, w_1 \text{ or } w_2 = 0\}, \text{ and}$$
$$\Theta_= = \{(\mathbf{v_1}, \mathbf{v_2}, w_1, w_2) \mid \mathbf{v_1} = c\mathbf{v_2}, \forall c \in \mathbb{R}^+\backslash\{0\}\}.$$

Under the subset topology, $\overline{\Theta}_{\text{ReLU}}$ is an open submanifold of Θ_{ReLU}. Further, suppose that,

$$f : \Theta_{\text{ReLU}} \to \Theta_{\text{ReLU}}$$

is a diffeomorphism, and that,

$$f|_{\overline{\Theta}_{\text{ReLU}}} : \overline{\Theta}_{\text{ReLU}} \to \overline{\Theta}_{\text{ReLU}}$$

is well-defined, and is a bijection. Then, it follows that $f|_{\overline{\Theta}_{\text{ReLU}}}$ is also a diffeomorphism.

Proof. To prove that $\overline{\Theta}_{\text{ReLU}}$ is an open submanifold, it suffices to show that $\overline{\Theta}_{\text{ReLU}}$ is an open set in the standard topology of $\mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}$. We know that,

$$\Theta_{\text{ReLU}}\backslash(\Theta_0 \cup \Theta_=) = \Theta_{\text{ReLU}} \cap {\Theta_0}^{\mathsf{c}} \cap {\Theta_=}^{\mathsf{c}}.$$

Notation. ${\Theta_0}^{\mathsf{c}}$ and ${\Theta_=}^{\mathsf{c}}$ denote the complements of their respective sets.

If we write $(x_1, ...x_{2d+2}) \in (\Theta_{\text{ReLU}} = \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R} \times \mathbb{R})$, then from inspection,

$$\Theta_0 = \bigcup_{j=1}^{2d+2} \{(..., x_j, ...) \mid x_j = 0\}.$$

Each such $\{(..., x_j, ...) \mid x_j = 0\}$ is a closed subset of Θ_{ReLU}. Since the finite union of closed sets is closed, Θ_0 is closed. Thus, ${\Theta_0}^{\mathsf{c}}$ is open.

Consider now, the space,

$$\Theta^*_= = \{(\mathbf{v_1}, \mathbf{v_2}, w_1, w_2) \mid \mathbf{v_1} = c\mathbf{v_2}, \forall c \geq 0\}.$$

It is true that $\Theta^*_=\backslash\Theta_= \subset \Theta_0$. This is because,

$$\begin{aligned}(\Theta^*_=\backslash\Theta_=) \cup \{(0, 0, w_1, w_2)\} &= \{(\mathbf{v_1}, \mathbf{v_2}, w_1, w_2) \mid \mathbf{v_1} = c\mathbf{v_2}, c = 0\} \\ &= \{(0, \mathbf{v_2}, w_1, w_2)\} \\ &\subset \Theta_0.\end{aligned}$$

Thus,

$$\Theta_0 \cup \Theta_= = \Theta_0 \cup \Theta_=^*.$$

$\Theta_=^*$ is closed, since any convergent sequence in $\Theta_=^*$ must have its limit in $\Theta_=^*$; any sequence must be written as,

$$(v^1, c^1v^1, {w_1}^1, {w_2}^1), ..., (v^i, c^iv^i, {w_1}^i, {w_2}^i), ...$$

Since all values of $v \in \mathbb{R}^d$, $w_1, w_2 \in \mathbb{R}$ are included in $\Theta_=^*$, and $c = [0, \infty)$, which is closed, all limit points are included. Therefore $\Theta_=^{*\prime}$ is open.

Thus, $\overline{\Theta}_{\text{ReLU}}$ is an open set of Θ_{ReLU}, which provides us an open submanifold, by restricting the charts of Θ_{ReLU} to $\overline{\Theta}_{\text{ReLU}}$. The inclusion map $\mathfrak{i} : \overline{\Theta}_{\text{ReLU}} \hookrightarrow \Theta_{\text{ReLU}}$, is its smooth embedding.

In the second part, we know that for any $p \in \Theta_{\text{ReLU}}$, there exists charts (U, ϕ) where $p \in U$, and (V, ψ) where $f(p) \in V$, then, $\psi \circ f \circ \phi^{-1}$ is smooth. Then, suppose $\overline{p} \in \overline{\Theta}_{\text{ReLU}}$, and that $\mathfrak{i}(\overline{p}) = p$. Then, $(U \cap \overline{\Theta}_{\text{ReLU}}, \phi|_{U \cap \overline{\Theta}_{\text{ReLU}}})$ is a chart of $\overline{\Theta}_{\text{ReLU}}$. Similarly, we can write $f|_{\overline{\Theta}_{\text{ReLU}}}(\overline{p}) \in (V \cap \overline{\Theta}_{\text{ReLU}})$, with $\psi|_{V \cap \overline{\Theta}_{\text{ReLU}}}$ being its chart map.

Then, since $\mathfrak{i}$ and f are smooth, we can state that,

$$\psi|_{V \cap \overline{\Theta}_{\text{ReLU}}} \circ f|_{\overline{\Theta}_{\text{ReLU}}} \circ \phi^{-1}|_{U \cap \overline{\Theta}_{\text{ReLU}}} = \psi|_{V \cap \overline{\Theta}_{\text{ReLU}}} \circ f \circ \mathfrak{i} \circ \phi^{-1}|_{U \cap \overline{\Theta}_{\text{ReLU}}},$$

is smooth.

Finally, since $f|_{\overline{\Theta}_{\text{ReLU}}}$ is bijective, and f is a diffeomorphism, a similar argument can be made for the inverse being smooth. Thus, $f|_{\overline{\Theta}_{\text{ReLU}}}$ is a diffeomorphism. ■

Remark 3.3.1: This version of the theorem is strong. It can be weakened, where if f is smooth, or continuous, then $f|_{\overline{\Theta}_{\text{ReLU}}}$ is smooth, or continuous. •

Theorem 3.3.2 tells us that despite the points we had removed, Θ_{ReLU} is still a smooth manifold. Furthermore, any diffeomorphisms, smooth maps or continuous maps, when appropriately restricted to $\overline{\Theta}_{\text{ReLU}}$ will remain as diffeomorphisms, or smooth or continuous maps.

Notation. To avoid overbearing notation, in the following we will override Θ_{ReLU} to mean $\overline{\Theta}_{\text{ReLU}}$.

Symmetry group of non-negative homogeneity

We can now look at the symmetries. Firstly, the non-negative homogeneity property can be written as the action of the scaling group $\mathbb{R}^+\backslash\{0\} \times \mathbb{R}^+\backslash\{0\}$ acting on $\mathbf{v_1}, w_1$, and $\mathbf{v_2}, w_2$. That is,

$$\begin{aligned}
\mathcal{G}_{\text{nnh}} &= \mathbb{R}^+\backslash\{0\} \times \mathbb{R}^+\backslash\{0\}, \\
\rho_{\text{nnh}} &: \mathcal{G}_{\text{nnh}} \times \Theta_{\text{ReLU}} \to \Theta_{\text{ReLU}} \\
((c_1, c_2), (\mathbf{v_1}, \mathbf{v_2}, w_1, w_2)) &\mapsto \rho_{\text{nnh}}((c_1, c_2), (\mathbf{v_1}, \mathbf{v_2}, w_1, w_2)) \\
&= (c_1\mathbf{v_1}, c_2\mathbf{v_2}, c_1^{-1}w_1, c_2^{-1}w_2).
\end{aligned}$$

Regardless of the points we had removed, this group action is still well defined, since $c_1\mathbf{v_1} = c_2\mathbf{v_2}$ only if $\mathbf{v_1} = c_1^{-1}c_2\mathbf{v_2}$, which have already been removed. Furthermore, $(c_1\mathbf{v_1}, c_2\mathbf{v_2}, c_1^{-1}w_1, c_2^{-1}w_2) \neq 0$, since $c_1, c_2, \mathbf{v_1}, \mathbf{v_2}, w_1, w_2 \neq 0$.

By contradiction, we can show that this action is free. Suppose that it is not free. Then, there exists $(c_1, c_2) \neq (1, 1)$ such that $(c_1\mathbf{v_1}, c_2\mathbf{v_2}, c_1^{-1}w_1, c_2^{-1}w_2) = (\mathbf{v_1}, \mathbf{v_2}, w_1, w_2)$. Thus,

$$\begin{aligned} c_1 &= \mathbf{v_1}/\mathbf{v_1} = w_1/w_1 = 1, \\ c_2 &= \mathbf{v_2}/\mathbf{v_2} = w_2/w_2 = 1. \end{aligned}$$

The equation above is well defined since $\mathbf{v_1}, \mathbf{v_2}, w_1, w_2 \neq 0$. There is clearly a contradiction;hus the action must be free.

By Theorem 3.3.2, ρ_{nnh} is smooth. As with Example 3.1.2, the map $(g, p) \to (\rho_{\text{nnh}}(p), 0)$, for $g \in \mathcal{G}_{\text{nnh}}$ and $p \in \Theta_{\text{ReLU}}$ is proper; this argument follows similarly, since $\Theta_{\text{ReLU}} \times \Theta_{\text{ReLU}}$ is Hausdorff, and ρ_{nnh} has a continuous left inverse. Thus, the quotient space is a smooth manifold.

Symmetry group of node permutation

Secondly, we consider the permutation of nodes. The group we use here is the group $\mathcal{G}_{\text{per}} = \{0, 1\}$, where the neutral element is 0, and the group multiplication is prescribed by $1 \cdot 1 = 0$. Thus $1^{-1} = 1$. Its action on Θ_{ReLU} is,

$$\begin{aligned} \rho_{\mathcal{G}_{\text{per}}} : \mathcal{G}_{\text{per}} \times \Theta_{\text{ReLU}} &\to \Theta_{\text{ReLU}} \\ (g, (\mathbf{v_1}, \mathbf{v_2}, w_1, w_2)) &\mapsto \rho_{\mathcal{G}_{\text{per}}}(g, (\mathbf{v_1}, \mathbf{v_2}, w_1, w_2)) \\ &= \begin{cases} (\mathbf{v_1}, \mathbf{v_2}, w_1, w_2) & \text{if } g = 0 \\ (\mathbf{v_2}, \mathbf{v_1}, w_2, w_1) & \text{if } g = 1 \end{cases}. \end{aligned}$$

This group action is also well-defined, irrespective of the removed points. This is a discrete group action, and is therefore smooth. It is free, since for $(\mathbf{v_1}, \mathbf{v_2}, w_1, w_2)$ and a transformed output $(\mathbf{v_2}, \mathbf{v_1}, w_2, w_1)$, equality only holds if $\mathbf{v_1} = \mathbf{v_2}$. Since we had removed these elements, this is a contradiction. Properness holds, since

Finally, the symmetries caused by the reducibility only occur on the subset where $\mathbf{v_1}, \mathbf{v_2}, w_1$, or w_2 is equal to zero, or $\mathbf{v_1} = c\mathbf{v_2}$. Since these points were removed, the new Θ_{ReLU} contains models that are not reducible. Hence, we can ignore this symmetry.

Having looked at *individual* symmetry groups of Θ_{ReLU}, we now consider the product group $\mathcal{G}_{\text{ReLU}} = \mathcal{G}_{\text{nnd}} \times \mathcal{G}_{\text{per}}$. Here, the group action is given by,

$$\begin{aligned} \rho_{\text{ReLU}} = \mathcal{G}_{\text{ReLU}} \times \Theta_{\text{ReLU}} &\to \Theta_{\text{ReLU}} \\ (g_{\text{ReLU}} = (g_{\text{ndd}}, g_{\text{per}}), (\mathbf{v_1}, \mathbf{v_2}, w_1, w_2)) &\mapsto \rho_{\text{ReLU}}((g_{\text{ndd}}, g_{\text{per}}), (\mathbf{v_1}, \mathbf{v_2}, w_1, w_2)) \\ &= \rho_{g_{\text{nnd}}} \circ \rho_{g_{\text{per}}}((\mathbf{v_1}, \mathbf{v_2}, w_1, w_2)). \end{aligned}$$

Notation. $g_{\text{ReLU}} = (g_{\text{ndd}}, g_{\text{per}})$ denotes that an element of $\mathcal{G}_{\text{ReLU}}$ is to be denoted by g_{ReLU}, but also consists of $(g_{\text{ndd}}, g_{\text{per}})$.

Further, by $\rho_{g_{\text{per}}}$ and $\rho_{g_{\text{nnd}}}$, we denote the transformation specified by the group actions g_{ndd}, and g_{per} respectively.

Interestingly, this product action is still free. If the action from $\mathcal{G}_{\text{per}}$ is chosen from the group element 0, then, since $\mathcal{G}_{\text{nnd}}$ is free, the composed map must be free. On the other hand, if $\mathcal{G}_{\text{per}}$ is 1, we can show, by contradiction, that the composed action must still be free. If it isn't, then for some $(\mathbf{v_1}, \mathbf{v_2}, w_1, w_2)$ there must exist $((c_1, c_2) \neq (0, 0)) \in \mathcal{G}_{\text{nnd}}$ such that $(c_2\mathbf{v_2}, c_1\mathbf{v_1}, c_2^{-1}w_2, c_1^{-1}w_1) = (\mathbf{v_1}, \mathbf{v_2}, w_1, w_2)$. If so, we would have that $\mathbf{v_1} = c_2\mathbf{v_2}$, and $\mathbf{v_2} = c_1\mathbf{v_1}$; this is a contradiction, since such constant scales had been removed.

For any action defined by an element of $\mathcal{G}_{\text{nnd}} \times \mathcal{G}_{\text{per}}$, we have a smooth map, since each $\rho_{g_{\text{per}}}$ and $\rho_{g_{\text{nnd}}}$ is smooth. In order to check whether the complete group action is proper, we must check if a continuous inverse exists for $(g, p) \mapsto (\rho_{\text{ReLU}}(g, p), p)$, for $g \in \mathcal{G}_{\text{ReLU}}$, (as per Proposition A.53(d) in [58, Appendix A]).

Suppose we are given $p = (\mathbf{v_1}, \mathbf{v_2}, w_1, w_2)$ and $\rho_{\text{ReLU}}(g, p) = (\mathbf{v_1}', \mathbf{v_2}', w_1', w_2')$. Let us say that the correct component from $\mathcal{G}_{\text{per}}$ is 0. Then,

$$(\mathbf{v_1}', \mathbf{v_2}', w_1', w_2') = (c_1\mathbf{v_1}, c_2\mathbf{v_2}, c_1^{-1}w_1, c_2^{-1}w_2).$$

From this c_1 and c_2 can easily be found. Alternatively, if we assume that $\mathcal{G}_{\text{per}} = 1$, then

$$(\mathbf{v_1}', \mathbf{v_2}', w_1', w_2') = (c_2\mathbf{v_2}, c_1\mathbf{v_1}, c_2^{-1}w_2, c_1^{-1}w_1).$$

Therefore, to find c_1, we would carry out $\mathbf{v_2}'/\mathbf{v_1}$. However, in reality, $\mathbf{v_2}' = c_2\mathbf{v_2} \implies c_1 = c_2\mathbf{v_2}/\mathbf{v_1}$. This cannot be, since $\mathbf{v_1} = (c_2/c_1)\mathbf{v_2}$ would have been removed.

We can similarly identify if the original element of $\mathcal{G}_{\text{per}}$ was 1 by checking the converse of the above. Thus, a continuous inverse exists; therefore, the quotient manifold theorem holds.

Remark 3.3.2: When $d \geq 2$, the elements that were removed are indeed measure zero. However, when $d = 1$, this is no longer the case. •

3.4 Visualising symmetries using GENNI

The discussion above shows that studying the symmetries of a neural network in general can be tricky. One must account for the various symmetry groups that can cause such symmetries, but also how they interact in the full product group. In particular, large networks can have very non-trivial isotropy groups w.r.t. the group action.

In order to aide in the study of symmetries in neural networks, we had proposed a method for visualising the Geometry of Equivalences for Neural Network Identifiability (GENNI) in [59], which we summarise here. This technique provides a visualisation of nearby models of a particular model (as given by its parameter) that are in the same equivalence class (in terms of functional equivalence). For a given architecture $\kappa : \Theta \times X \to Y$, and a given model $\theta_0 \in \Theta$, we initially find $m+1$ points that are symmetric to θ_0 by minimising the following objective:

$$\begin{aligned} J_{\theta_0} &: \Theta \to \mathbb{R} \\ &\theta \mapsto J_{\theta_0}(\theta) = \frac{1}{|\overline{X}|} \sum_{x \in \overline{X}} |\kappa(\theta_0, x) - \kappa(\theta, x)|_2^2. \end{aligned} \tag{3.6}$$

This objective is an approximation of the metric on the parametric model space:

$$\begin{aligned} d &: \Theta_\kappa \times \Theta_\kappa \to \mathbb{R}^+ \backslash \{0\} \\ &(\theta_1, \theta_2) \mapsto d(\theta, \theta) = \sqrt{\int_X |\kappa(\theta_0, x) - \kappa(\theta, x)|_2^2 \, \mathrm{d}x}. \end{aligned}$$

Since J_{θ_0} is minimsed whenever the architecture maps are equivalent, we note that $[\theta_0] := \{\theta^* \mid \theta^* = \arg\min_{\theta \in \Theta} J_{\theta_0}(\theta)\} = \mathrm{preim}_{\pi_\kappa}(\theta_0)$, where $\pi_\kappa : \Theta \to \Theta_\kappa$ is the projection map w.r.t. to the equivalence relation $\sim_\kappa$. Thus, by minimising Equation (3.6), we find equivalent models.

Following this, we find a linear subspace that contains these parameter values using the Gram-Schmidt algorithm. We use this subspace to sample more symmetric parameter values on a predetermined grid, and keep them if, for some $\epsilon > 0$, $J_{\theta_0}(\theta) < \epsilon$. This set is called the $\epsilon-$equivalent set. This is then visualised graphically; if $m \leq 3$, this can be visualised directly. If not, we carry out dimensionality reduction, such as UMAP [65].

This page was left intentionally blank.

Chapter 4

Learning in Machines

Tasks and models can be connected using loss functions and learning algorithms. Together, a parametric model, loss function and learning algorithm form a local representation of the task.

4.1 Learning Problems

A task is a system, along with a particular data-generating process. This doesn't intrinsically specify what we want to learn. Since the data-generating process contains compositions of structure maps, what we want to learn is related to these structure maps. The key notion is that one of these structure maps is unknown, as we want to find a suitable proxy that we can use. Together, a task, its unknown structure, and the measure of suitability gives us a learning problem.

4.1.1 Subject of learning

Given a task, a learning problem can be created if there is a single consecutive aspect of a structure map of the task that is unknown to us. Recall that a task is a derived system. Further, in Definition 2.6.2, we said that for each system of functions that derives a system, there is a single, unique structure map that it affects. A particular structure map, however, could be affected by several systems of functions.

Denote by $\mathbb{F} = \{(\mathfrak{R}_i, \mathfrak{S}_i)\}_{i\in\mathcal{I}}$ the set of systems of functions that derived a particular task $\mathfrak{t} = ((\mathfrak{R}, \mathfrak{S}), \mathfrak{P})$. Suppose that a subset $(U_\mathbb{F} \subseteq \mathbb{F}) = \{(\mathfrak{R}_j, \mathfrak{S}_j)\}_{j\in\mathcal{I}_U}$ is such that all $(\mathfrak{R}_j, \mathfrak{S}_j)$ affects a single structure map $f \in \mathfrak{S}$, and,

$$f = f^{\text{pre}} \circ \bigcirc_{j\in\mathcal{I}_U} f_j \circ f^{\text{suf}}. \tag{4.1}$$

f_j should be interpreted as in Definition 2.6.2.

Notation. $\bigcirc_{j\in\mathcal{I}_U} f_j$ denotes $... \circ f_j \circ ...$ for $j \in \mathcal{I}_U$.

We say that $U_\mathbb{F}$ affects f *consecutively*. Then, f is a valid unknown structure map, that can be learned. It is implicitly assumed that this structure map is

used in the data-generating process of the task. The data-generating process is a composition of structure maps; we further assume that f is the only unknown structure map, and that f^{pre} and f^{suf} correspond to known components of the structure map, if any.

We call f the *subject of learning*; it is the unknown structure map.

Let us look at examples of unknown structure maps in some ML scenarios:

Example 4.1.1 Supervised Learning: Consider the supervised learning problem from Example 2.5.1:

$$\mathfrak{t}_{\text{SL}} = ((\mathfrak{R}_{\text{SL}}, \mathfrak{S}_{\text{SL}}), \mathfrak{P}_{\text{SL}}),$$

where

$$\begin{aligned}
&\mathfrak{R}_{\text{SL}} = \{X, Y\},\\
&\mathfrak{S}_{\text{SL}} = \{f : X \to Y\},\\
&\mathfrak{P}_{\text{SL}} : X \times X \to X \times Y\\
&\qquad (x, x) \mapsto \mathfrak{P}_{\text{SL}}(x, x) = (\text{id}_X(x), f(x)).
\end{aligned}$$

For such a task, the subject of learning would be f. ▲

Example 4.1.2 Model-free Reinforcement Learning: A minimal RL task can be defined, as in Section 2.5.2, by,

$$\mathfrak{t}_{\text{RL}} = ((\mathfrak{R}_{\text{RL}}, \mathfrak{S}_{\text{RL}}), \mathfrak{P}_{\text{RL}}),$$

where,

$$\begin{aligned}
&\mathfrak{R}_{\text{RL}} = \{\mathcal{S}, \mathcal{A}, \mathbb{R}, \{\gamma\}, \Pi\},\\
&\mathfrak{S}_{\text{RL}} = \{\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S},\\
&\qquad\qquad \mathcal{R} : \mathcal{S} \to \mathbb{R}\},\\
&\mathfrak{P}_{\text{RL}} : \mathcal{S} \times \Pi \to \mathcal{O}^{\infty}_{\text{RL}}\\
&\qquad (s_0, \pi) \mapsto \mathfrak{P}_{\text{RL}}(s_0, \pi) = f^{\pi}_{\text{roll}}(s_0).
\end{aligned}$$

See Section 2.5.2 for definitions of the notation.

This task is defined such that we can input an arbitrary policy from $\pi \in \Pi$ and obtain trajectories. Typically, the goal of a RL problem is to find the optimal policy, where optimality is defined by,

$$\pi^* = \arg\max_{\pi \in \Pi} V_\pi(s_0),$$

where $V_\pi(s_0)$ is the value of π, for some initial state $s_0 \in \mathcal{S}$. Thus, the subject of learning is the structure map π^* from another, associated system,

$$\mathfrak{z}_{\pi^*} = (\mathfrak{R}_{\pi^*}, \mathfrak{S}_{\pi^*}),$$

where,

$$\begin{aligned}
&\mathfrak{R}_{\pi^*} = \{\mathcal{S}, \mathcal{A}\},\\
&\mathfrak{S}_{\pi^*} = \{\pi^* : \mathcal{S} \to \mathcal{A}\}.
\end{aligned}$$

▲

Example 4.1.3 Model-based Reinforcement Learning: The main difference here, compared to model-free RL in Example 4.1.2 is that both the optimal policy *and the transition dynamics* are subjects of learning. ▲

We can now define a task as being learnable when an unknown structure map has been identified.

Definition of a Learning task

Definition 4.1.1 (Learning task)**.** *Suppose we are given a task* $\mathfrak{t} = ((\mathfrak{R}, \mathfrak{S}), \mathfrak{P})$*, which was derived from* $\mathbb{F} = \{(\mathfrak{R}_i, \mathfrak{S}_i)\}_{i \in \mathcal{I}}$*.*

Further, we are given a subset $(U \subseteq \mathbb{F}) = \{(\mathfrak{R}_j, \mathfrak{S}_j)\}_{j \in \mathcal{I}_U}$ *which consists of systems of functions that consecutively affect a single structure map* $f \in \mathfrak{S}$ *of* $\mathfrak{t}$*, as in Equation* (4.1)*. We call* f *the chosen subject of learning.*

Together, $\mathfrak{t}_f = (\mathfrak{t}, f)$ *form a learning task.*

An important note is that the subject of learning is chosen; different learning tasks can be derived from a single task, depending on which suitable unknown structure map is chosen as the subject of learning.

Although tasks can exist in a vacuum, we want to think about tasks that exist in task spaces. A task space is derived by considering all variations of the derived system, w.r.t. all systems of functions that were used. Then, given $U \subseteq \mathfrak{S}$, we can derive a learning task space by projecting the task space onto the coordinates that correspond to the systems of functions that affect the structure maps in U.

Definition of a Space of Learning Tasks

Definition 4.1.2 (Space of Learning Tasks)**.** *Given a space of tasks* $\mathfrak{T}$*, let us denote by* $\mathbb{F} = \{(\mathfrak{R}_i, \mathfrak{S}_i)\}_{i \in \mathcal{I}}$ *the systems of functions that were used to derive* $\mathfrak{T}$*. Specify a subset* $(U_{\mathbb{F}} \subseteq \mathbb{F}) = \{(\mathfrak{R}_j, \mathfrak{S}_j)\}_{j \in \mathcal{I}_U}$ *such that, for each* $(\mathfrak{t} \in \mathfrak{T}) = ((\mathfrak{R}, \mathfrak{S}), \mathfrak{P})$*, there exists a* $f \in \mathfrak{S}$ *which is affected consecutively by* $U_{\mathbb{F}}$*, and* $(\mathfrak{t}, f)$ *is a learning task.*

The space of learning tasks is the submanifold of $\mathfrak{T}$ *which is derived as the total variation space (see Definition 2.6.4) of systems derived from* $U_{\mathbb{F}}$*.*

Notation. We will be using $\mathfrak{T}$ to denote either a task space, or a learning task space; if their differences are important, we will specify as needed.

As an example of where the task space and the space of learning tasks differ, consider Example 4.1.3. A potential space of RL tasks could be obtained by varying both the transition dynamics, as well as the reward function. However, for the purpose of learning the transition dynamics, any variations in the reward function aren't useful, and thus can be projected out.

The space of learning tasks can be interpreted with an equivalence relation, where we say that we aren't interested in any variation of tasks that are derived from $U_{\mathbb{F}^{\mathsf{c}}}$. This does not mean that the outcomes of the data-generating process is independent of $U_{\mathbb{F}^{\mathsf{c}}}$. However, as we will see in the next section, this does

mean that such variations do not affect how we evaluate a proxy for the subject of learning.

The definition of a learning task is made as in Definition 4.1.1 so that it is clear that the *only* differences between any task $\mathfrak{t} \in \mathfrak{T}$ is characterised by the variations in $\mathfrak{T}$. Thus, any other salient property is the same across all learning tasks. For example, if the subject of learning is continuous in one learning task, it is continuous in any other learning task from the same space of learning tasks. This becomes important since this way, by assumption, any learning algorithm that can be applied to one learning task, can be applied to any other learning task.

We saw in Example 4.1.3, that it is possible to consider more than one subject of learning. The point however is that a single learning problem is associated to a single subject of learning. The solutions of multiple learning problems can be interlinked, as we will see in Section 8.1; this does not deviate from this condition.

4.1.2 Loss functions

Having identified the structure maps of a task that we would like to learn, it remains to specify how to determine the suitability of proxies of the subjects of learning. This is carried out by a loss function.

A loss function can be generally thought of as a map on a space of learning tasks and a *space of proxy maps*. A proxy map is a possible replacement for the unknown structure map. A proxy must satisfy any other structure maps that are composed of the unknown structure map. For example, if the structure map is continuous, then the proxy must also be continuous. The space of proxy maps is the space of all functions that can act as a proxy map.

Definition of a Space of Proxy maps

Definition 4.1.3 (Space of Proxy maps)**.** *Given a learning task* $(\mathfrak{t}, f)$*, where* f *has domain and codomain* X *and* Y *respectively, and* $\mathfrak{t} = ((\mathfrak{R}, \mathfrak{S}), \mathfrak{P})$*, the space of proxy maps* $\mathcal{F}^{\text{px}}$ *is the* largest *space of functions, where every* $f^{\text{px}} \in \mathcal{F}^{\text{px}}$ *satisfies:*

a) $f^{\text{px}} : X \to Y$*,*

b) f^{px} *satisfies all structure maps that* f *satisfied in the systems of functions from which it was derived.*

Furthermore, given a space of learning tasks $\mathfrak{T}$*, which has a topology* $\mathcal{O}_{\mathfrak{T}}$*,* $\mathcal{F}^{\text{px}}$ *has a topology* $\mathcal{O}_{\mathcal{F}^{\text{px}}}$ *such that,*

$$\mathcal{O}_{\mathfrak{T}} \subseteq \mathcal{O}_{\mathcal{F}^{\text{px}}}|_{\mathfrak{T}},$$

where $\mathcal{O}_{\mathcal{F}^{\text{px}}}|_{\mathfrak{T}}$ *denotes the subset topology on* $\mathfrak{T}$ *w.r.t.* $\mathcal{O}_{\mathcal{F}^{\text{px}}}$*.*

Since the differences between the structure maps of tasks in $\mathfrak{T}$ are described by the variations of the task space, the space of proxy maps is the same for

all learning tasks in $\mathfrak{T}$. Thus, there is a unique space of proxy maps for each space of learning tasks.

In addition to satisfying any other conditions that the structure of the learning tasks set out, a space of proxy maps also has a topology that can be used to derive the topology of the space of learning tasks. Common topologies on function spaces apply here; for example, if $\mathcal{F}^{\text{px}}$ is a space of continuous functions, then the compact-open topology can be used.

Definition 4.1.4 (Loss Function)**.** *Given a learning task space $\mathfrak{T}$, a loss function $\mathcal{L}$ is a map,* Definition of a Loss Function

$$\mathcal{L} : \mathcal{F}^{\text{px}} \times \mathfrak{T} \to \mathbb{R},$$

where $\mathcal{F}^{\text{px}}$ is the space of proxy maps for $\mathfrak{T}$.

Smaller values of loss functions imply that a particular proxy map is more suitable for a given task. Thus, in a local region, the most suitable proxy map would be the minimising proxy map, for a given learning task. It should be noted that we have not made any assumptions about the loss function, be it smoothness, continuity, or convexity. Furthermore, as we will see in the examples below, the loss function will typically use of data-generating process of the task. A task can vary relative to structure maps that are not used when defining the learning task, and the data-generating process can vary with these too. However, the loss function is a map on the space of learning tasks, and by assumption, is not affected by any variations of the task outside of those in the space of learning tasks.

Example 4.1.4 Supervised Learning: Suppose we are given the learning task from Example 4.1.1, which comes from a space of learning tasks $\mathfrak{T}_{\text{SL}}$. For this, let us assume the Y is $\mathbb{R}^m$ We assumed that the subject of learning is a smooth, continuous map. A commonly used loss function is the squared loss:

$$\begin{aligned}\mathcal{L}_{\text{SL}} &: \mathcal{F}^{\text{px}} \times \mathfrak{T}_{\text{SL}} \to \mathbb{R} \\ &(f^{\text{px}}, \mathfrak{t}_{\text{SL}}) \mapsto \mathcal{L}(f^{\text{px}}, \mathfrak{t}_{\text{SL}}) = \int_X ||f^{\text{px}}(x) - \text{pr}_2 \circ \mathfrak{P}_{\text{SL}}(x)||_2^2 \, \text{d}x.\end{aligned}$$

Other possible loss functions include any p-th power L^p loss, and the cross-entropy loss (for classification problems). If Y is not Euclidean, then the loss can be defined with an appropriate metric or distance function. ▲

Example 4.1.5 Reinforcement Learning: Suppose we are given the learning task from Example 4.1.2, which comes from a space of learning tasks $\mathfrak{T}_{\text{RL}}$. Recall that the subject of learning of a RL problem is another system $\mathfrak{z}_{\pi^*}$. The space of proxy maps is the space of policies Π. Then, the loss function is a reciprocal of the value function of the RL problem[1]. That is,

$$\begin{aligned}\mathcal{L}^{s_0}_{\text{RL}} &: \Pi \times \mathfrak{T}_{\text{RL}} \to \mathbb{R} \\ &(\pi, \mathfrak{t}_{\text{RL}}) \mapsto \mathcal{L}^{s_0}_{\text{RL}}(\pi, \mathfrak{t}_{\text{RL}}) = V_\pi(s_0).\end{aligned}$$

[1]The reciprocal here is because the reward function increases as we get closer to the optimal policy, while a loss function is assumed to decrease.

As we saw in Definition 2.5.3, the value function is defined over a trajectory, which can be generated from the data-generating process of the RL task, as defined in Section 2.5.2. ▲

Together, a learning task and a loss function gives us a learning problem.

Definition of a (Space of) Learning problem(s)

Definition 4.1.5 ((Space of) Learning problem(s))**.** *A learning task* $\mathfrak{t}$ *contained in a space of learning tasks* $\mathfrak{T}$*, and a loss function* $\mathcal{L} : \mathcal{F}^{\text{px}} \times \mathfrak{T} \to \mathbb{R}$ *is a learning problem.*

The tuple $(\mathfrak{T}, \mathcal{L})$ *is a space of learning problems.*

Thus, a learning problem contains information of *what* needs to be learned, and *how* to determine when we have learned it (by measuring a potential solution against the loss function). We included the space of learning tasks, from which the particular task was taken, as a part of the definition of a learning problem. Typically, when considering just learning a single task, this might not be necessary. However, some information regarding its topology could be useful when devising the model architecture; this is left for future work. The importance of the space of learning tasks is in its use in transfer.

4.2 Learning Algorithms

In practice, we will typically not work with the full space of proxy maps; we will be making a choice of a model architecture, and thus a model space. Since we assume that we have knowledge of all structure maps that are not unknown, the parametric model space is a subset of the space of proxy maps. Thus, the loss function that is used is a restriction of the loss function defined in Definition 4.1.4. Conceptually, the full loss function is independent of any choice of the model architecture we make, and as per Definition 4.1.5, is a component of the learning problem. However, for practical purposes, it suffices to ensure that it works on $\Theta_\kappa \times \mathfrak{T}$.

The goal of a learning problem $(\mathfrak{t}, \mathcal{L})$ is typically to find a $f^{\text{px}*} \in \mathcal{F}^{\text{px}}$ such that,

$$f^{\text{px}*} = \underset{f^{\text{px}} \in \mathcal{F}^{\text{px}}}{\arg\min}\, \mathcal{L}(f^{\text{px}}, \mathfrak{t}).$$

This would be carried out by a learning algorithm:

Definition of a Learning Algorithm

Definition 4.2.1 (Learning Algorithm)**.** *Given a space of learning problems* $(\mathfrak{T}, \mathcal{L})$*, and a model space* Θ_κ*, a learning algorithm is map,*

$$\mathfrak{L} : \mathfrak{T} \to \Theta_\kappa,$$

where

$$\mathfrak{L}(\mathfrak{t}) \in \underset{\mathfrak{m} \in \Theta_\kappa}{\arg\min}\, \mathcal{L}_{\mathfrak{t}}(\mathfrak{m}).$$

A learning algorithm is therefore a map that takes a task to a model in the model space, ideally to the model which is best suited to replace the subject of learning.

4.2.1 Gradient Descent

Summary of Gradient Descent

The gradient descent method is a popular learning algorithm. For this, we must assume that loss function, for each task, is smooth w.r.t. the model space. That is, the map $\mathcal{L}_{\mathfrak{t}} : \Theta_\kappa \to \mathbb{R}$ is smooth. The gradient descent algorithm is typically a discrete technique, where we make an update to the current estimate of the best model using information of the gradient of $\mathcal{L}_{\mathfrak{t}}$ at that point. In the present work however, we will present an idealised version of gradient descent, where the algorithm is said to follow the flow of the gradient vector field on Θ_κ, derived from the loss function.

More precisely:

Definition of a Gradient Descent

Definition 4.2.2 (Gradient Descent)**.** *Given a learning problem $(\mathfrak{t}, \mathcal{L}) \in (\mathfrak{T}, \mathcal{L})$, and a model space Θ_κ such that $\mathcal{L}_{\mathfrak{t}} : \Theta_\kappa \to \mathbb{R}$ is a smooth function, we obtain a co-vector field on Θ_κ by taking the differential of the negative loss $-\mathcal{L}_{\mathfrak{t}}$, denoted by $d\mathcal{L}_{\mathfrak{t}}$.*

Let us denote $v_{d\mathcal{L}_{\mathfrak{t}}} : \Theta_\kappa \times \mathbb{R} \to \Theta_\kappa$ as the flow of $d\mathcal{L}_{\mathfrak{t}}$. Then, given an initial model $\mathfrak{m}_0 \in \Theta_\kappa$, the gradient descent learning algorithm is the map

$$\begin{aligned} \mathfrak{L}_{\mathrm{GD}} : \mathfrak{T} &\to \Theta_\kappa \\ \mathfrak{t} &\mapsto \mathfrak{L}_{\mathrm{GD}}(\mathfrak{t}) = \lim_{t\to\infty} v_{d\mathcal{L}_{\mathfrak{t}}}(\mathfrak{m}_0, t). \end{aligned}$$

Behaviour of Gradient Descent

Thus, given an initial starting point, the gradient descent algorithm returns to the attracting singular point that this point flows to. From a dynamical systems point of view, some key assumptions have been made about the loss function. Firstly, it is assumed that the loss function contains at least one singular point[2] where its linearised system contains at least one eigenvalue with a negative real value.

By the Stable (Center[3]) Manifold Theorem [64], we know that there exists a smooth stable manifold around this singular point. Then, we assume that the chosen initial point is in the stable manifold of one such a singular point. This means that the appropriate map as in Definition 4.2.2 can be defined.

Differences in practice

As mentioned before, we discretise the gradient descent algorithm in practice. This is equivalent to using Euler's Method [111] to numerically solve the differential equations of the flow, with the initial value being set to $\mathfrak{m}_0$. We also do not let $t \to \infty$. Finally, typical ML uses stochastic gradients, where approximations of the gradient are used.

[2] In dynamical systems literature [100], this is called a fixed point, since it refers to fixed points of the differential equations defining the dynamical system. We use the terminology used by [58].

[3] If there are eigenvalues with zero real value

This page was left intentionally blank.

Chapter 5

Representations and Learning

In this chapter, we argue that the previous chapters can be unified under single framework of *representations*.

Summary of representations

Our notion of a representation was inspired by Marr [63]. In [63], a representation is a formal *scheme* by which *something* can be described; the result of the application of such a scheme to a particular object is its *description*. The representation is an agreed-upon convention that can be used to describe things in a consistent manner.

Examples of representations

Pedagogical examples of this are the languages that we communicate with. Each word in a language capture the notion of an elementary idea, and sentences allow us to describe more complex ideas, built from such elementary units. The notions that words describe are universally agreed by other users of a language, which allows us to communicate what we think, feel and want to express. A language exists for this reason, a reason against which we can measure its usefulness (at least qualitatively). The representation is the process by which we translate ideas into words and symbols.

Similarly, we pose the learning algorithm as a representation of a space of tasks. The symbols that describe tasks are contained in the chosen model space. In the rest of this chapter, we formalise these notions, and describe how the previous statement can be arrived at. We end this chapter by briefly discussing the similarities of our theory to that of category theory.

5.1 Representations

In order to formalise the intuitions we have about representations, we begin by defining a representation map.

Definition of a Representation map

Definition 5.1.1 (Representation map)**.** *Given a space of symbols* $\mathfrak{Y}$*, and a space* X*, a representation map* $\mathfrak{K}_X^{\mathfrak{Y}}$ *on* X *is a* surjective *map,*

$$\mathfrak{K}_X^{\mathfrak{Y}} : X \to \mathfrak{Y},$$

Notation. Given a representation map $\mathfrak{K}_X(X)$ of X, where the symbol space is not specified, we will denote by $\mathfrak{K}_X(X)$ any valid symbol space for this representation map.

Furthermore, the notation $\mathfrak{K}(\cdot)$, where $\cdot$ is a *space* denotes the output space of the representation map; if it is an element, then the usual map applies. In the future, we will also suppress the notation $\mathfrak{K}_X^{\mathfrak{Y}}$ to $\mathfrak{K}$ for brevity; it is expected that X and $\mathfrak{Y}$ will be clear from the context, unless otherwise specified.

In addition to being surjective, a representation map could be injective too. In this case, the representation would be *exact*; if not, it is *approximate*.

Given a representation map, a *description* of an element $x \in X$ is,

Definition of a Description

Definition 5.1.2 (Description)**.** *Given a representation map $\mathfrak{K}_X^{\mathfrak{Y}}$ on X using symbols $\mathfrak{Y}$, a description of $x \in X$ is $\mathfrak{y} \in \mathfrak{Y}$ such that,*

$$\mathfrak{y} = \mathfrak{K}_X^{\mathfrak{Y}}(x).$$

Thus, a description of an element of a set depends on the chosen representation map. In Section 5.1.5, we will discuss how there can be many equivalent representations.

The final condition of Definition 5.1.1, which states that the preimages of any represented element are disjoint implies that the representation map creates a partition of the represented set. That is, $\sim_{\mathfrak{K}_X^{\mathfrak{Y}}}$ is an equivalence relation, where $x \sim_{\mathfrak{K}_X^{\mathfrak{Y}}} y \iff \mathfrak{K}_X^{\mathfrak{Y}}(x) = \mathfrak{K}_X^{\mathfrak{Y}}(y)$ for $x, y \in X$. This ensures that there is no ambiguity about how we can *interpret* a given symbol, up to the equivalence relation given by the partition. An *interpretation* of a description is then defined as,

Definition of a Interpretation of a description

Definition 5.1.3 (Interpretation of a description)**.** *Suppose $\mathfrak{y}$ is a description of $x \in X$ under a representation map $\mathfrak{K}_X^{\mathfrak{Y}}$. Furthermore, denote $\sim_{\mathfrak{K}_X^{\mathfrak{Y}}}$ as the equivalence relation induced by $\mathfrak{K}_X^{\mathfrak{Y}}$. Then, an interpretation $\mathfrak{I}$ of the description $\mathfrak{y}$ is a bijective map,*

$$\mathfrak{I} : \mathfrak{Y} \to X/\sim_{\mathfrak{K}_X^{\mathfrak{Y}}},$$

such that for $\mathfrak{y} \in \mathfrak{Y}$,

$$\mathfrak{I}(\mathfrak{y}) \in \mathit{preim}_{\mathfrak{K}_X^{\mathfrak{Y}}}(\mathfrak{y}).$$

$\mathfrak{I}$ is also called an interpretation map.

From this definition, the map $\mathfrak{I} \circ \mathfrak{K}_X^{\mathfrak{Y}}$ is the projection map of the equivalence relation $\sim_{\mathfrak{K}_X^{\mathfrak{Y}}}$.

5.1.1 Representation of a system

In the present work, we will be using representation maps to represent systems. Since a system contains some structure, we can then place additional constraints on the interpretation.

Definition of a Representation of a system

Definition 5.1.4 (Representation of a system)**.** *Suppose we are given two systems* $(\mathfrak{R}, \mathfrak{S})$ *and* $(\mathfrak{R}_\mathfrak{y}, \mathfrak{S}_\mathfrak{y})$*. We will call* $(\mathfrak{R}_\mathfrak{y}, \mathfrak{S}_\mathfrak{y})$ *the system of symbols.*

$(\mathfrak{R}_\mathfrak{y}, \mathfrak{S}_\mathfrak{y})$ *has the property that for each relatum* $X \in \mathfrak{R}$*, there is a corresponding symbol space* $\mathfrak{Y}_X \in \mathfrak{R}_\mathfrak{y}$*. Furthermore, for each* $f \in \mathfrak{S}$*, there is a corresponding* $f_\mathfrak{y} \in \mathfrak{S}_\mathfrak{y}$*. Thus, there is a correspondence in the sets of domains and codomains too.*

A collection of representation maps and interpretations $\{(\mathfrak{K}_X^{\mathfrak{Y}_X}, \mathfrak{I}_X)\}_{X \in \mathfrak{R}}$ *will be called a representation of the system* $\mathfrak{z}$ *if,*

a) for each domain or codomain $D_\mathfrak{y}^X$ *that is a set of subsets*[1] *of* $\mathfrak{Y}_X$*,*

$$preim_{\mathfrak{K}_X}(x) \in D^X, \text{ for } x \in D_\mathfrak{y}.$$

b) for any structure map $(f \in \mathfrak{S}) : D^{X_1} \times ... \to D^{Y_1} \times ...$ *and the corresponding* $(f_\mathfrak{y} \in \mathfrak{S}_\mathfrak{y}) : D_\mathfrak{y}^{X_1} \times ... \to D_\mathfrak{y}^{Y_1} \times ...$*,*

$$preim_{\mathfrak{K}_{X_1}} \circ f_\mathfrak{y}(\mathfrak{y}_{X_1}, ...), ... = f(preim_{\mathfrak{K}_{X_1}}(x_1), ...).$$

A representation of a system ensures that the descriptions of any element of a space in the relata are *consistent* w.r.t. its structure. The symbol space preserves the structure of the original system by ensuring that where possible, carrying out an operation (by applying a structure map) in the symbol space has a corresponding operation in the original system.

Corollary 5.1.1. *Definition 5.1.4 immediately implies a structure on the quotient space* $X/\sim_{\mathfrak{K}_X^\mathfrak{Y}}$*, where* $X \in \mathfrak{R}$ *of the system being represented.*

More precisely, if $(f \in \mathfrak{S}) : D_{X_1} \times ... \to DY_1 \times ...$ *is a structure map, then there is an induced map,*

$$\begin{aligned} f|_{\sim\mathfrak{K}} : D_{X_1/\sim_{\mathfrak{K}_{X_1}}} \times ... &\to DY_1/\sim_{\mathfrak{K}_{Y_1}} \times ... \\ (x_1, ...) \mapsto f|_{\sim\mathfrak{K}}(x_1, ...) &= \mathfrak{I}_{Y_1} \circ f_\mathfrak{y}(\mathfrak{I}_{X_1}^{-1}(x_1), ...), ... \\ &= \mathfrak{K}_{Y_1} \circ f(preim_{\mathfrak{K}_{X_1}} \circ \mathfrak{I}_{X_1}^{-1}(x_1), ...), ... \end{aligned}$$

Proof. Follows by the bijectivity of the interpretation map. ■

[1] Recall that the set of domains and codomains consists of subsets or sets of subsets of relata. In the former case, the required condition is $\text{preim}_{\mathfrak{K}_X}(x) \subseteq D^X$, which is trivially satisfied by the surjectivity of the representation map. Thus, we have not stated this in Definition 5.1.4.

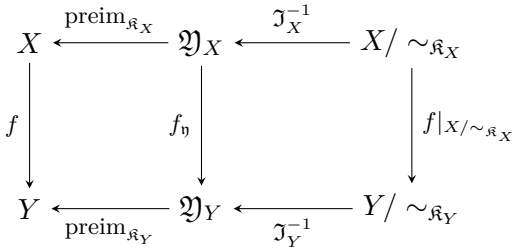


Figure 5.1. An example of the commutative diagram that must be satisfied by a representation of a system. Here, the structure map is $f : X \to Y$.

Definition 5.1.4 and Corollary 5.1.1 imply a commutative diagram for each structure map in $\mathfrak{S}$. An example of such a diagram is shown in Figure 5.1.1.
The preservation of structure that a representation satisfies complies with what is typically thought of as structure preservation in mathematics.

Example 5.1.1 Linear maps can represent linear systems: Consider the vector systems,

$$\mathfrak{z}_{V_1} = (\{V_1, \mathbb{R}\}, \{f_+^1 : V_1 \times V_1 \to V_1, f_\cdot^1 : \mathbb{R} \times V_1 \to V_1)$$
$$\mathfrak{z}_{V_2} = (\{V_2, \mathbb{R}\}, \{f_+^2 : V_2 \times V_2 \to V_2, f_\cdot^2 : \mathbb{R} \times V_2 \to V_2),$$

as in Example 2.1.3.

Theorem 5.1.2. *For a map $\mathfrak{K} : V_1 \to V_2$ with a valid interpretation to be a representation of $\mathfrak{z}_{V_1}$ with symbol space $\mathfrak{z}_{V_2}$, it is necessary and sufficient that $\mathfrak{K}$ is linear. We assume that the representation and interpretation of $\mathbb{R}$ are the identity maps.*

Proof. Since the domains of the structure maps are subsets of the relata, we do not need to consider Condition (a) of Definition 5.1.4. We will only show the proof for f_+ for brevity, since a similar procedure can be done for $f_\cdot$.

Notation. In this proof, we will denote a vector $v_j^i \in V_i$, where the superscript i refers to the vector space V_i that the vector is an element of, and the subscript j indexes the vectors. Thus v_1^1 and v_2^1 are 2 vectors from the same vector space V_1.

Necessity: Suppose Condition (b) is true. Then for $f_+^{1,2}$, we start with the following true statement,

$$\text{preim}_{\mathfrak{K}} \circ f_+^2(v_1^2, v_2^2) = f_+^1(\text{preim}_{\mathfrak{K}}(v_1^2), \text{preim}_{\mathfrak{K}}(v_2^2)).$$

Let us also denote for $v_1^1, v_2^1 \in V_1$,

$$\begin{aligned} \mathfrak{K}(v_1^1) &= v_1^2, \\ \mathfrak{K}(v_2^1) &= v_2^2, \end{aligned} \tag{5.1}$$

and,

$$f_+^1(v_1^1, v_2^1) = v. \tag{5.2}$$

Assuming that the condition for linearity is false,

$$\mathfrak{K} \circ f_+^1(v_1^1, v_2^1) \neq f_+^2(\mathfrak{K}(v_1^1), \mathfrak{K}(v_2^1)).$$

Then,

$$\begin{aligned} \mathfrak{K} \circ \text{preim}_{\mathfrak{K}} \circ \mathfrak{K}(v) &\neq \mathfrak{K} \circ \text{preim}_{\mathfrak{K}} \circ f_+^2(\mathfrak{K}(v_1^1), \mathfrak{K}(v_2^1)) \\ \mathfrak{K}([v]) &\neq \mathfrak{K} \circ \text{preim}_{\mathfrak{K}} \circ f_+^2(v_1^2, v_2^2). \end{aligned}$$

This implies[2] that,

$$\begin{aligned} [v] &\neq \text{preim}_{\mathfrak{K}} \circ f_+^2(v_1^2, v_2^2) \\ &\neq f_+^1([v_1^1], [v_2^1]). \end{aligned}$$

This contradicts Equation 5.2. Thus the assumption that linearity is false is contradicted.

Sufficiency: Linearity of $\mathfrak{K}$ implies that,

$$\mathfrak{K} \circ f_+^1(v_1^1, v_2^1) = f_+^2(\mathfrak{K}(v_1^1), \mathfrak{K}(v_2^1)).$$

Then, if we also follow the convention of Equation 5.1,

$$\begin{aligned} \text{preim}_{\mathfrak{K}} \circ \mathfrak{K} \circ f_+^1(v_1^1, v_2^1) &= \text{preim}_{\mathfrak{K}} \circ f_+^2(\mathfrak{K}(v_1^1), \mathfrak{K}(v_2^1)) \\ [f_+^1(v_1^1, v_2^1)] &= \text{preim}_{\mathfrak{K}} f_+^2(v_1^2, v_2^2) \\ f_+^1([v_1^1], [v_2^1]) &= \text{preim}_{\mathfrak{K}} f_+^2(v_1^2, v_2^2) \\ f_+^1(\text{preim}_{\mathfrak{K}}(v_1^2), \text{preim}_{\mathfrak{K}}(v_2^2)) &= \text{preim}_{\mathfrak{K}} f_+^2(v_1^2, v_2^2), \end{aligned}$$

as needed by Definition 5.1.4. Note that due to the surjectivity of the representation, the above works for any $v_1^2, v_2^2 \in V_2$. ▲

Linear maps typically preserve the algebra of a vector space; we see that the structure preservation of a representation of a vector system is exactly that of a linear map. ▲

Example 5.1.2 Continuous maps represent topological systems: For topological systems, Condition (a) of Definition 5.1.4 is exactly the definition for the representation to be continuous map. In fact, this makes the representation map a quotient map, and the topology on the symbol space the quotient topology. ▲

5.1.2 Equivalence of representations

Given 2 representations of a system, it can by useful to determine if they are equivalent to each other. Formally, this is defined as,

[2] Since $\mathfrak{K}$ is a valid map (i.e. it is not one-to-many), $\mathfrak{K}(x) \neq \mathfrak{K}(y) \implies x \neq y$.

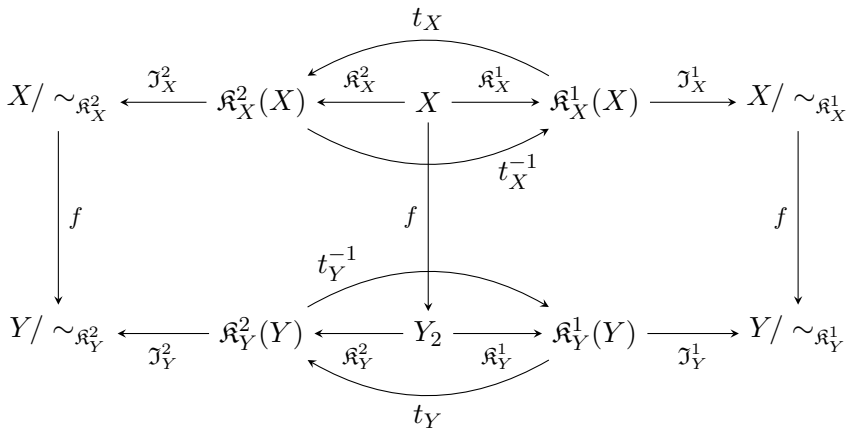


Figure 5.2. An example of a commutative diagram that must be satisfied by equivalent representations of a system. Here, the structure map is $f : X \to Y$.

Definition of a Equivalence of representations

Definition 5.1.5 (Equivalence of representations)**.** *Given a system* $(\mathfrak{R}, \mathfrak{S})$*, suppose that* $\{(\mathfrak{K}^1_U, \mathfrak{I}^1_U)\}_{U\in\mathfrak{R}}$ *and* $\{(\mathfrak{K}^2_U, \mathfrak{I}^2_U)\}_{U\in\mathfrak{R}}$ *are representations of it. We say that these representations are equivalent if the map,*

$$\begin{aligned} t_U : \mathfrak{K}_1(U) &\to \mathfrak{K}_2(U) \\ u &\mapsto t_U(u) = \mathfrak{K}_2 \circ \mathfrak{I}_1(u), \end{aligned}$$

is an isomorphism, and that for any structure map $(f \in \mathfrak{S}) : X_1 \times ... \to Y_1 \times ...$*,*

$$f(\mathfrak{I}^2_{X_1} \circ t_{X_1} \circ \mathfrak{K}^1_{X_1}(x_1), ...) = (\mathfrak{I}^2_{Y_1} \circ t_{Y_1} \circ \mathfrak{K}^1_{X_1} \circ f^1(x_1, ...), ...),$$

and,

$$f(\mathfrak{I}^1_{X_1} \circ t^{-1}_{X_1} \circ \mathfrak{K}^2_{X_1}(x_1), ...) = (\mathfrak{I}^1_{Y_1} \circ t^{-1}_{Y_1} \circ \mathfrak{K}^2_{X_1} \circ f^1(x_1, ...), ...).$$

f^i is the projection of $f(\cdot)$ onto its ith output.

That is, a commutative diagram as in Figure 5.2 could be drawn for any $f \in \mathfrak{S}$.

5.1.3 Local and Total Representations

It is sometimes, as in Example 5.1.5, useful to define representations on subsystems. We defined a subsystem in Definition 2.1.4. A local representation of a system is a representation of one of its subsystems. A local representation could be derived from a representation of a system, by restricting the representation and interpretation maps.

Definition of a Induced Local representation

Definition 5.1.6 (Induced Local representation)**.** *Suppose that* $\widetilde{\mathfrak{z}} = (\mathfrak{R}_{\widetilde{\mathfrak{z}}}, \mathfrak{S}_{\widetilde{\mathfrak{z}}})$ *is a subsystem of* $\mathfrak{z} = (\mathfrak{R}, \mathfrak{S})$*. Furthermore,* $\{(\mathfrak{K}^{\mathfrak{Y}_X}_X, \mathfrak{I}_X)\}_{X\in\mathfrak{R}}$ *is a representation of* $\mathfrak{z}$*, where* $\mathfrak{Y}_X$ *is a relatum of* $(\mathfrak{R}_\mathfrak{y}, \mathfrak{S}_\mathfrak{y})$*.*

This induces a representation $\{(\mathfrak{K}^{\mathfrak{Y}_{\widetilde{X}}}_{\widetilde{X}}, \mathfrak{I}_{\widetilde{X}})\}_{(\widetilde{X}\in\mathfrak{R}_{\widetilde{\mathfrak{z}}})\subseteq X\in\mathfrak{R}}$ *on* $\widetilde{\mathfrak{z}}$*, where,*

a) $\mathfrak{Y}_{\widetilde{X}} \in \mathfrak{R}_{\widetilde{\mathfrak{y}}}$ *of* $(\mathfrak{R}_{\widetilde{\mathfrak{y}}}, \mathfrak{S}_{\widetilde{\mathfrak{y}}})$*, which is a subsystem of* $(\mathfrak{R}_{\mathfrak{y}}, \mathfrak{S}_{\mathfrak{y}})$ *where* $(\mathfrak{Y}_{\widetilde{X}} \in \mathfrak{R}_{\widetilde{\mathfrak{y}}}) = \{\mathfrak{K}_X^{\mathfrak{Y}_X}(u) \,|\, \forall u \in \widetilde{X}\}$*,*

b) and,

$$\begin{aligned}
\mathfrak{K}_{\widetilde{X}}^{\mathfrak{Y}_{\widetilde{X}}} &: \widetilde{X} \to \mathfrak{Y}_{\widetilde{X}} \\
&\widetilde{x} \mapsto \mathfrak{K}_{\widetilde{X}}^{\mathfrak{Y}_{\widetilde{X}}}(\widetilde{X}) = \mathfrak{K}_X^{\mathfrak{Y}_X}|_{\widetilde{X}}(\widetilde{x}), \\
\mathfrak{I}_{\widetilde{X}} &: \mathfrak{Y}_{\widetilde{X}} \to \widetilde{X}/\sim_{\mathfrak{K}_{\widetilde{X}}^{\mathfrak{Y}_{\widetilde{X}}}} \\
&\widetilde{\mathfrak{y}} \mapsto \mathfrak{I}_{\widetilde{X}}(\widetilde{\mathfrak{y}}) = \mathfrak{I}_X|_{\widetilde{X}}(\widetilde{\mathfrak{y}}).
\end{aligned}$$

$\{(\mathfrak{K}_{\widetilde{X}}^{\mathfrak{Y}_{\widetilde{X}}}, \mathfrak{I}_{\widetilde{X}})\}_{(\widetilde{X} \in \mathfrak{R}_{\widetilde{\mathfrak{z}}}) \subseteq X \in \mathfrak{R}}$ *is called the induced local representation of subsystem* $\widetilde{\mathfrak{z}}$*.*

Remark 5.1.1: $(\widetilde{X} \in \mathfrak{R}_{\widetilde{\mathfrak{z}}}) \subseteq X \in \mathfrak{R}$ in $\{(\mathfrak{K}_{\widetilde{X}}^{\mathfrak{Y}_{\widetilde{X}}}, \mathfrak{I}_{\widetilde{X}})\}_{(\widetilde{X} \in \mathfrak{R}_{\widetilde{\mathfrak{z}}}) \subseteq X \in \mathfrak{R}}$ specifies that $\widetilde{X}$ corresponds to $X \in \mathfrak{R}$, where $\widetilde{X} \subseteq X$. Thus $\widetilde{X}$ is the relata that was obtained by taking the subset of X when defining the subsystem. •

Given a set of subsystems of $\mathfrak{z}$, and their local representations, we can write a total representation of $\mathfrak{z}$ in terms of the local representations, if the union of all the systems is $\mathfrak{z}$. Where the intersections of subsystems are not null, it is necessary that the local representations in the intersections are equivalent, as per Definition 5.1.5. That is,

Definition 5.1.7 (Total Representation)**.** *Suppose we are given a collection* $\{\widetilde{\mathfrak{z}}_i = (\mathfrak{R}_i, \mathfrak{S}_i)\}_{i \in \mathcal{I}}$ *of a system* $\mathfrak{z}$*, such that,* Definition of a Total Representation

$$\bigcup_{i \in \mathcal{I}} \widetilde{\mathfrak{z}}_i = \mathfrak{z},$$

where the union of subsystems was defined in Definition 2.1.5.

For each such subsystem $\widetilde{\mathfrak{z}}_i$*, we are given a representation* $\{(\mathfrak{K}_i, \mathfrak{I}_i)\}_{X \in \mathfrak{R}_i}$*. Then, the collection,*

$$\left\{\{(\mathfrak{K}_i, \mathfrak{I}_i)\}_{X \in \mathfrak{R}_i}\right\}_{i \in \mathcal{I}},$$

is called a total representation of $\mathfrak{z}$ *if, when* $\widetilde{\mathfrak{z}}_i \cap \widetilde{\mathfrak{z}}_j$ *are valid subsystems, the local representations induced by* $\{(\mathfrak{K}_i, \mathfrak{I}_i)\}_{X \in \mathfrak{R}_i}$ *and* $\{(\mathfrak{K}_j, \mathfrak{I}_j)\}_{X \in \mathfrak{R}_j}$ *are equivalent.*

Note that a total representation is not a representation, since there is ambiguity where the relata intersect. However, this ambiguity can be resolved up to the equivalent representations.

5.1.4 Examples of Representations

Let us look at some examples. In these examples, we will be suppressing the notation of representation maps.

Example 5.1.3 Natural numbers: The system of natural numbers we are interested in is $(\{\mathbb{N}, \{0,1\}\}, f_< : \mathbb{N} \times \mathbb{N} \to \{0,1\})$, where for $x, y \in \mathbb{N}$,

$$f_<(x,y) = \begin{cases} 1 & \text{if } x < y \\ 0 & \text{otherwise} \end{cases}. \tag{5.3}$$

$f_<$ is simply a structure map that implies ordering of the natural numbers. The most common representation for this system is the Arabic numeral system. The Arabic numeral scheme expresses the natural numbers in a base of 10. Formally, the symbol space $\mathfrak{Y}_{\text{Ar}}$ of the Arabic numeral system is the collection of all symbols $a_K...a_1a_0$ for $a_i \in \{0,1,2,...,9\}$, and $K \in \mathbb{N}$ is the largest number such that for any $k > K, a_k = 0$; this is to prevent leading 0s. The set $\{0,1,2,...,9\}$ is ordered, and can be multiplied and added in the usual, expected way. The notation $...a_2a_1a_0$ isn't a product of numbers, but rather each number written one after the other.

Then, the representation map $\mathfrak{K}_{\text{Ar}}$ is given by:

$$\begin{aligned} \mathfrak{K}_{\text{Ar}} : \mathbb{N} &\to \mathfrak{Y}_{\text{Ar}} \\ n &\mapsto \mathfrak{K}_{\text{Ar}}(n) = a_K...a_2a_1a_0, \end{aligned} \tag{5.4}$$

where,

$$\begin{aligned} a_0 &= n \bmod 10, \\ a_1 &= \frac{(n - a_0)}{10} \bmod 10, \\ a_2 &= \frac{(n - 10a_1 - a_0)}{100} \bmod 10, \\ &\vdots \\ a_K &= \frac{\left(n - \sum_{i=0}^{K-1} a_i 10^i\right)}{10^K} \bmod 10. \end{aligned}$$

For example, 42 is a_1a_0, where $a_0 = 2$ and $a_1 = 4$. Note that this representation map is exact. Finally, the interpretation $\mathfrak{I}_{\text{Ar}}$ is,

$$\begin{aligned} \mathfrak{I}_{\text{Ar}} : \mathfrak{Y}_{\text{Ar}} &\to \mathbb{N} \\ \mathfrak{I}(a_K...a_1a_0) &\mapsto \mathfrak{I}_{\text{Ar}}(a_K...a_1a_0) = \sum_{i=0}^{K} a_i 10^i. \end{aligned} \tag{5.5}$$

As an interesting note, since the representation map is exact, this map is actually the inverse of the representation map.

As per Definition 5.1.4, we must show that $\mathfrak{I}_{\text{Ar}}$ is consistent with the structure of the system. For this, we will assume that the representation map of $\{0,1\}$ is the same, since its constituents are also natural numbers. We must show that for $x, y \in \mathbb{N}$, $f_<(\mathfrak{I}_{\text{Ar}} \circ \mathfrak{K}_{\text{Ar}}(x), \mathfrak{I}_{\text{Ar}} \circ \mathfrak{K}_{\text{Ar}}(y)) = \mathfrak{I}_{\text{Ar}} \circ \mathfrak{K}_{\text{Ar}}(f_<(x,y))$. That is, does the representation have the same ordering as the natural numbers?

To show this, we must prove that there is an ordering in $\mathfrak{Y}_{\text{Ar}}$ that is induced by $\mathfrak{K}_{\text{Ar}}$; if $x < y$ then $\mathfrak{I}_{\text{Ar}} \circ \mathfrak{K}_{\text{Ar}}(x) < \mathfrak{I}_{\text{Ar}} \circ \mathfrak{K}_{\text{Ar}}(y)$. This follows naturally, since

$\mathfrak{I}_{\text{Ar}}$ is the inverse of $\mathfrak{K}_{\text{Ar}}$. Thus, $\mathfrak{I}_{\text{Ar}} \circ \mathfrak{K}_{\text{Ar}}$ is the identity map on $\mathbb{N}$, which implies that the required condition holds. ▲

Example 5.1.4 Linear maps: The second example we will look at is the space of linear maps between the vector spaces X and Y. Firstly, we define representations of a vector space V; the dimension of V is d. The system here was described in Equation (2.5). Let us choose the symbol space $\mathfrak{Y}_{\text{Ve}}$ to be $\mathbb{R}^d$. A possible representation map $\mathfrak{K}_{\text{Ve}}$ is,

$$\begin{aligned} \mathfrak{K}_{\text{Ve}} : V &\to \mathbb{R}^d \\ v &\mapsto \mathfrak{K}_{\text{Ve}}(v) = (a_1, ..., a_d), \end{aligned} \tag{5.6}$$

where, $a_1, ..., a_d \in \mathbb{R}$ are such that, for the independent vectors $e_1, ..., e_d \in \mathbb{R}^d$,

$$v = \sum_{i=1}^{d} a_i e_i.$$

We can immediately notice that $e_1, ..., e_d \in \mathbb{R}^d$ are basis vectors, and $a_1, ..., a_d$ are called the vector coordinates. This immediately gives us a valid interpretation $\mathfrak{I}_{\text{Ve}}$,

$$\begin{aligned} \mathfrak{I}_{\text{Ve}} : \mathbb{R}^d &\to V \\ (a_1, ..., a_d) \mapsto \mathfrak{I}_{\text{Ve}}(a_1, ..., a_d) &= \sum_{i=1}^{d} a_i e_i, \end{aligned} \tag{5.7}$$

where $e_1, ..., e_d$ form a basis. To prove the consistency of this representation with the structure maps, we must show that $\mathfrak{I}_{\text{Ve}} \circ \mathfrak{K}_{\text{Ve}}$ is linear. This is of course a well known result in linear algebra, and will not be repeated here.

Now, let us denote X and Y as vectors spaces of dimension m and n respectively. A system of linear maps $\mathcal{F}_{\text{lin}}$ between X and Y can be written as $(\{\mathcal{F}_{\text{lin}}, X, Y\}, \{f_{\text{lin}}\})$, where,

$$\begin{aligned} f_{\text{lin}} : \mathcal{F}_{\text{lin}} \times X &\to Y \\ (g, x) &\mapsto f_{\text{lin}}(g, x) = g(x). \end{aligned}$$

$f_{\text{lin}}(g, x)$ is map that simply evaluates a linear map g at a point $x \in X$. We will choose the symbol space for this to be the matrix space $\mathbb{R}^{m \times n}$. Let us adopt the representation $(\mathfrak{K}_{\text{Ve}}, \mathfrak{I}_{\text{Ve}})$ for representations $(\mathfrak{K}_X, \mathfrak{I}_X)$ and $(\mathfrak{K}_Y, \mathfrak{I}_Y)$ of $\mathbb{R}^m$ and $\mathbb{R}^n$ respectively. Then, a representation map $\mathfrak{K}_{\text{lin}}$ for this space of linear maps can be defined as,

$$\begin{aligned} \mathfrak{K}_{\text{lin}} : \mathcal{F}_{\text{lin}} &\to \mathbb{R}^{m \times n} \\ g &\mapsto \mathfrak{K}_{\text{lin}}(g) = \mathbf{A}, \end{aligned} \tag{5.8}$$

where for any $x \in X$ and $(y \in Y) = g(x)$,

$$\mathbf{A}\mathfrak{I}_X \circ \mathfrak{K}_X(x) = \mathfrak{I}_Y \circ \mathfrak{K}_Y(y).$$

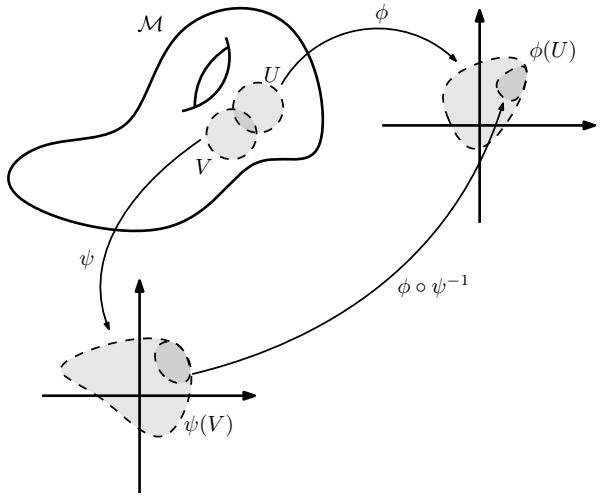


Figure 5.3. A topological manifold M with examples of its coordinate charts. We see here that despite being a complicated topological entity (See the hole in the middle), locally, it is equivalent to $\mathbb{R}^2$. Further, note that the chart transition map $\phi \circ \psi^{-1} : V_{\mathbb{R}^2} \to U_{\mathbb{R}^2}$ exists because a homeomorphism is invertible and continuous, since the composition of continuous maps remains so.

It's interpretation $\mathfrak{I}_{\text{lin}}$ is once again implied. Here, since $\mathbf{A}$ describes a map, we will define the interpretation of $\mathbf{A}(x)$. Thus,

$$\begin{aligned} \mathfrak{I}_{\text{lin}}(x) : \mathbb{R}^{m\times n} &\to \mathcal{F}_{\text{lin}} \\ \mathbf{A} &\mapsto \mathfrak{I}_{\text{lin}}(\mathbf{A}) = \mathbf{A}\mathfrak{I}_X \circ \mathfrak{K}_X(x). \end{aligned} \tag{5.9}$$

Once again, the proof of consistency is a known result in linear algebra. ▲

Example 5.1.5 Topological Manifolds: In Example 2.1.2, we showed how a topological space can be written in terms of a system. A topological manifold is a topological space, with the addition of an atlas (see Appendix A.3). Not all topological spaces can be made into manifolds; the requirements to allow this are given in Definition A.3.1. An atlas $\mathcal{A}$ contains bi-continuous maps from the open sets of the topological space onto open sets of some $\mathbb{R}^d$; see Figure 5.3. Each such map, called a chart map, can be seen as a *local* representation map.

In it's totality, a topological manifold can be seen as a total representation of a topological space system $(\{X\}, \{\mathcal{O}_X, \mathfrak{S}_{\text{top}}\})$. To see this, we first construct a subsystem $\mathfrak{z}U = (\{\{U\}, \{\mathcal{O}_X|_U, \mathfrak{S}_{\text{top}}|_U\}\})$, where $U \in \mathcal{O}_X$ such that $\exists(U, \phi_U) \in \mathcal{A}$, and $\mathcal{O}_X|_U$ is the subset topology. The chart map $\phi_{\mathfrak{z}U} : U \to (U_{\mathbb{R}^d} \subseteq \mathbb{R}^d)$ is a homeomorphism to a subset of $\mathbb{R}^d$. Thus, we can say that $U_{\mathbb{R}^d}$ is the symbol space, $\phi_{\mathfrak{z}U}$ is the representation map, and $\phi^{-1}_{\mathfrak{z}U}$ is the interpretation.

If we construct such subsystems for all charts of the atlas, we have local representations for each of them. Since a requirement of an atlas is that chart maps are consistent where their domains intersect, the local representations

defined here are equivalent. Thus, the union of all such subsystems and their representations gives us a total representation of the topological system. ▲

5.1.5 Properties of Representations

This definition, and the previous examples illustrates some important properties of a representation of system.

A representation is not unique. Examples 5.1.3 and 5.1.4 are only single examples of potential representations of their respective systems. There are many other equivalent representations that could be used. Take the natural numbers. Other numeral systems that can represent $\mathbb{N}$ include the binary system, and the Roman numeral system. The former is similar to the Arabic system, but its space of symbols is derived from $\{0, 1\}$, and its representation and interpretation maps are defined in terms of powers of 2. The Roman system on the other hand, uses a symbol space that derived from primitives that include {I, V, X, L, C, D, M}, among others. For example, 42 is 101010 and XLII respectively.

The representations of vector spaces and spaces of linear maps in Example 5.1.4 were derived from a particular basis. As is well known in linear algebra, we can carry out simple changes of bases to obtain equivalent representations. For example, if $(g \in \mathcal{F}_{\text{lin}}) : X \to Y$, and its description is $\mathbf{A}$ w.r.t. the bases e_X and e_Y for X and Y respectively. Choosing different bases for X and Y will induce a different, but equivalent representation on the space of linear maps.

There is a similar argument for the local representations given by the chart maps of a topological manifold. At the intersections of the chart domains, the local representations are equivalent by definition . Take the linear maps again. If the transformations of the bases of X and Y are given by $\mathbf{B_X}$ and $\mathbf{B_Y}$, then the isomorphism g will be $\overline{\mathbf{A}} = \mathbf{B_Y A B_Y^{-1}}$.

A representation can be approximate. However, not all valid representations are equivalent. Previously, we alluded to a difference between exact and approximate representations. There can be no isomorphism between exact and approximate representations. So far, the representations we have described so far were exact. Approximate representations do not uniquely describe every element of the system being represented. However, they must, by definition, still be consistent with the structure of the system.

For example, a possible approximate representation of the natural numbers is to first map $n \in \mathbb{N}$ to its closest and smallest even number. That is, $f_{\text{even}}(n) = \lfloor n/2 \rfloor$. Then the representation map $\mathfrak{K}_{\text{Ar}}^{\text{even}} = \mathfrak{K}_{\text{Ar}} \circ f_{\text{even}}$. The interpretation remains the same. Note that the output space of $\mathfrak{K}_{\text{Ar}}^{\text{even}}$ is a subset of $\mathfrak{Y}_{\text{Ar}}$.

Similarly, we can approximate vector spaces by projecting down to a vector subspace. For example, for a vector space of dimension d, the new representation map could only report the first $p < d$ outputs of the original representation

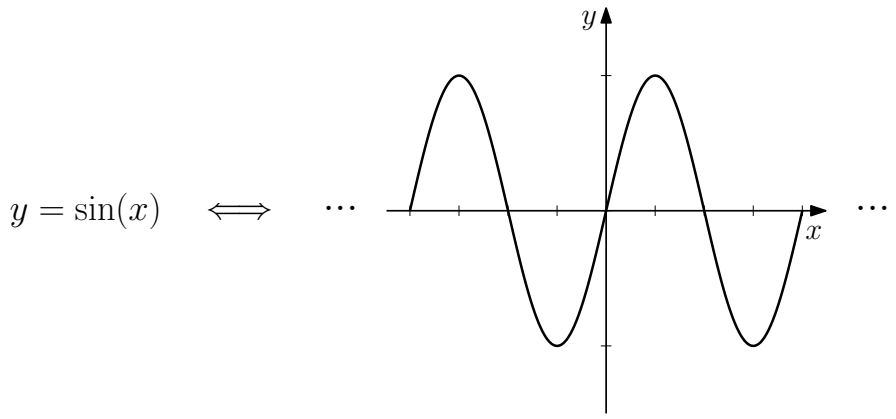


Figure 5.4. Two representations of the sin function. On the left, it is written in mathematical notation; on the right, it is drawn as a graph.

map given in Equation (5.6). This would mean that information regarding the remaining bases is lost; however, working within this representation, the structure maps are still satisfied.

Representations can be differently useful. Typically, a representation of a system is created so that the system can be used for some purpose. For example, perhaps a representation of the natural numbers is chosen so that we can describe, in notation, the operation of addition. Here, the system of natural numbers would contain an additional structure map $f_+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}; f_+(n_1, n_2) = n_1 + n_2$; it is trivial to show that the Arabic, binary and Roman numeral systems are consistent with f_+. For example, in the Arabic system, $42 + 2 = 44$. In order to carry out an arbitrary addition in this representation, we follow a set of rules. Given $a^1_K ... a^1_1 a^1_0$ and $a^2_H ... a^2_1 a^2_0$, we start from the right, and add a^1_i and a^2_i; if $a^1_i + a^2_i \geq 10$, we do a carry over operation.

The rules are much more complicated for the Roman numeral system. If addition was the reason for developing a representation of the natural numbers by humans, then the Arabic is more suitable than the Roman system. Alternatively, if the addition was to be carried out by machines, then the binary representation is the clear best.

In terms of Example 5.1.4, the choice of bases of vector spaces could have computational consequences. For example, orthonormal bases make it easier to *compute* the description of a vector, since each vector coordinate is the dot product between the vector and the corresponding basis vector. If possible, a representation of V can be chosen such that the basis vectors are the eigenvectors of the linear map. Then, the description of this linear map would be a diagonal matrix. Such a representation is easier to study.

An interesting example that illustrates this is shown in Figure 5.4, which shows two representations of the sin function. The first (on the left) is the mathematical notation for this function. With this description comes an understanding of

its properties; these include its relation to circles, the trigonometric identities, perhaps its relation to a notion of periodicity, and its role in, say Fourier analysis. This notation evokes our abstract understanding of this function.

The second (on the right), is a graphical representation of it. Here, it is represented as an image, where all pixels are colored white except for the points that (approximately) correspond to different values of $y = \sin(x)$. In this particular example, we sampled 100 points of the function. We differentiate this representation from simply listing the (x, y) values of the function. This representation evokes our visual understanding of the sin function. It also tells us of the periodic nature of the function; it is however difficult to see its connection to circles. This graphical view helps us imagine when and where a sin function could be useful; for example, it helps convince us that they must play a role in modelling waves.

5.2 Learning in terms of Representations

Transfer requires local structure

From the point of view of solving a single learning problem $(\mathfrak{t}, \mathcal{L})$, as long as the solution $\mathfrak{L}(t) \in \Theta_\kappa$ is sufficiently good, we do not necessarily care about how neighbouring models correspond to neighbouring tasks. That is, a single task learning problem can be purely seen as an optimisation problem. However, when considering problems of transfer, that inherently involve multiple learning tasks from the same presumed $\mathfrak{T}$, this consideration becomes important, since, as we will see in Chapter 6, transfer involves learning structure on the entire $\mathfrak{T}$.

Correspondence of learning with represetations

In the ideal case, we want the model space Θ_κ and the learning algorithm $\mathfrak{L}$ to form a representation of the $\mathfrak{T}$, where the model space is the space of symbols, and the learning algorithm is the representation map. If this is the case, we will say that the Θ_κ and $\mathfrak{L}$ are *well-behaved*.

The interpretation of each model is given by its architecture map κ. We mentioned in Section 2.6.1 that we assume that $\mathfrak{T}$ is a smooth manifold, which itself can be seen as a system. We also saw in Section 5.1.1 that representations are structure preserving; without proof[3], we state that representations of a smooth manifold can be given by smooth, surjective maps onto another smooth manifold; we have already assumed that Θ_κ is a smooth manifold.

When is a learning algorithm smooth?

The conditions for which the learning algorithm is smooth is complicated. Let us assume that we are using gradient descent, where the initial model $\mathfrak{m}_0$ is constant for all learning tasks. In this case, the smoothness of the learning algorithm depends on how the singular points of $\mathcal{L}_\mathfrak{t}$ vary as we change the task $\mathfrak{t}$. Stating that the loss function is smooth might not be enough. For example, [93] showed that even for a seemingly innocuous system, where the governing equations were well behaved, the resulting change in dynamics is fractal in nature. Therefore, for the present work, we assume that the learning algorithm is well behaved locally.

[3]This proof would be similar to those given in Section 5.1.1.

Local task and model spaces

By locally, we mean that there exists a submanifold of $\mathfrak{T}$, and a submanifold of Θ_κ where the learning algorithm is a representation map. Henceforth, when we refer to the task space, or the model space, we will be referring to these, unless specified otherwise. Another assumption here is that each model in the submanifold corresponds to a *suitable* solution of a task, in that its loss value is sufficiently small. This resolves an issue where the interpretation map maps to the space of proxy maps, rather than the task space, as would be required by Definition 5.1.3. However, since each model now is a suitable proxy for a task, we proxy this submanifold as being the task space.

In practice, we do not actually start with the $\mathfrak{T}$. So far, we used it as an abstract tool to conceptualize and build the machinery to talk about representations. We would typically consider some tasks that we think belong to a single $\mathfrak{T}$, and determine the loss function, learning algorithm and model space. In this way, instead of ensuring that the learning algorithm matches the criteria of a representation map, we assume a space of tasks that allows the entire setup to be a representation. The simplest way to achieve this would be to assume that locally (as above) the task space is equivalent to the model space.

Induced topology on Task Space

Another setting could be if the elements of a task space are given, but some other structure (such as its topology) isn't. We could induce topologies, for example, by stating that $\text{preim}_{\mathfrak{L}}(U^{\Theta_\kappa}) \subseteq \mathfrak{T}$, for $U^{\Theta_\kappa} \in \mathcal{O}_{\Theta_\kappa}$ is an open set in $\mathfrak{T}$. It is trivial to show that this is a valid topology for $\mathfrak{T}$, using the fact that unions and intersections commute with preimages. In this topology, the learning algorithm is guaranteed to be continuous, by definition.

This topology does not however say anything about whether the loss function is continuous under the product topology, even if we assume that $\mathcal{L}_\mathfrak{t}$ is continuous. In order for $\mathcal{L}$ to be continuous, $\text{preim}_{\mathcal{L}}(U^{\mathbb{R}}) = (U^{\Theta_\kappa}, V)$ must be an open set. Since $\mathcal{L}_\mathfrak{t}$ is continuous, we know that U^{Θ_κ} is open. However, for $V \subseteq \mathfrak{T}$ to be open, $\mathfrak{L}(V)$ must be an open set in Θ_κ. In other words, it must be the case that $U^{\mathbb{R}} = \mathcal{L}(\mathfrak{L}(V), V)$. There is no guarantee that this true; hence we cannot generally say if $\mathcal{L}$ is continuous.

5.3 Relation to Category Theory

Category theory as the structure of structure

In light of the theory developed so far, it might be prudent to discuss some category theory. Category theory formalises the structure of mathematical structure. One can observe that many structures in mathematics can be described a sets, properties of those sets, and maps that preserve these properties. For example, topology deals with sets with topologies and continuous maps that preserve topologies. In differential geometry, smooth manifolds come with smooth structures, and smooth maps that preserve these structures. Measure spaces, and measure preserving maps are seen in measure theory.

Brief introduction to Categories

Thus, category theory emphasizes *composable arrows* between elements of a *category* that can form *commutative diagrams* [62]. A category consists of a

class of objects and a class of morphisms whose domains and codomains are contained in the class of objects. Typically, the set of objects are spaces, such as topological spaces in the category of all topological spaces **Top**.

Morphisms are seen as the structure preserving maps, such as continuous maps in **Top**. In the class of morphisms, there must exist an identity map for each object. Finally, a category is equipped with a binary composition operation that is associative. This operation takes morphims and spits out morphisms.

Summary of previous chapters

In the previous chapters, we have seem very similar concepts being described in our definitions of systems and representations. Systems contain in their construction a set of objects (possibly other systems), a set of domains and codomains, and maps between them. The set of (structure) maps contains the identity map for each element in the relata. Systems can be inherited from, or derived such that structure maps are preserved. Representations are structure preserving maps, with commutative diagrams being drawn to depict equivalences between such maps.

Thus, coincidentally, there appears to be an undeniable correspondence between the present work and category theory. The present work seems to have begun a description of the category of learnable problems and solutions, and to cast learning as the expression of particular morphisms in this category. We do not attempt to formalise this correspondence here, and will leave this work for the future. Perhaps, in doing so, our theory and its application can be greatly streamlined and enhanced by the literature that exists for category theory.

This page was left intentionally blank.

II

The Structure of Transfer

This page was left intentionally blank.

Chapter 6

Defining Transfer

Transfer in the context of learning, is the notion where properties of *knowledge* learned from one context can be re-used in another. This is a concept that we, as humans, are intimately familiar with. When we learn something new, we often try to think of it in terms of what we already know, allowing us to learn without much redundancy [91, 26]. In fact, without transfer, learning would be situational, where each piece of knowledge or skill we acquire can only be applied to the context in which it was learned. Considering the range and depth of different situations that we encounter, having to learn from scratch for each of them would be computationally infeasible, and incredibly inefficient!

Transfer avoids relearning concepts

Transfer allows us to avoid relearning concepts that are useful across multiple problems. Consider addition as it is used in physics, mathematics, accounting, and simply everyday life. Since we are able to identify that these contexts require the same notion of addition, with the differences being *what* is being added, and *how many* summands there are, we are required to learn addition only once. In cases where such transfer isn't possible, or if we aren't capable of transfer, we would have to learn how to add for each case separately.

Transfer is almost certainly a necessary component for efficient learning. In order to study transfer, we must first produce an airtight definition of it. This definition, in addition to being stated clearly and concisely, must satisfy our intuitions about transfer, while also allowing us to find other, previously unseen examples of this process. While we may, in a general sense, state that transfer is the sharing of knowledge between different contexts, we require a definition that is explicit about its constituent components. In particular, we require detailed descriptions of the following:

- what allows for transfer,
- what does it mean to share knowledge, and
- what does it mean to transfer.

In addition, since we want to study transfer mathematically, the definition must lend itself to a mathematical formalisation. A key issue with the definitions

provided in the literature is that they do not allow us to understand transfer. For example, Baxter's definition of transfer is intuitively sound, and he showed several important properties of transfer using his model [7]. However, it is difficult to discern neither *what* is being transferred, nor the *structure* that allows for transfer. Baxter's model provides a technique for learning a suitable bias, but it doesn't give us a sufficient definition of transfer, since such a definition must be extracted from his method as merely the process that occurs when carrying out the learning of bias. We believe that an elementary definition of transfer can be made; this should allow for an intellectually satisfying pathway to study transfer, since identifying *what* we want a learning algorithm to learn should precede the dissemination of *how* we should learn it.

Therefore, in the present chapter, we will produce a definition of transfer that satisfies our requirements, and addresses these issues. In Section 6.1, we will see that transfer can be found in many areas, ranging from human learning, to how we organise knowledge. We will see that an underlying thread that connects all these is a notion of a transfer, a bias that is exploited by transfer. The existence of symmetries between knowledge and problems is what enables us to transfer, and explicitly defining transfer using symmetries will be beneficial for us.

6.1 Everyday Examples of Transfer

The idea of transfer can be seen in many other aspects of the human experience. In this section, we discuss a few examples of such cases. These examples are not from the context of learning in machines.

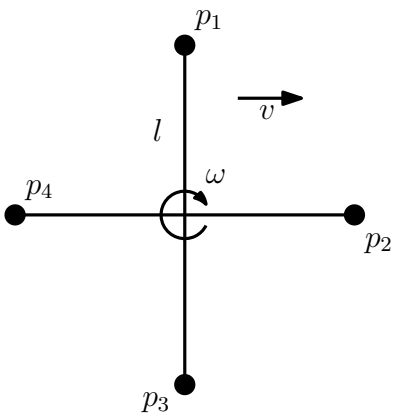


Figure 6.1. A rotating and translating set of point masses. The rods connecting the points together are massless. The length of each connecting rod, measured end to end, is $2l$. From the center, the set of points is rotating with angular velocity ω and translating horizontally with a linear velocity of v.

Abstraction is a form of transfer

Example 6.1.1 Abstraction: Consider the problem of trying to model the motion of the 4 point masses $p_1, ..., p_4$ in Figure 6.1. As a first attempt, we could describe each point mass independently of each other. This would generate

the following equations:

$$\begin{aligned} x_1(t) &= \cos(\theta_0^1 + \omega t) + vt + x_0^1 \\ y_1(t) &= \sin(\theta_0^1 + \omega t) + y_0^1, \end{aligned} \tag{6.1}$$

$$\begin{aligned} x_2(t) &= \cos(\theta_0^2 + \omega t) + vt + x_0^2 \\ y_2(t) &= \sin(\theta_0^2 + \omega t) + y_0^2, \end{aligned} \tag{6.2}$$

$$\begin{aligned} x_3(t) &= \cos(\theta_0^3 + \omega t) + vt + x_0^3 \\ y_3(t) &= \sin(\theta_0^3 + \omega t) + y_0^3, \end{aligned} \tag{6.3}$$

$$\begin{aligned} x_4(t) &= \cos(\theta_0^4 + \omega t) + vt + x_0^4 \\ y_4(t) &= \sin(\theta_0^4 + \omega t) + y_0^4. \end{aligned} \tag{6.4}$$

Here, $(x_0^1, y_0^1), ..., (x_0^4, y_0^4)$ denote the initial positions of $p_1, ..., p_4$ respectively. Further, $\theta_0^i = \arctan(\frac{y_0^i}{x_0^i})$. Modelling this problem in this way is tedious; it will only get worse if the number of point masses increases. It is also repetitive, where we have to repeat several properties that are constant between all the point masses.

Firstly, we note that point masses are connected to each other, and are all at a constant distance of l away from the center of rotation. Furthermore, they are all offset from each other by $\pi/2$rad. Finally, all the particles are rotating with a constant and similar angular velocity, and are moving linearly with a constant and similar velocity[1]. Putting these together, we can write the following, generalised equations of motion.

$$\begin{aligned} x_n(t) &= \cos((n-1)\pi/2 + \omega t) + vt + (x_0^c + l\cos((n-1)\pi/2)) \\ y_n(t) &= \sin((n-1)\pi/2 + \omega t) + (y_0^c + l\sin((n-1)\pi/2)), \end{aligned} \tag{6.5}$$

where the point (x_0^c, y_0^c) is the initial position of the center of the system. This can be further simplified as,

$$\begin{aligned} x_n(t) &= \cos((n-1)\pi/2 + \omega t) + l\cos((n-1)\pi/2)) + x_c(t) \\ y_n(t) &= \sin((n-1)\pi/2 + \omega t) + l\sin((n-1)\pi/2)) + y_c(t), \end{aligned} \tag{6.6}$$

where,

$$\begin{aligned} x_c(t) &= vt + x_0^c \\ y_c(t) &= y_0^c. \end{aligned} \tag{6.7}$$

That is, we can describe the motion of each point mass as the motion of the center and a correction term that corresponds to the differences between each point mass (and the angular velocities)[2]. We have abstracted out the fact that

[1] Note that these properties were already used when writing Equations 6.1-6.4.

[2] This abstraction isn't unique. We could, for example, equivalently abstract out the motion of a single point mass, rather than the center of rotation. In fact, this can be continuously changed to any other point in $\mathbb{R}^2$.

the motion of the center is a consistent property between those of the point masses. In doing this, we have modelled the problem in terms of what was abstracted out, and the remaining properties required to describe the motion appropriately.

There is an important relationship between abstraction and transfer[3]. As in the example above, to transfer is to identify what is consistent between two concepts, and to express related concepts in terms of what can be abstracted out, and what cannot. ▲

School curricula involve transfer

Example 6.1.2 Human learning: The transfer of learning in humans has been studied extensively [91, 101], and is often exploited when we develop curricula for teaching new knowledge and skills [81, 56]. In most school curricula, we first learn general skills that can be used in more complex, yet specialised knowledge we tackle later on. An obvious example are the basics of mathematics (such as algebra, calculus and graphical visualisation), which are then utilised in the more advanced topics of physics, economics, and mathematics itself. Transfer is clearly important in human learning.

Transfer is ubiquitous in human learning

In psychology, and the theory of learning, transfer is defined as the use, or influence of previously acquired knowledge[4] in application to learning new knowledge [99, 1], or knowledge being applied in novel or similar ways and situations [91]. Thorndike [108] proposed that transfer occurs in situations that contain *identical* elements between them, and therefore can be satisfied with similar responses; the need for exactness weakens this theory [91]. Generalisation theories [91, 95] relaxes this constraint[5], allowing transfer to occur in *similar* situations. ▲

Example 6.1.3 Organisation of knowledge: Another example of such abstraction, and therefore transfer is in the way by which we organise and communicate knowledge. Here, knowledge is the collection of discussions and writings that we have amassed as a society over the years. Consider any piece of scientific writing. More often than not, such a document will contain a literature review, and/or a background section. Such sections describe the knowledge on which the present writing is built upon. It is the knowledge that had been generated by others, and must be assumed when reading the novel scientific result.

Kuhnian structure of science

Such ideas are succinctly described in the Kuhnian philosophy of science [52].

[3]It should be noted that not all examples of abstraction involves transfer. In a general lumped mass abstraction, a collection of particles is lumped as a solid object, we could model its motion in terms of a point mass. However, it does not allow us to identify the motion of a *particular* particle, and transfer to other particles.

[4]In the present work, *knowledge*, as it pertains to humans, is used as an umbrella term that includes knowledge (what we know), skills (what we can do), and strategies (when to use our knowledge and skills).

[5]It should be noted that *generalisation* and *identical elements* are prescribed in different theories (connectionism and classical conditioning respectively). We ignore these distinctions for clarity.

Kuhn posits a process by which scientific discovery occurs. Given a problem that is to be solved, initial thinkers would often independently generate theories from first principles. Each such school of thought would describe or explain a subset of the observations made regarding the problem to be solved. They would each be written in terms of the point of view taken by their generating school. What should be noted is that at this stage, each school would be very disparate from each other, and would be very biased by their particular beliefs and tendencies.

Paradigms and transfer

Following this, there will come a time when a unifying theory is developed, that takes the common aspects of each school of thought, and provides a general expression of the discovery that can be applied to the many points of view that we taken previously. Such a theory is called a *paradigm*. A paradigm marks a point of maturity for a field of study, and contains the concepts and skills that a practitioner of the field must be familiar with, as a minimum. Examples of paradigms include the theory of motion by Newton, and Maxwell's theory of electromagnetism. These are the concepts that are common to the different problems that a field tries to solve, and can therefore be transferred between them.

Transfer in language

This type of abstraction is used whenever we communicate. We often make references to common pieces of knowledge to aide in an explanation. In fact, communication can only occur through a common understanding. This includes both the spoken and unspoken aspects of language; we see that confusions occur when we assume certain cultural norms that would be different when communicating with an individual from a differing culture. ▲

Example 6.1.4 Compression: An important consequence of transfer is its ability to compress knowledge. If several pieces of knowledge are to be represented in language, or otherwise, transfer can allow us to do so without repetition. That is, if the pieces of knowledge can be represented in a way that separates what is common between them from what is different, we will only need to express what is common once. This was already seen in Section 6.1.1. We can also see this idea exploited in common compression techniques in computer science, such as the interframe compression of video [2].

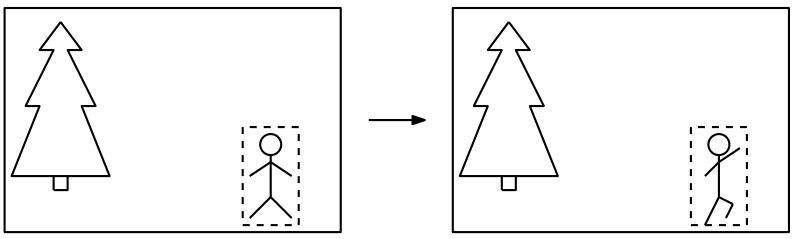


Figure 6.2. Interframe compression, illustrated using two images, or frames set in the same scene. The key, and only difference between these can be found inside the dashed box, where the human changes poses.

Interframe compression

Interframe compression works by identifying a group a consecutive frames where

consistent subset of the frames change over time (hence the name *inter*frame). The first of these frames will be called the key frame, and will be expressed in its entirety. We will also store the boundary of the region that changes. Then, the following frames will be expressed in terms of their differences within this boundary, with the key frame. Thus, we will not be repeating the portion of this group of frames that is equal to each other. This technique exploits the fact that often, many parts of video stay constant. This could for example, be because of an object moving in a constant background, as in Figure 6.2. In more complicated forms of this compression, motion prediction and the like can be used to compress video that contain motion of objects that are predictable; for example, the movement of the camera.

Note that this form of compression is lossless; it is merely a re-writing of each frame with a separation between the aspects of the frames that are consistent, and those that are different. We can therefore avoid redundant expression of former. ▲

6.2 Bias and Transfer

Transfer makes processes simpler

Transfer, in the context of a process such as learning, expression, or storage, is a means of making said process *simpler*. Transfer in learning has the additional goal, or perhaps consequence, of making the learning better. Transfer achieves this by exploiting redundancy; this is most clearly seen in Example 6.1.4, where it is clearly redundant to repeatedly specify elements of a sequence of images that do not *change* between them.

Section 6.1 showed that what we intuitively think of as transfer occurs in many different contexts; this could be in learning, expression, or storage, among others. For simplicity, let us, without distinction, denote these contexts as *transfer between objects*.

Objects and their properties

Suppose we are given a set of objects O. Associated with this set of objects is a set of properties P. Each property $(p \in P) = \{s_i\}_{i \in \mathcal{I}}$ consists of a set of statements that can be assigned to an object; each object must be assigned a statement from all properties. Given an object $o \in O$, it's characterisation $\mathcal{C}(o)$ is the set of statements from each property that is assigned to it. The characterisation of each object is unique, and thus, each object can be uniquely identifiable from its characterisation. A simple example of this is shown in Figure 6.3.

What do we mean by complexity

Without a loss in generality, we will assume that all statements in all properties are used at least once. Let us call the complexity of characterising an object as the number of elements in O. The larger $|O|$ is, the higher the complexity of characterisation is. In a sense, this measures how hard it is to uniquely identify an object in O.

This measure of complexity directly relates to other notions of complexity we might have about a process in which transfer is involved. In storage (as in

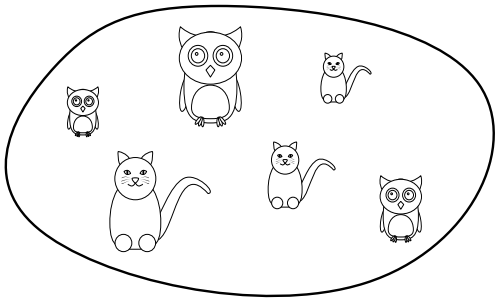


Figure 6.3. A zoo of animals. Here, the set of properties is $\{p_{\text{size}} = \{1, 2, 3\}, p_{\text{genus}} = \{\text{cat}, \text{owl}\}\}$. Each animal in the zoo can be identified by its size and genus.

Example 6.1.4), the number of bits (and therefore memory) used to represent an image correlates with the bitdepth and resolution of an image. In learning in machines, the set of properties is the number of parameters of the parametric model space[6], and the complexity would be the size of the parametric model space. The size of the parametric model could be measured against a notion of *volume*, or, if we assume a manifold space, its dimensions, which correlates with the number of parameters. Typically, the difficulty of learning correlates both with size of the model space[7], and the number of parameters [92, Chapter 6].

Transfer reduces complexity

A goal of transfer is to reduce the complexity of characterising an object o, given the characterisation of another object o_0. This can happen only if there is some overlap between $\mathcal{C}(o)$ and $\mathcal{C}(o_0)$. As an example, consider the zoo in Figure 6.3. Suppose $\mathcal{C}(o) = (\text{owl}, 1)$ and $\mathcal{C}(o_0) = (\text{owl}, 2)$. Here, both animals are owls, and differ only in their size. Thus, if we knew that o does in fact have the same genus of o_0, we do not need to characterise (learn, express, or store) this property. The genus of o can be *transferred* from o_0.

Bias

This process identifies a *bias* that can be assumed about a subset of a set of objects. If we can say that 2 objects belong to a particular, proper subset of objects, and we have knowledge of the characterisation of the bias, we can exclusively work with the subset, rather than the full set. Since the size of a proper subset is strictly smaller than that of the full set, the characterisation is less complex.

Transfer and bias

Therefore, without lack of generality, it can be stated that *transfer is the process of identifying bias, and applying it to create subsets.* This is clear in Section 6.1. In Example 6.1.1, the bias is the arrangement of the point masses, and initial linear and angular velocities. The pre-assumed knowledge is a bias of any learning curriculum, or scientific paradigm; in the former, curricula are designed so that knowledge from earlier years are needed (up to an extent) for topics of later years. Finally, the fixed pixels, their values and positions are

[6]Remember, we do not consider non-parametric models in the present work.
[7]This is called the hypothesis space in SLT literature.

the bias in inter-frame compression.

Transfer in learning. As stated in [7], transfer in learning can been see as the *automatic learning of bias*. The importance of transfer in learning is accentuated by the importance inductive bias in learning. To a practitioner of machine learning, it is well known that *inductive bias* is important. For a given learner (as in Chapter 4), its inductive biases generally are its properties, other than the training data, that make certain solutions, or *generalisations* more likely than others [68]. That is, given knowledge of the past (the data), an inductive bias makes some theories (the solution) about predicting the future (application of the solution) more desirable than others. Biases are assumptions that are made, either implicitly, or explicitly, about the learning problem, and are embedded in the choices made when a solution is designed. In the structural view of machine learning, such biases come in the form of the model architecture that is chosen.

Model architectures provide bias

If our learning problem is regression, the model architecture, given the finiteness of its parameters, biases the potential description of the unknown structure map to be within model space of the architecture. That is, if we had chosen a linear architecture, our description can *only* ever be a linear function (as opposed to any other map between the input and output spaces). In reinforcement learning (RL), the form of its reward function (which directly informs the loss of a RL problem) often produces policies that are surprising; this is fixed by techniques of reward shaping [71].

No-Free Lunch theorems

By the No-Free Lunch (NFL) theorems [124, 125, 92], and/or the Law of Conservation of Generalisation Performance [88], we know that a universal learner is impossible. A universal learner will perform better than any other learner over all tasks[8]. NFL theorems show however, that if a learner performs well in some tasks (compared to another learner), there will exist other tasks that it performs worse than the other learner, which contradicts a universal learner.

Bias is needed

The argument of an NFL theorem, or the Law of Conservation of Generalisation Performance [88], stems from a fundamental inability of *unbiased* induction to distinguish between learning solutions that match in training performance (that is, w.r.t. the training data), but differ in unseen examples. An unbiased learner would say that all possible outcomes in the unseen regions are *equally possible*; if one was to jump up, there could be equal probability of them falling down, or that an intergalactic parrot would appear to sing 'Hallelujah' while they floated in space.

Thus, bias is *needed*. This bias is an assumption of some higher level regularity [84] of our universe; a statement about what is more likely in our universe than others. A common example is Occam's razor [12]. This is a bias towards *simpler* solutions. The assumption here is that solutions that are more complex

[8] Assuming that the distribution over all tasks is uniform, or other particular distributions [31].

are less likely to be found in our universe, which can be shown using a minimum description length argument [33]. Arguments against the use of Occam's Razor in learning stem from the futility of assuming that a *complex* universe can be described using *simple* models [121, 25]. It has however been shown that Occam's Razor is built into the successful paradigm of Bayesian theory [77].

Another example is that intrinsic properties of an object do not change; if an object tends to fall when thrown up [9], it will tend to do so in the future, and not spawn intergalactic, singing birds.

Regularities can be on any set of objects

Such higher level regularities do not need to be on the grand scale of our universe, but could be about any set of objects. By making a choice of a model space, we have made assumptions about the type of learning problems that it can solve. This bias applies to all solutions available in the model space. To transfer between solutions then, we must find biases within a model or task space. This is goal of learning to transfer, to find a useful bias that can be assumed about a subset of tasks or models.

6.3 Notions of Transfer and Relatedness

So far, transfer was described in terms of subsets. For transfer, we decided on a single subset to which the objects of interest belonged. This says nothing about the remaining objects in the set of all objects. Consider the set of all animals. A catergorisation we have chosen is in terms of an animal's phylum, class, order, family, genus and species. For example, the red fox species is of the phylum vertebrate, class mammal, order carnivore, family Canidae (dog), and genus Vulpes (true fox). At each level, a particular animal be in multiple of that bias. That is, a red fox cannot be a mammal and a bird. This classification shows the regularity at each level. All animals in the class mammal, have the property of being a mammal, and therefore satisfy the axioms of being a mammal.

Transfer can occur at each level of a hierarchy

In such a catergorisation, there was a structure that was given to the entire set, since each level created a partition of the sets in question. At each level, the pertinent properties of animals in a class can be transferred; the properties of being a mammal, such as having mammary glands is true for *all* animals in the mammal class. In other words, each level contains an equivalence relation, where we can say that 2 animals are equivalent if they belong to the same subset. A partition is a global structure, and gives us a consistent sense in which to discuss transfer.

Consider the level of phylum; the catergorisation of animals into their respective phylums is carried out by considering consistently *certain* properties. Each phyla can be seen as a particular *body plan* [112]. Such consistency can be

[9] In this example, we consider the data points gathered by throwing objects up, and seeing them fall down; we do not assume any knowledge of a theory of gravity, or otherwise.

useful, and thus, we will only consider transfer in the context of such global notions in the rest of this thesis.

Set partitions allow for transfer

To summarise, we have decided that to transfer within a set of objects is to identify a partition of this set. Within each component of the partition, we can say that to transfer is to move between objects in a component. If the partitions were defined w.r.t. the constancy of some subset of all properties that can be applied to an object, then the constant properties of each component is precisely *what* is transferred. If instead, we started by proposing a partition of the set of objects, then, at least in the context of learning, there will be a notion of *invariant properties*; we make this precise in Section 7.1.

Let us make a notion of transfer precise.

Definition of a Notion of Transfer

Definition 6.3.1 (Notion of Transfer)**.** *Given a set of objects O, a partition $\mathcal{P}_O = \{U_i\}_{i\in\mathcal{I}}$ on O will be called a* notion of transfer*.*

We can make the following philosophical remarks about a notion of transfer, or a set of related objects.

Remark 6.3.1 A notion of transfer is set specific: We note that a notion of transfer is prescribed on a set of objects. This means that a notion of transfer is only meaningful in the context of the set of objects that is considered. Consequently, any notions of *what* is transferred, how *effective* the transfer is, among others is dependent on the set of objects too.

To see this, consider again Example 6.1.1. A possible set of motions of point masses is the collection of 2-dimensional motions that can be written as,

$$x(t) = \cos(\theta_0 + \omega t) + vt + x_0$$
$$y(t) = \sin(\theta_0 + \omega t) + y_0,$$

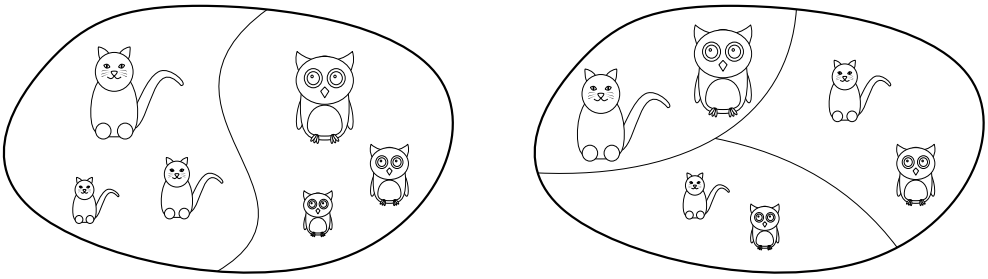
where $\theta_0 = \arctan(\frac{y_0}{x_0})$, for a constant $l \in \mathbb{R}^+$, $x_0^2 + y_0^2 = l^2$, and v and ω are fixed. Since each such motion can be identified by θ_0, this set is in fact isomorphic to $\mathbb{S}$. Then, the partition of this set, of which Example 6.1.1 is a notion of transfer can, for each $\theta \in [0, \pi/2)$, be defined by the components containing,

$$\{\theta, \theta + \pi/2, \theta + \pi, \theta + 3\pi/2\}.$$

In words, each component is defined by an initial angle from $[0, \pi/2)$, and contains the motions which start multiples of $\pi/2$ rad away.

Alternatively, the set of all motions that we consider could be the set of all possible motions of a point mass. It is clear that this set is much more complicated than the previous set, and any notions of what is transferred, and how effective transfer is follows suit. •

Remark 6.3.2 Notions of transfer aren't unique: It is possible for multiple valid notions of transfer to exist, since it is possible to make a variety of partitions of a set. This is shown in Figure 6.4. In Figure 6.4a, each subset has



(a) Partitioning the space of cats and owls w.r.t. their animal class (i.e. mammals or birds).

(b) Partitioning the space of cats and owls w.r.t. their size.

Figure 6.4. Different partitions (and therefore classifications) of our zoo of cats and owls.

the property that all animals in it are either cats or owls. Similarly, in Figure 6.4b, the size of each animal in a subset is the same. In some conditions, such as if the space of things has the structure of a manifold, the number of different partitions could be uncountable. •

Definition 6.3.2 (Transfer)**.** *Given a set of objects O that is equipped with a notion of transfer $\mathcal{P}_O$, we say that transfer occurs when we restrict movement between objects within any single $\mathcal{P} \in \mathcal{P}_O$.*

Definition of a Transfer

6.3.1 Relatedness

Transfer occurs between *related* objects. Then, given a notion of transfer, we can be precise about what a *set of related objects* is,

Definition 6.3.3 (Set of Related Objects)**.** *Given a set of objects O, and a notion of transfer $\mathcal{P}_O = \{U_i\}_{i\in\mathcal{I}}$, each component $(U_i \in \mathcal{P}_O) \subseteq O$ is defined as a set of related objects.*

Definition of a Set of Related Objects

Therefore, given objects $o_1, o_2 \in O$, they are *related* iff there exists a component $U \in \mathcal{P}_O$ such that $o_1, o_2 \in O$. It is known that given any set, there is a transformation group, called its permutation group, that acts transitively on it. This can give us a transformative notion of relatedness. That is,

Definition 6.3.4 (Notion of Relatedness)**.** *Given a set of objects O, a set of (pseudo) groups $\Pi_O = \{\mathcal{G}_i\}_{i\in\mathcal{I}}$ of transformations (or actions) which act on subsets of O is called a* notion of relatedness *if:*

Definition of a Notion of Relatedness

a) U_i, U_j are the domains of $\mathcal{G}_i, \mathcal{G}_j \in \Pi_O$, then $U_i \cap U_j = \varnothing$,

b) $\bigcup_{i\in\mathcal{I}} U_i = O$,

c) the set $\{\mathcal{O}, ...\}$, which contains all the orbits of all $\mathcal{G} \in \Pi_O$, is a partition of O. This is a notion of transfer.

Groups are defined in Appendix A.4. A pseudogroup is a generalisation of

a group. For local diffeomorphisms over a manifold, a pseudogroup can be defined as follows,

Definition of a Pseudogroups

Definition 6.3.5 (Pseudogroups)**.** *[61] If M is a smooth manifold, and $\mathcal{G}$ is a collection of local diffeomorphisms on open subsets of M into M, then $\mathcal{G}$ is a pseudogroup if:*

a) $\mathcal{G}$ is closed under restrictions: if $\tau : U \to M$, for U an open set of M, is in $\mathcal{G}$, then $\tau|_V \in \mathcal{G}$ for any $V \subseteq U$ is also in $\mathcal{G}$,

b) if $U \subseteq M$ is an open set, and $U = \bigcup_s U_s$, and $\tau : U \to M$, with $\tau|_{U_s} \in \mathcal{G}$, then $\tau \in \mathcal{G}$,

c) $\mathcal{G}$ is closed under composition: if $\tau_1, \tau_2 \in \mathcal{G}$, then $\tau_1 \circ \tau_2 \in \mathcal{G}$, whenever the composition is defined (i.e over intersections of the appropriate domain and codomain),

d) $\mathcal{G}$ contains an identity diffeomorphism over M,

e) $\mathcal{G}$ is closed under inverse: if $\tau \in \mathcal{G}$, then $\tau^{-1} \in \mathcal{G}$, with a domain $\tau(U)$, where U is the domain of τ.

Conditions (c)-(e) are similar what we expect from a group, whereas Conditions (a) and (b) allow the transformations to grow and shrink in expected ways.

Pseudogroups vs Groups

An important distinction between groups and pseudogroups is that the latter deals with *local* group-like structures. Therefore, the inclusion of pseudogroups in Definition 6.3.4 allows us to talk about group-like properties that act *locally* in an orbit. This is similar to how topological manifolds are *locally* Euclidean. We can then consider a much larger, and more interesting variety of partitions of sets of objects.

Given $\mathcal{G} \in \Pi_O$, a transformation $\pi_{\mathcal{G}} \in \mathcal{G}$ is a map,

$$\pi_{\mathcal{G}} : (U \subseteq O) \to (V \subseteq O).$$

The action of a transformation

The *action* of such a transformation is simply its application to an element of its domain. The domain of a $\mathcal{G} \in \Pi_O$ is the union of domains of each $\pi_{\mathcal{G}} \in \mathcal{G}$. The (pseudo)group nature of a notion of relatedness gives a (local) consistency of the transformations. It enforces that if $\pi = \pi_1 \circ \pi_2$, where $\pi, \pi_1, \pi_2 \in (\mathcal{G} \in \Pi_O)$, then for $x \in U$, which is the domain of $\mathcal{G}$,

$$\pi(x) = \pi_1(\pi_2(x)).$$

It also ensures the existence of inverse, and identity transformations. Thus if $\pi(x) = y$, then there must exists π^{-1}, where $\pi^{-1}(y) = x$. Such consistencies are clearly desireable.

Remark 6.3.3: An important note is that according to Definition 6.3.4, there doesn't need to only a single (pseudo)group that acts on the the set of objects. It is sufficient that we have a collection of (pseudo)groups that act locally, and are transitive on their respective orbits. •

A notion of relatedness is defined on a notion of transfer. A notion of relatedness is a stronger structure than a notion of transfer, since the former exactly specifies the latter. There can be different notions of relatedness that are defined on a notion of transfer; one such is derived from the permutation groups.

Correspondance between notions of transfer and relatedness

Let us say that,

Definition 6.3.6 (Transfer between objects)**.** *Given a notion of relatedness* Π_O*,* transfer *from* $o_1 \in U_i \subseteq O$ *to* $o_2 \in U_i \subseteq O$ *is the action of a transformation from* $\mathcal{G}_i \in \Pi_O$*.*

Definition of a Transfer between objects

Remark 6.3.4: In general, we do not want to make the notion of relatedness the collection of permutation groups that make us a partition. This is not particularly interesting. We mention it here for generality; in Chapter 7, we discuss a particular way of creating partitions of spaces of tasks, which give rise to interesting notions of relatedness. •

This idea of transfer works because by Definition 6.3.4, each set of related objects is an orbit of a particular group in the notion of relatedness. Thus, the transformations applied to each set of related objects do not move to a different set of related objects. If transfer is the movement within a set of related objects, then transfer between objects is the transformation of one object to a related object. Transfer occurs, and can occur between related objects; to say that objects are related is to say that transfer can occur between them.

Our notion of relatedness is similar to a definition given in [10, 9], where a single group of transformations act on $\mathfrak{t}$; our definition generalises this notion to include different groups of transformations that can act on subset of $\mathfrak{t}$, but produce the partition as needed.

Relation to previous work

Given a set of objects, and properties that can be used to characterise elements of this set, a notion of transfer gives us a way to state that a subset of the properties is of the same class; it defines an equivalence class. A notion of relatedness is a stronger structure that tells us of transformations that can be used to move between objects of the same equivalence class. Thus, transfer is the process of doing so.

A notion of transfer defines an equivalence class

6.4 Learning to Transfer

In the discussion so far, we referred to objects in general. Let us bring the focus back to transfer in the context of learning. We have stated that, in order to transfer, we must identify a partition of the set of objects; this gives us a formal notion of transfer. Therefore, to consider transfer between tasks, we must consider a partition on a space of tasks. The implied notion of relatedness then gives us transformations between the structures of tasks. Thus transfer between tasks occurs when we apply a transformation from a transformation (pseudo)group of this notion of transformations.

Intuitive description of learning to transfer

We have stated before that in learning, we want to find a representation of the space of learning tasks. This representation is given by a model space, and a learning algorithm. Under some notion of transfer, if we transfer between tasks, the interesting aspect to us is whether we can transfer between the descriptive models of each task. Thus, given a notion of transfer on the model space, we can intuitively think of *learning to transfer* as finding the set of related models that correspond to the set of related tasks. Then, we can *transfer* between models by learning (in the single task sense) within this restricted set; we call this *learning by transfer*. This correspondence between the transfer structure (notion of transfer) and the model and task space matches our expectations of learning being *structure preserving*.

Formally, we define,

Definition of a Transfer Task (Model) Space

Definition 6.4.1 (Transfer Task (Model) Space)**.** *A task (model) space* $\mathfrak{T}^{\mathfrak{X}}$ ($\Theta_\kappa{}^{\mathfrak{X}}$) *with a notion of transfer* $\mathcal{P}_{\mathfrak{T}}$ ($\mathcal{P}_{\Theta_\kappa}$) *in its structure is a Transfer Task (Model) Space.*

A transfer task or model space is a task or model space on which transfer can occur through the notion of transfer induced by their respective notions of transfer. When a transfer model space is chosen for a transfer task space, we will assume that there is a (pseudo) group homomorphism from $\Pi_{\mathfrak{T}} \to \Pi_{\Theta_\kappa}$.

We then define

Definition of a Transfer loss function

Definition 6.4.2 (Transfer loss function)**.** *A transfer loss function is a map,*

$$\mathcal{L}_{\mathfrak{X}} : \mathcal{P}_{\Theta_\kappa} \times \mathcal{P}_{\mathfrak{T}} \to \mathbb{R}$$

that can be written as,

$$\mathcal{L}_{\mathfrak{X}}(\mathcal{P}_{\Theta_\kappa}, \mathcal{P}_{\mathfrak{T}}) = f_{\mathcal{L}_{\mathfrak{X}}} \circ \mathcal{L}(\mathcal{P}_{\Theta_\kappa}, \mathcal{P}_{\mathfrak{T}}).$$

Here, $f_{\mathcal{L}_{\mathfrak{X}}}$ is some function that acts on the set of outcomes when the loss function of the space of learning tasks is applied over the entire space of $\mathcal{P}\Theta_\kappa \times \mathcal{P}_{\mathfrak{T}}$. An example of such a function could be,

$$\sum_{(\mathfrak{m},\mathfrak{t}) \in \mathcal{P}_{\Theta_\kappa} \times \mathcal{P}_{\mathfrak{T}}} \mathcal{L}(\mathfrak{m}, \mathfrak{t}). \tag{6.8}$$

Alternatively, it could be,

$$\min_{(\mathfrak{m},\mathfrak{t}) \in \mathcal{P}_{\Theta_\kappa} \times \mathcal{P}_{\mathfrak{T}}} \mathcal{L}(\mathfrak{m}, \mathfrak{t}).$$

$\mathcal{L}_{\mathfrak{X}}$ is defined on a space of subsets. As with learning in a single task setting, this represents a measure of how good a particular set of related models is for a given set of related tasks; it takes into account the loss of each task and model combination. Similarly, the task component will be replaced by samples of $\mathcal{P}_{\mathfrak{T}}$. How $\mathcal{P}_{\Theta_\kappa}$ will be practically dealt with will be described in Section 7.3. Then

Definition 6.4.3 (Learning to Transfer Algorithm)**.** *Given* $\mathfrak{T}^{\mathfrak{X}}$*,* ${\Theta_\kappa}^{\mathfrak{X}}$ *and* $\mathcal{L}_{\mathfrak{X}}$*, an algorithm that learns to transfer is a map,* Definition of a Learning to Transfer Algorithm

$$\mathfrak{X} : \mathcal{P}_{\mathfrak{T}} \to \mathcal{P}_{\Theta_\kappa},$$

where,

$$\mathfrak{X}(U_{\mathfrak{T}}) \in \underset{U_{\Theta_\kappa} \in \mathcal{P}_{\Theta_\kappa}}{\arg\min}\, \mathcal{L}_{\mathfrak{X}}(U_{\Theta_\kappa}, U_{\mathfrak{T}}).$$

Remark 6.4.1: Written in this way, the similarities to single task learning are immediate. Learning to transfer has the same form as vanilla learning; the main difference is that learning to transfer occurs on the level of sets of (related) tasks, rather than on individual tasks. •

Once $\mathfrak{X}(U_{\mathfrak{T}})$ has been found for a task $\mathfrak{t} \in U_{\mathfrak{T}}$ by *learning to transfer*, we can *learn by transfer* by carrying out a typical learning algorithm on $U_{\mathfrak{T}}$. That is,

Definition 6.4.4 (Learning by transfer)**.** *Given a learning task* $\mathfrak{t} \in U_{\mathfrak{T}}$*, with loss function* $\mathcal{L}$*, and for which we have learned to transfer, with* $\mathfrak{X}(U_{\mathfrak{T}}) = (U_{\Theta_\kappa} \in \mathcal{P}_{\Theta_\kappa})$*, learning by transfer can be carried out by an algorithm,* Definition of a Learning by transfer

$$\mathfrak{L}_{U_{\mathfrak{T}}} : U_{\mathfrak{T}} \to U_{\Theta_\kappa},$$

where,

$$\mathfrak{L}_{U_{\mathfrak{T}}}(\mathfrak{t}) \in \underset{\mathfrak{m} \in U_{\Theta_\kappa}}{\arg\min}\, \mathcal{L}_{\mathfrak{t}}(\mathfrak{m}).$$

Learning by transfer is to learn on a space that was selected such that transfer is efficient. By restricting ourselves to $\mathfrak{X}(U_{\mathfrak{T}})$, we are carrying out transfer, as per Definition 6.3.2. We see that learning to transfer gives us the ability to learn by transfer.

In Chapter 8, we will give examples of these techniques. We will see that differences in major methods, such as multi-task learning, transfer learning, and meta-learning derive from how these steps and components interact with each other, as well as how the data used for each step is generated.

Remark 6.4.2: Under this framework, it should be noted that, as a minimizer, the learning by transfer algorithm $\mathfrak{L}_{U_{\mathfrak{T}}}(\mathfrak{t})$ does not necessarily have to be $\arg\min_{\mathfrak{m} \in \Theta_\kappa} \mathcal{L}_{\mathfrak{t}}(\mathfrak{m})$, but merely must minimise in the set of related models.

In general, we would want U_{Θ_κ} and the learning by transfer algorithm to represent $U_{\mathfrak{T}}$; the latter must then be as well behaved as we expect a single task learning algorithm to be. Thus, it could be that for each $U_{\mathfrak{T}} \in \mathcal{P}_{\mathfrak{T}}$, a different learning by transfer algorithm needs to be devised. •

6.4.1 Equivariance of Transfer

Similar to continuous and smooth maps preserving the structure of topological spaces and smooth manifolds, the structure preserving property of a learning to transfer problem is equivariance. Equivariance is defined as follows;

Definition of a Equivariance

Definition 6.4.5 (Equivariance)**.** *Given a (pseudo)group $\mathcal{G}_M$ and an action θ_M on a space M, a (pseudo)group $\mathcal{G}_N$ and an action θ_N on a space N, and a (pseudo)group homomorphism $\rho : \mathcal{G}_M \to \mathcal{G}_N$, we say that a map $f : M \to N$ is equivariant if,*

$$f(\theta_M(g_m, m)) = \rho(g_m)(f(m)),$$

for any $m \in M$ and $g_m \in \mathcal{G}_M$.

Equivariance, intuitively

Equivariance [72] is a property which, not unlike invariance, can be placed on maps, and transformations that act on the domain and codomain. Here, instead of the output staying constant when the domain is transformed, we expect the output to also change in a similar[10] manner. In other words, an equivariant map commutes with a group action, up to the group homomorphism. This notion has been growing in interest in the ML community; for example, it is known that CNNs are translation equivariant, as visual problems follow similar properties [21, 19].

Using a learning to transfer algorithm $\mathfrak{X}$, and a learning by transfer algorithm $\mathfrak{L}_{U_{\mathfrak{T}}}$, we can construct a learning algorithm $\mathfrak{L}$ as,

$$\begin{aligned} \mathfrak{L} : \mathfrak{T} &\to \Theta_\kappa \\ \mathfrak{t} &\mapsto \mathfrak{L}(\mathfrak{t}) = \mathfrak{L}_{U_{\mathfrak{T}}}(\mathfrak{t}), \end{aligned}$$

where $\mathfrak{t} \in (U_{\mathfrak{T}} \in \mathcal{P}_{\mathfrak{T}}$. Further, recall that $\mathfrak{L}_{U_{\mathfrak{T}}} : U_{\mathfrak{T}} \to \mathfrak{X}(U_{\mathfrak{T}})$.

Then, equivariance means that for $\pi_{\mathfrak{T}} \in \Pi_{\mathfrak{T}}$, and the corresponding $\pi_{\Theta_\kappa} \in \Pi_{\Theta_\kappa}$,

$$\mathfrak{L} \circ \pi_{\mathfrak{T}}(\mathfrak{t}) = \pi_{\Theta_\kappa} \circ \mathfrak{L}(\mathfrak{t}).$$

In general, whether this condition is met depends on the constructed learning algorithm $\mathfrak{L}$, which in turn depends on $\mathfrak{X}$, all $\mathfrak{L}_{U_{\mathfrak{T}}}$, the transfer loss function, and the loss function (single task). However, if this map is equivariant, then together with the model space, this algorithm is a representation of the transfer task space.

For this reason, let us define a simpler, stricter version of learning to transfer, that combines these components.

Definition of a Learning to Transfer (Simple)

Definition 6.4.6 (Learning to Transfer (Simple))**.** *Suppose we are given a space of learning tasks $(\mathfrak{T}, \mathcal{L})$, where $\mathfrak{T}$ is equipped with a notion of transfer $\mathcal{P}_{\mathfrak{T}}$. We similarly select a model space Θ_κ that comes with its own $\mathcal{P}_{\Theta_\kappa}$, and an associated learning algorithm $\mathfrak{L}$ that are well-behaved.*

In addition, if $\mathfrak{L}$ is equivariant *w.r.t. $\mathcal{P}_{\mathfrak{T}}$ and $\mathcal{P}_{\Theta_\kappa}$,*

$$\begin{aligned} \mathfrak{X} : \mathcal{P}_{\mathfrak{T}} &\to \mathcal{P}_{\Theta_\kappa} \\ U_{\mathfrak{T}} &\mapsto \mathfrak{X}(U_{\mathfrak{T}}) = \mathfrak{L}(U_{\mathfrak{T}}), \end{aligned}$$

[10] Similar here implies either that the transformation set is the same, or that there is a structure preserving map to a transformation set that acts on the codomain.

is a learning to transfer algorithm.

Further, we can define for $U_{\mathfrak{T}} \in \mathcal{P}_{\mathfrak{T}}$*, and* $(U_{\Theta_\kappa} \in \mathcal{P}_{\Theta_\kappa}) = \mathfrak{X}(U_{\mathfrak{T}})$*,*

$$\begin{aligned} \mathfrak{L}_{U_{\mathfrak{T}}} : U_{\mathfrak{T}} &\to U_{\Theta_\kappa} \\ \mathfrak{t} &\mapsto \mathfrak{L}_{U_{\mathfrak{T}}}(\mathfrak{t}) = \mathfrak{L}|_{U_{\mathfrak{T}}}(\mathfrak{t}). \end{aligned}$$

The correctness of the derived learning to transfer and learning by transfer algorithms above is due to the well-behavedness of the model space and learning algorithm, as it means that $\mathfrak{L}$ is smooth and continuous.

We have a lot choice

As with learning single tasks, the structure on the task space is typically not concretely specified. Therefore, given a learning algorithm, model space, and loss function, we can choose this structure such that the requirements above are satisfied. If necessary, we can also choose local submanifolds of $\mathfrak{T}$ and Θ_κ that conform to our requirements.

This page was left intentionally blank.

Chapter 7

Foliations and Transfer

If learning to transfer requires us to partition our task space, then there must be mathematical way to express such a partition, along with a way to obtain the notion of relatedness from this partition. Recall that we will be assuming that the task space $\mathfrak{T}$ is a smooth manifold.

In order to partition a manifold, we can use a combination of 2 strategies:

Partitions can be using in 2 ways

a) we can tessellate $\mathfrak{T}$, where we create subsets (or submanifolds) that are of the same dimension as $\mathfrak{T}$.

b) we can create *parallel* submanifolds that are of dimensions lower than $\mathfrak{T}$.

Figure 7.1 graphically shows examples of these. It is possible to consider hierarchies that consist of combinations of these partitions. When combining these, it is not necessary that the dimensions of the submanifolds be equal to each other. We choose that each component of the partition is a submanifold

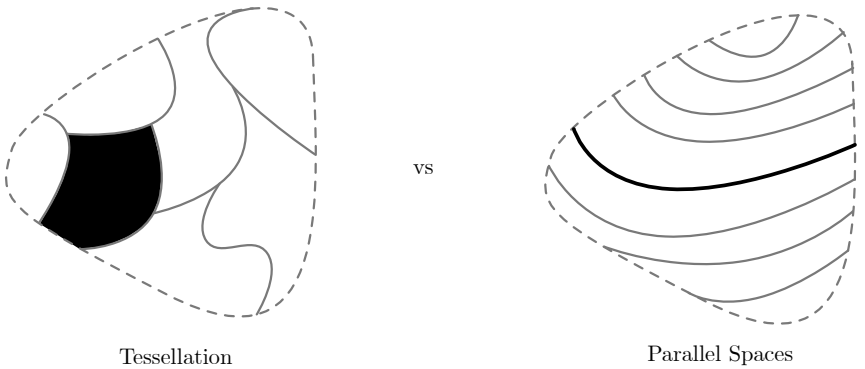


Figure 7.1. Comparison of tessellation and parallel spaces. The shaded regions denote the subsets we are considering in each case. A tessellation creates disjoint, smaller subsets that are of the same dimension as the original set. Parallel spaces on the other hand creates disjoint subsets that are of a lower dimension than the original set.

so that the continuity and smoothness benefits we obtain by thinking about

a space of tasks as a manifold apply to each component. A mathematical framework that provides a systematic and parametric way of partitioning manifolds into submanifolds is the theory of foliations.

A brief history of foliations.

The study of foliations stems from a need to qualitatively understand non-linear first order partial differential equations. Initial works were introduced by Lagrange, and then generalised by Pfaff [37]. These gave examples of simplifying such problems without providing the conditions under which such simplifications were possible. A major theorem that links to regular foliations is the Frobenius' Theorem, which gives conditions under which a smooth distribution (a smooth sub-bundle of the tangent bundle) on a smooth manifold has smooth integral manifolds. The collection of such integral manifolds is a foliation. These conditions can be used to determine whether some overdetermined systems of partial differential equations have solutions [58].

Foliations have found use in the geometric study of control systems. Hermann [39] initially described the connection between the accessibility problem in control theory and foliations. The accessibility problem asks which points can be reached by following the flow of a system of differential equations, given an initial point. From a control point of view, this corresponds to attainable states of control system. Such contexts involved the use of singular foliations; Stefan [98] and Sussman [102] independently extended Frobenius' Theorem to this case, under mild assumptions.

Summary of chapter

In this chapter, we provide some definitions of foliations, as well as examples. The key theorem (Theorem 7.2.1) is used to show how a foliation can be used to produce a notion of relatedness. Section 7.3 gives a brief discussion about the structure of the space of leaves of a foliation (the quotient space of equivalence classes). We end by discussing how the structure of a foliation can be used to learn to transfer.

7.1 Foliations

A foliation is a geometric structure that can be placed on a smooth manifold[1]. A foliation can allow us to mathematically write partitions such as those we saw in Figure 7.1 on a smooth manifold.

A foliation is a restriction on the atlas that is given to a smooth manifold.

Definition of a Regular Foliation

Definition 7.1.1 (Regular Foliation)**.** *If $\mathcal{M}$ is a smooth manifold of dimension d with an atlas $\mathcal{A}_{\mathfrak{m}}$, a $(k < d)$ -dimensional foliation $\mathcal{F}^k$ of $\mathcal{M}$ is a covering subset of its atlas, that satisfies:*

a) if $(U, \phi) \in \mathcal{F}$, then $\phi : U \to (U_1 \subseteq \mathbb{R}^{d-k}) \times (U_2 \subseteq \mathbb{R}^k)$, and

[1] Smoothness is not a necessary condition to introduce a foliation on a manifold, but we consider it for our present purposes.

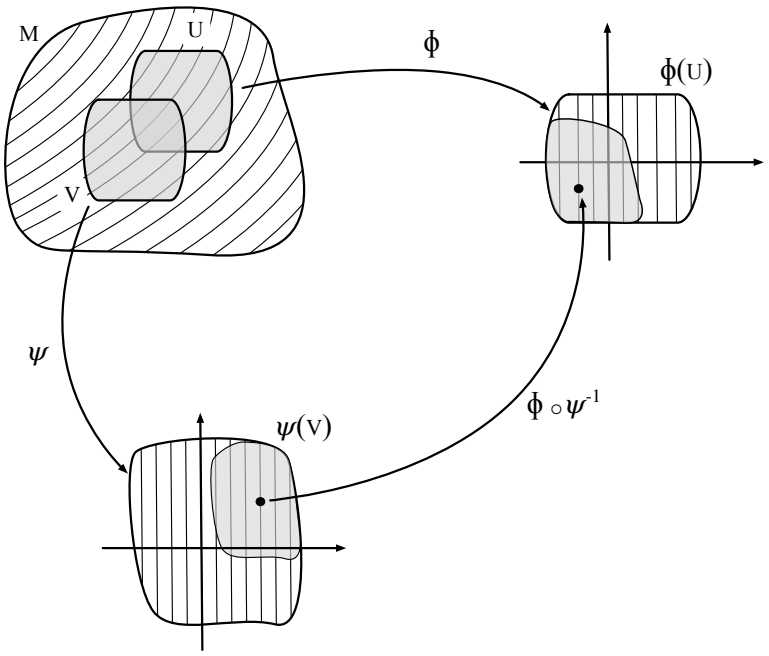


Figure 7.2. A 1-dimensional regular foliation on $\mathbb{R}^2$.

b) if $(U,\phi),(V,\psi) \in \mathcal{F}$*, where* $U \cap V \neq \varnothing$*, any chart transition function* $\psi \circ \phi^{-1} : \phi(U \cap V) \to \psi(U \cap V)$ *has the form of* $\psi \circ \phi^{-1}(x,y) = (h_1(x), h_2(x,y))$*, where* $x \in U_1$*,* $y \in U_2$*, and*

$$h_1 : U_1 \to (V_1 \subseteq \mathbb{R}^{d-k}),$$
$$h_2 : U_1 \times U_2 \to (V_2 \subseteq \mathbb{R}^k).$$

Foliations create partitions

Intuitively, this definition describes the decomposition of a manifold into non-overlapping connected submanifolds that are of dimension k [13]. Each such submanifold is called a *leaf*. Locally, a foliation creates a rectification of the coordinates. This is implied by Condition (b), whereby during a chart transition, the first $d-k$ coordinates of the output depend only on the first $d-k$ coordinates of the input; the remaining k coordinates depend on the entire input coordinate. Thus the first $d-k$ coordinates of a point on the manifold are *consistent* up to chart transitions.

A precise definition of a leaf can be made using*plaques*;

Definition of a Plaque

Definition 7.1.2 (Plaque)**.** *Given a foliation chart* (U,ϕ) *of a foliation* $\mathcal{F}^k$ *on manifold* $\mathcal{M}$*, a plaque* α_q^U *is a subset of* U *obtained by,*

$$\phi^{-1}(\{q \in (U_1 \subseteq \mathbb{R}^{d-k})\} \times (U_2 \subseteq \mathbb{R}^k)).$$

A plaque is component of a local partition of the open set U. It is obtained by keeping the $d-k$ coordinates *fixed*. Plaques can be said to be consistent if

their first $d-k$ coordinates are equivalent. That is, for $(U_1, \phi_1), (U_2, \phi_2) \in \mathcal{F}^k$, where $U_1 \cap U_2 \neq \varnothing$, plaques $\alpha_q^{U_1} \subset U_1$ and $\alpha_s^{U_2} \subset U_2$ are consistent iff $s = h_1(q)$. h_1 is as in Definition 7.1.1. Then, given a particular plaque, a leaf is the union of all plaques consistent with it; this is an equivalence relation. Thus,

Definition of a Leaf

Definition 7.1.3 (Leaf)**.** *Given foliation $\mathcal{F}^k$ of manifold $\mathcal{M}$. A leaf $L \subset \mathcal{M}$ is a collection of points, where for any $p, q \in L$, there exists a sequence of consistent plaques $(\alpha_{s_i}^{U_i})_{i=0}^{I}$, such that,*

a) $p \in \alpha_{s_0}^{U_0}$, and $q \in \alpha_{s_I}^{U_I}$, and,

b) $U_i \cap U_{i-1} \neq \varnothing$.

Alternative definition of a regular foliation

Alternative definitions of a regular foliation can grant us some more insight about them. See [69] for proofs of their equivalences; we will not cover those here for brevity.

There is a close relationship between smooth vector fields and foliations. This is because an equivalent definition of a foliation is;

Definition of a Regular Foliation (Distribution)

Definition 7.1.4 (Regular Foliation (Distribution))**.** *Given a manifold $\mathcal{M}$, a distribution $\mathcal{D}$ on $\mathcal{M}$ is a smooth subbundle of the tangent bundle on $\mathcal{M}$. A distribution is* integrable *if sections of $\mathcal{D}$ are closed under the Lie bracket*[2]*.*

A $k-$dimensional foliation on $\mathcal{M}$ is an integrable distribution $\mathcal{D}$ on $\mathcal{M}$ of rank k.

A smooth section of a tangent bundle is a smooth vector field on a manifold. A $k-$dimensional subbundle can be specified by a $k-$dimensional vector subspace of the tangent space at each point $m \in \mathcal{M}$; the smoothness requirement asks that as we move along the manifold, this vector space also moves smoothly. Thus, smooth subbundles allow us to define smooth vector fields on $\mathcal{M}$. The integrability of a distribution means that the vector subspaces are in fact the tangent spaces of a submanifold of $\mathcal{M}$; the dimension of this submanifold is clearly equal to the dimension of the vector subspace; k. This submanifold is called the integral manifold. The integral manifolds of a distribution are precisely the *leaves* of a foliation.

Regular foliations can also be defined by submersions;

Definition of a Regular Foliation (Submersions) [69]

Definition 7.1.5 (Regular Foliation (Submersions) [69])**.** *Consider an open cover $\{U_i\}$ on a smooth $\mathcal{M}$ of dimension d. On each U_i, define a submersion $s_i : U_i \to \mathbb{R}^q$ such that there exists a necessarily unique diffeomorphism,*

$$\gamma_{ij} : s_j(U_i \cap U_j) \to s_i(U_i \cap U_j),$$

with,

$$\gamma_{ij} \circ s_j|_{U_i \cap U_j} = s_i | U_i \cap U_j.$$

[2]This is a result of Frobenius Theorem; see Theorem 19.12 of [58]

Furthermore, these diffeomorphisms satisfy the Haeflinger Cocycle [34]

$$\gamma_{ij} \circ \gamma_{jk} = \gamma_{ik}.$$

The collection of submersions is a foliation of dimension $d - q$.

The submersions here define the projection maps of points onto the leaves that they belong to. The diffeomorphisms are the maps h_1 in Definition 7.1.1.

A key feature of a regular foliation is that the dimension of the leaves is constant. In this case, the leaves create a partition of the manifold that looks similar to how we described parallel spaces. However, a generalisation of this exists, called a singular foliation [102, 98], which allows us to write tessellations, and combinations of the two styles of partitioning in the same language.

Definition of a Singular Foliation [98]

Definition 7.1.6 (Singular Foliation [98])**.** *If M is a smooth manifold of dimension d with an atlas $\mathcal{A}_{\mathfrak{m}}$, a singular foliation is a partition of M into immersed, connected submanifolds of non-constant dimensions. There exists an atlas of* distinguished *charts, where for each $x \in M$ there exists a chart ϕ such that:*

a) if $x \in U$, then $\phi : U \to (U_1 \subseteq \mathbb{R}^{d-n}) \times (U_2 \subseteq \mathbb{R}^n)$, where n is the dimension of the leaf containing x, and U_1 and U_2 contain 0,

b) $\phi(x) = (0, 0)$, and,

c) if L is a leaf, then $L \cap \phi^{-1}(U_1 \times U_2) = \phi^{-1}(l \times U_2)$, where $l = \{w \in U_1 : \phi^{-1}(w, 0) \in L\}$.

It should be noted that foliations, in general, can have complicated topologies and properties on their leaves, and spaces of leaves [80, 13, 55].

Foliations, particular singular foliations can be generated in many ways. For example, this is of great importance to the field of geometric control theory, where singular foliations are used to describe solvable families of control strategies [47]. Several results in the study of singular foliations [53] were derived for this; it has been shown in an Orbit Theorem [47], that orbits of families of vector fields, under certain conditions generate singular foliations. Since we can use such families of vector fields to generate (local) diffeomorphisms, they can give us a notion of relatedness. More simply however, it is known that the locally free action of a Lie group produces a regular foliation [13, 54]. Let us look at some examples of foliations.

Example 7.1.1: Any submersion $s : \mathcal{M} \to \mathcal{N}$ on a manifold $\mathcal{M}$ can produce a $d - n$ foliation [13, 69], where n is the dimension of $\mathcal{N}$. The leaves are the connected components of the fibres of s.

Take the sinusoidal architecture from Example 3.1.2. We showed in Section 3.2 that the model space for this architecture is $\mathbb{S} \times \mathbb{R}$. The submersion $\mathbb{S} \times \mathbb{R} \to \mathbb{R}$ generates a foliation where each leaf is $\mathbb{S}$, and is given by $\mathbb{S} \times \{x \in \mathbb{R}\}$. ▲

Example 7.1.2: Perhaps the simplest example of a foliation is the 1-dimensional foliation on $\mathbb{R}^2$ given by the submersion,

$$s_{\mathbb{R}^2} : \mathbb{R}^2 \to \mathbb{R};$$
$$x \mapsto s_{\mathbb{R}^2}(x) = \mathrm{pr}_1(x).$$

Here, there is only one open set, the entirety of $\mathbb{R}^2$. The leaves are simply the straight lines spanned by the second coordinate, at each value of the first coordinate. This is a *trivial* foliation. ▲

Example 7.1.3: A singular foliation of $\mathbb{R}$ is the partition $\{(-\infty, 0), \{0\}, (0, \infty)\}$. ▲

Example 7.1.4: A singular foliation on $\mathbb{R}^2$ can be defined by the circles at center x_0, of any radius $r \geq 0$. Each circle is then a leaf. When $r > 0$, the leaves are 1-dimensional. When $r = 0$, the leaf is the 0-dimensional point x_0. ▲

7.2 Relatedness from Foliations

Suppose we are given a task space $\mathfrak{T}$, which we write as a finite dimensional smooth manifold. We can then define a notion of transfer on this by defining a foliated structure $\mathcal{F}^k$, regular or otherwise, on $\mathfrak{T}$. The following theorem shows that, given any foliation, we can construct a set of pseudogroups of transformations on $\mathfrak{t}$ that can behave as a notion of relatedness.

Relatedness from $\mathcal{F}^k$

Theorem 7.2.1. *Given a leaf L of a foliation $\mathcal{F}^k$ on a manifold $\mathcal{M}$, there exists a (pseudo)group of transformations that act transitively on L. Then the collection of such (pseudo)groups over all leaves of $\mathcal{F}^k$ is a notion of relatedness.*

Proof. We know that a foliation creates a partition of the manifold. If we can show that each leaf is the orbit of its own pseudogroup, the conditions of Definition 6.3.4 will be satisfied.

Recall that a leaf L is an immersed, connected, smooth submanifold of a manifold on which $\mathcal{F}^k$ is defined [58, 98].Further, assume that L has a dimension of d.

Transformations defined by a global flow in $\mathbb{R}^d$

Let us first look at the Euclidean space $\mathbb{R}^d$. Let us equip $\mathbb{R}^d$ with the standard frame. The standard frame $V = (V_1, ..., V_d)$ is a collection of vector fields V_i, where the j-th component of the vector $V_i(x)$ is given by $V_i^j(x) = \delta_i^j$ for $x \in \mathbb{R}^d$. Note that since V is a global frame, we have that $(V_1(x), ..., V_d(x))$ is a basis for the tangent space at $x \in \mathbb{R}^d$.

We know that V is a global frame, and that each V_i is a complete vector field. Thus the global flow $\upsilon_i : \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}^d$ exists. Then, $\upsilon_i(t) : \mathbb{R}^d \to \mathbb{R}^d$, for some

$t \in \mathbb{R}$ is a diffeomorphism. We define the following map,

$$\begin{aligned} \tau_\Delta : \mathbb{R}^d &\to \mathbb{R}^d \\ x &\mapsto \tau_\Delta(x) = \upsilon_d(\mathrm{pr}_d(\Delta)) \circ ... \circ \upsilon_1(\mathrm{pr}_1(\Delta))(x). \end{aligned} \tag{7.1}$$

where $\Delta \in \mathbb{R}^d$. This map moves a point $x \in \mathbb{R}^d$ to $x + (\mathrm{pr}_1(\Delta), ..., \mathrm{pr}_d(\Delta))$. For all $\Delta \in \mathbb{R}^d$, the transformations τ_Δ form an additive group. Its inverse is given by,

$$\begin{aligned} \tau_\Delta^{-1} : \mathbb{R}^d &\to \mathbb{R}^d \\ x &\mapsto \tau_\Delta^{-1}(x) = \upsilon_d(-\mathrm{pr}_d(\Delta)) \circ ... \circ \upsilon_1(-\mathrm{pr}_1(\Delta))(x) \end{aligned}$$

Thus

$$\tau_\Delta^{-1}(x) = \tau_{-\Delta}(x)$$

Furthermore, this group is transitive in $\mathbb{R}^d$, since we can find a transformation from any point $x \in \mathbb{R}^d$ to any other point $y \in \mathbb{R}^d$. Suppose $x = (x_1, ..., x_d)$ and $y = (y_1, ..., y_d)$. The following is true:

$$y = \upsilon_n(y_n - x_n) \circ ... \circ \upsilon_1(y_1 - x_1)(x) = \tau_{x-y}(x);$$

let us denote this as a transformation τ_{xy}. Note that there is an inverse $\tau_{xy}^{-1} = \tau_{yx} = \upsilon_n(x_n - y_n) \circ ... \circ \upsilon_1(x_1 - y_1)$, where,

$$x = \upsilon_n(x_n - y_n) \circ ... \circ \upsilon_1(x_1 - y_1)(y).$$

Thus, given the coordinates of any two points, we can construct a diffeomorphism between them using the standard frame, written in terms of compositions of flows of vector fields.

Foliated Atlas of Coordinate Balls

Next, we identify some properties of the topology on L. Assume that L has a subset topology derived from the foliation [13]. From [58, Theorem 1.10], we know that L has a countable topological basis $\mathfrak{B}$ of coordinate balls. Take the domains of the charts of $\mathcal{F}^k$ restricted to L (denoted by $\mathcal{F}^k|_L$); these are open subsets of L, and form an open cover. We create a refinement of this cover, where for each domain $U \in \mathcal{F}^k|_L$, we find $B_U \subseteq \mathfrak{B}$ where $\bigcup_{\mathcal{B}_U^k \in B_U} \mathcal{B}_U^k = U$. By restricting the chart maps to the appropriate $\mathcal{B}^k$, we have an atlas on L made up of coordinate balls. Denote this as $\overline{\mathcal{A}}_{\mathfrak{B}} = \{(\mathcal{B}, \overline{\phi})\}$.

Notation. In the notation $\overline{\mathcal{A}}\mathfrak{B} = \{(\mathcal{B}, \overline{\phi})\}$, we have suppressed any mention of the original foliated atlas, since it suffices to work with $\mathcal{A}_{\mathfrak{B}}$ on the leaf. It is assumed that $\overline{\phi}$ was constructed from the appropriate chart map in the original foliated atlas.

Further, if we write $X \in \mathcal{A}$, where X is a set, and $\mathcal{A}$ is an atlas, we mean the domain of a chart in $(X, \phi) \in \mathcal{A}$. Thus, $\mathcal{B} \in \overline{\mathcal{A}}_{\mathfrak{B}}$ is the domain of $(\mathcal{B}, \overline{\phi}) \in \overline{\mathcal{A}}_{\mathfrak{B}}$.

By definition, we know that any coordinate ball $\mathcal{B}$ is homeomorphic to an open ball $\mathcal{B}^r_{\mathbb{R}^d}$ in $\mathbb{R}^d$. By ϕ, let us denote homeomorphism such that the center of $\mathcal{B}$ maps to 0 in the coordinate space. For any $(\mathcal{B}, \overline{\phi}) \in \mathcal{A}_{\mathfrak{B}}$, suppose that the center of $\mathcal{B}$ is denoted is denoted by $c_{\mathcal{B}}$. Then, let ϕ be defined by,

$$\begin{aligned} \phi : \mathcal{B} &\to \mathcal{B}^r_{\mathbb{R}^d} \\ x &\mapsto \phi(x) = \overline{\phi}(x) - \overline{\phi}(c_{\mathcal{B}}). \end{aligned}$$

Notation. $\mathcal{B}^r_{\mathbb{R}^d}$ is an open ball in $\mathbb{R}^d$ of radius $r > 0$. This does not refer to the previous notation $\mathcal{B}^k_U$.

Thus, let us now work with the atlas $\mathcal{A}_{\mathfrak{B}} = \{(\mathcal{B}, \phi)\}$, where the coordinates of every coordinate ball is centered at $0 \in \mathbb{R}^d$.

As a final remark, we also know that L is connected. Thus, given any two points on $p, q \in L$, we can find a sequence $\mathcal{B}_1, ..., \mathcal{B}_N \in \mathfrak{B}$ such that $p \in \mathcal{B}_1$, $q \in \mathcal{B}_N$, and $\mathcal{B}_n \cap \mathcal{B}_{n-1} \neq \varnothing$ for $N \geq n \geq 2$. One can think of this as a sequence of connected coordinate balls that connect p and q.

Transformations on coordinate balls

Consider the chart $(\mathcal{B}, \phi) \in \mathcal{A}_{\mathfrak{B}}$. Each standard ball $\phi(\mathcal{B})$ is itself homeomorphic to $\mathbb{R}^d$ via a map such as,

$$h(x_i) = \frac{x_i}{r - ||x||} = y_i, \tag{7.2}$$

for any $x = (x_1, ..., x_i, ..., x_d) \in \mathcal{B}^r_{\mathbb{R}^d}$ and $y = (y_1, ..., y_i, ..., y_d) \in \mathbb{R}^d$.

Note that h exists for each of the coordinates. The inverse of this map is given by,

$$h^{-1}(y_i) = \frac{r y_i}{1 + ||y||} = x_i.$$

Note that these maps are smooth. Then, we can construct a diffeomorphism, for $\Delta \in \mathbb{R}^d$ as,

$$\begin{aligned} \pi_\Delta : \mathcal{B} &\to \mathcal{B} \\ p &\mapsto \pi^{\mathcal{B}}_{(\Delta}(p) = \phi^{-1} \circ h^{-1} \circ \tau_\Delta \circ h \circ \phi(p). \end{aligned} \tag{7.3}$$

The inverse of this map is,

$$\pi^{-1}_\Delta(p) = \phi^{-1} \circ h^{-1} \circ \tau^{-1}_\Delta \circ h \circ \phi(p). \tag{7.4}$$

Then, for any $p, q \in \mathcal{B}$, we can construct a map π_{pq},

$$\pi_{pq}(p) = \phi^{-1} \circ h^{-1} \circ \tau_{xy} \circ h \circ \phi(p) = q,$$

where $x = h \circ \phi(p)$ and $y = h \circ \phi(q)$. The inverse $\pi^{-1}_{pq} = \pi_{qp}$ of this map from $q \in \mathcal{B}^k$ to $p \in \mathcal{B}$ is given by,

$$\pi_{qp}(q) = \phi^{-1} \circ h^{-1} \circ \tau_{yx} \circ h \circ \phi(q) = p.$$

For all $\Delta \in \mathbb{R}^d$, the set of all transformations π_Δ is then a group of diffeomorphisms that acts transitively on $\mathcal{B}$. Let us denote this group as $\Pi^{\mathcal{B}}$. We have effectively used the homeomorphism defined in Equation (7.2) to move the transformations on $\mathbb{R}^d$ given in Equation (7.1) to transformation in $\mathcal{B}$.

Pseudogroup of transformations

An open subset of $\mathcal{B}$ is denoted as $U^{\mathcal{B}}$. We will also denote a finite sequence of elements in $\mathbb{R}^d$ as $\mathcal{S}_{\mathbb{R}^d} = \{x^1, ..., x^N\}$, and a finite sequence of coordinate balls $\mathcal{S}_{U^{\mathcal{B}},\mathfrak{B}} = \{U^{\mathcal{B}}, \mathcal{B}_2, ...\mathcal{B}_N\}$, where $\mathcal{B}_j \in \mathcal{A}_{\mathfrak{B}}$. Further, $\mathcal{B} \cap \mathcal{B}_2 \neq \varnothing$, and $\mathcal{B}_{j-1} \cap \mathcal{B}_j \neq \varnothing$ for $j \geq 3$. Finally, let us denote by $\pi^{\mathcal{B}}_{(x_1,...,x_d)}$ a map of the form Equation (7.3).

We define a set of transformations Π on the leaf L, where each $\pi \in \Pi$ is a map,

$$\pi^{\mathcal{S}_{U^i,\mathfrak{B}}}_{\mathcal{S}_{\mathbb{R}^d}} : U^{\mathcal{B}} \to \mathcal{B}_N$$
$$p \mapsto \pi^{\mathcal{S}_{U^{\mathcal{B}},\mathfrak{B}}}_{\mathcal{S}_{\mathbb{R}^d}}(p) = \pi^{\mathcal{B}_N}_{x^N} \circ ... \circ \pi^{\mathcal{B}_2}_{x^2} \circ \pi^{U_{\mathcal{B}}}_{x^1}(p),$$

where x^j is such that for $p \in \mathcal{B}_{j-1} \cap \mathcal{B}_j$, $\pi^{\mathcal{B}_j}_{x^j}(p) \in \mathcal{B}_j \cap \mathcal{B}_{j+1}$. Π contains all transformations acting on the leaf L that can be written as finite compositions of maps from any $\Pi^{\mathcal{B}}$, as long as the domains and codomains of each transformation make sense.

We now show that Π satisfies each condition in Definition 6.3.5.

a) Satisfied by definition, since $\mathrm{dom}(\pi^i) = \mathcal{B}_q$, $\pi^i|_U$ for $U \subset \mathcal{B}$ exists.

b) First, let us denote by $\pi_1, \pi_2 \in \Pi$ transformations, where $\mathrm{dom}(\pi_1) = U_1$ and $\mathrm{dom}(\pi_2) = U_2$. Further, suppose we want a transformation map π, with domain $\mathrm{dom}(\pi) = U_1 \cup U_2$. and $\pi|_{U_i} = \pi_i$. Now for such a map to exist, it must be the case that there exists $\mathcal{B} \in \mathcal{A}_{\mathcal{B}}$, where $U_1, U_2 \subseteq \mathcal{B}$. This is because, in Equation (7.3), the homeomorphisms h are defined for a particular $\mathcal{B}$. Thus, to talk about restrictions, we can only consider domains in a particular $\mathcal{B}$.

   Now, consider $U = \bigcup_{i\in\mathcal{I}} U_i$. Further, $\mathcal{S}_{U,\mathfrak{B}}$ and $\mathcal{S}_{U_i,\mathfrak{B}}$, differ only in the first element, and of course $U_i \subseteq U$. Now if $\pi^{\mathcal{S}_{U,\mathfrak{B}}}_{\mathcal{S}_{\mathbb{R}^d}}$ is a transformation such that $\pi^{\mathcal{S}_{U,\mathfrak{B}}}_{\mathcal{S}_{\mathbb{R}^d}}|_{U_i} = \pi^{\mathcal{S}_{U_i,\mathfrak{B}}}_{\mathcal{S}^i_{\mathbb{R}^d}}$, for any $i \in \mathcal{I}$, it must be that $\mathcal{S}_{\mathbb{R}^d} = \mathcal{S}^i_{\mathbb{R}^d}$. Otherwise, the restriction would not work. Therefore, if $\pi^{\mathcal{S}_{U_i,\mathfrak{B}}}_{\mathcal{S}^i_{\mathbb{R}^d}} \in \Pi$ for any $i \in \mathcal{I}$, $\pi^{\mathcal{S}_{U,\mathfrak{B}}}_{\mathcal{S}_{\mathbb{R}^d}} \in \Pi$.

c) Since all compositions are included, this follows from definition.

d) By setting $x^i = 0$ for all x^i in $\mathcal{S}_{\mathbb{R}^d}$, we obtain the identity maps.

e) Follows from Equation (7.4).

Transitivity

Finally, we must show that Π is transitive. We do this by showing constructively that between any two points on L, we can find a map between them in Π.

Recall that due to the connectedness of L, we can find a sequence $\{\mathcal{B}_n\}_{pq}$ of coordinate balls that connect two points $p, q \in L$. Take the intersection $\mathcal{B}^n \cap \mathcal{B}^{n-1}$. We will denote by $p_{i,i-1}$ to be any point in this intersection. Then we can define a transformation π_{pq} between points $p, q \in L$ as,

$$\pi_{p,\ldots,p_{i,i-1},\ldots,q}(p) = \pi_{p_{n,n-1},q} \circ \ldots \circ \pi_{p,p_{2,1}}(p) = q.$$

Its inverse is given by,

$$\pi_{q,\ldots,p_{i,i-1},\ldots,p}(q) = \pi_{p_{2,1},p} \circ \ldots \circ \pi_{q,p_{n,n-1}}(q) = p.$$

We know that some foliations are naturally defined by the actions of Lie groups. In this theorem, we showed that *all* foliations create a notion of relatedness, at least in terms of *pseudogroups*. ■

The converse is also possible, where we define a notion of relatedness, and derive a foliation from it, as long as the orbits of the groups in the notion of relatedness are immersed, connected submanifolds. For example, the orbits of a free action of a Lie group forms a foliation [13, 69].

Correspondence between foliations and notion of relatedness

Thus, using foliations, we can go either way. If a notion of relatedness that is thought to be relevant to a particular problem of learning to transfer is known, we can apply it to generate a structure on a task space that reflects the induced notion of transfer. Alternatively, perhaps, a foliation can be learned, or simply defined, from which a notion of relatedness can be derived, for purposes of interpretability and understanding.

7.3 The leaf space and learning to transfer

In Section 6.4.1, we could reconcile the structure on the task space to suit our needs. Let us then look at the model space. Recall that we assume that the model space is a smooth d-dimensional manifold. We give this space an arbitrary regular[3] k-foliation. Since a foliation is specific atlas, we know that locally (in an open set), the manifold looks Euclidean, and the foliation creates a rectification.

Global and task specific parameters

This rectification can be interpreted as a separation of the parameters of a model space into global and task specific parameters. The global parameters are locally equal for all tasks in a set of related tasks (in this case, the leaves), and the task specific parameters identify the solution to a task within this set. The global parameters, over the entire model space are equal, up to chart transitions.

For a $U \in \Theta_\kappa$, where U is a subset of a chart domain, Definition 7.1.1 told us that $\phi(U) = (U_1, U_2)$, where $U_1 \subseteq \mathbb{R}^{d-k}$, and $U_2 \subseteq \mathbb{R}^k$. $\phi^{-1}(\{a\} \times U_2)$, for $a \in U_1$ is a plaque. If each such plaque is a component of a unique leaf a, then

[3]We will assume that foliations we use are regular henceforth, since they can be difficult to deal with without the additional complexity of a singular foliation.

perhaps we can consider moving through U_1 as moving through leaves, locally. Thus, in this case, we could consider using algorithms such a gradient descent for learning to transfer algorithms.

Nasty topologies of space of leaves

However, the space of leaves is the quotient space we obtain by identifying all points that belong to a leaf, and giving it the quotient topology. Generally, the structure of the space of leaves can be complicated and nasty, and might not even carry a manifold structure. Let us focus on regular foliations. An example where the space of leaves is complicated is the images of all curves on a torus of the form

$$\gamma_b(t) = (e^{it}, e^{i(at+b)}),$$

for $b \in \mathbb{R}$. If a is irrational, each leaf is dense in the torus because the curve will never close. In the Reeb Foliation [79, 13] on $\mathbb{S}^3$, the space of leaves is non-Hausdorff [13].

Classifying the topology of the space of leaves

There is some hope however. It is possible to classify the topology of the space of leaves. Suppose we have a regular foliated manifold $(\mathcal{M}^d, \mathcal{F}^k)$. A submanifold $\mathcal{N}$ of $\mathcal{M}$ is called a transverse section if, for every leaf it intersects, the tangent space of $\mathcal{N}$ is complementary to the tangent space of the leaf. Thus, at the points of intersection, $\dim(\mathcal{N}) + k = d$. A simple example of a transverse submanifold can be constructed as the following. Take a chart $(U, \phi) \in \mathcal{F}^k$. Then, we know that $\phi^{-1}(\{a\} \times U_2)$ (as above) is a plaque. Then, for $b \in U_2$, $\phi^{-1}(U_1 \times \{b\})$ is a transverse section.

Let us denote by $\mathcal{N} \cap L$ the set of all intersections with a leaf L. According to [13, Theorem 4], there are three possible topological outcomes for this set:

a) $\mathcal{N} \cap L$ is discrete, where each element of $\mathcal{N} \cap L$ is an isolated point,

b) all elements of $\mathcal{N} \cap L$ is in the non-empty interior of the closure $\overline{\mathcal{N} \cap L}$,

c) neither of the above; thus the closure $\overline{(\mathcal{N} \cap L)}$ has an empty interior, and $\mathcal{N} \cap L$ is not discrete.

Case (b) corresponds to a leaf that is locally dense in the manifold, such as in the example of the torus above, when a is irrational. Case (c) is a leaf that is nowhere dense, but its topology differs from the subset topology [85][4]. Such a leaf is called *exceptional*. In contrast, under Case (a), the leaf is described as *proper*, and is nowhere dense and carries the subset topology.

In the remainder of this thesis, we present examples of learning to transfer methods from the literature. For this, it suffices that we assume that we learn in the domain of a single chart, and that in this chart, each plaque corresponds to a unique leaf. This makes the formulation much simpler than if we had considered a nice, yet more general foliation. Studying the conditions under which such a foliation allows for the use of algorithms such as gradient descent is left as future work.

[4] Examples of foliations with exceptional leaves can be found in [85, 106].

This page was left intentionally blank.

Chapter 8

Examples

Having defined learning to transfer, and that we can represent it using foliations, we now show how examples of popular methods from the literature that carry out transfer can be expressed in the language of our framework.

Useful assumptions

We recall the major assumptions we had made. We diminished the complexity of the manifolds that were the model and task spaces by restricting them to open neighbourhoods. Any chosen foliation on the model space is assumed to contain a chart that is a superset of this open set, and that the plaques correspond to unique leaves; the structure of the task space is assumed to have been derived from the chosen loss functions and learning algorithms, such that they are well behaved.

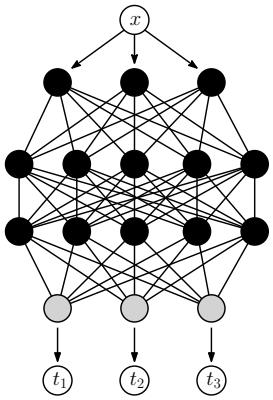
Overview of proofs

The general strategy we use in this chapter is to first show that the learning technique used implicitly defines a foliation on the model space. Since we assume to be working on a single chart, we simply show that the learning technique breaks the parameter space of the model into global and task specific parameters. In addition, we do not need to consider symmetries in the parameter space, particularly those that lead to connected subsets of symmetric parameters (such as the non-negative homogeneity of ReLU). This is because in the presented work use gradient descent; it is expected that gradient descent would account for these, since the gradient would be constant along symmetric points.

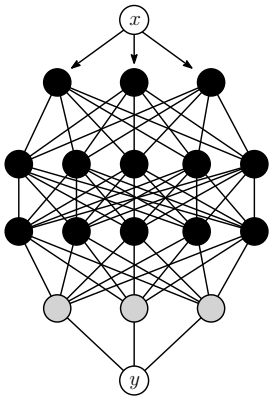
We then show that the learning algorithm is equivariant w.r.t. this implicit foliation, and the induced structure on the task space. In fact, this equivariance follows trivially due to our assumption that the structure on the task space is a representation. Thus, these algorithms carry out learning to transfer as per Definition 6.4.6.

As future work, we propose to look at devising learning to transfer techniques that do not simply make assumptions about the task space in the ways mentioned above. Could we exploit some known structure, such as knowledge of how related tasks transform?

The techniques we discuss below is not an exhaustive list; we believe that it is possible to cast techniques in continual learning [23], domain adaptation



(a) Hard parameter sharing in multitask learning using a neural network. The black nodes represent nodes that are shared between all tasks, whereas the gray nodes are task specific.



(b) Weight sharing in inductive transfer learning. The black nodes represent nodes that are fixed during learning the new task, whereas the gray nodes are updated.

Figure 8.1. Example network architectures for multitask learning and inductive transfer learning.

[118, 73], self-supervised learning [117], variational autoencoders [49], and others in this framework. This will be left for future work.

8.1 Multitask Learning

Multitask learning (MTL)[14] is a learning paradigm where several *related* tasks are solved in parallel to learn a *shared representation*. In the context of the thesis by Caruana [14], tasks are related if learning them together provides an improvement to their generalisation performance. MTL is described as an approach to inductive bias training.

A key feature that enables MTL to lead to improved generalisation is the parameter sharing; Figure 8.1a gives an example of a network architecture that allows for parameter sharing. Here, suppose that in an open set $U \subseteq \Theta = \mathbb{R}^d$, where Θ is the parameter space of an architecture κ. d is the number of parameters. It is clear that U decomposes as $U_1 \times U_2$, where each $U_i \subseteq \mathbb{R}^{k_i}$ and $k_1 + k_2 = d$.

Hard parameter sharing in MTL

Let us assume that the task space is equal to U; each $\{a\} \times U_2$ is a set of related tasks. Then, given an appropriate loss function $\mathcal{L}$, a gradient descent learning algorithm $\mathfrak{L}_{\text{MTL}}$ would proceed as follows. Given a collection $\{\mathfrak{t}_i\}_{i=1}^m$ of related tasks, and a collection $\{\theta_i\}_{i=1}^m$, where we assume that the first k_1 components are equal, MTL computes a MTL loss as,

$$\mathcal{L}_{\text{MTL}}(\{\theta_i\}_{i=1}^m, \{\mathfrak{t}_i\}_{i=1}^m) = \sum_{i=1}^m \mathcal{L}(\theta_i, \mathfrak{t}_i).$$

Note how this is similar to example transfer loss we gave in Example 6.8. For $j = 1$ to $j = k_1$, the flow that gradient descent follows is given by $\upsilon^j_{\partial_j \mathcal{L}_{\text{MTL}}}$, where $\partial_j \mathcal{L}_{\text{MTL}}$ is the partial derivative of the negative loss w.r.t. the $j-$th coordinate. Assuming initial parameter values $\{\theta_i^0\}_{i=1}^m$ with such properties, the $j-$th component of $\mathfrak{L}_{\text{MTL}}(\mathfrak{t}_i)$ is computed as,

$$\text{pr}_j \circ \mathfrak{L}_{\text{MTL}}(\mathfrak{t}_i) = \lim_{t\to\infty} \upsilon^j_{\partial_j \mathcal{L}_{\text{MTL}}}(\text{pr}_j(\theta_i^0), t),$$

for $j = 1$ to $j = k_1$. Note that for this range $\text{pr}_j(\theta_i^0)$ are equal over all i. Furthermore, $\partial_j \mathcal{L}_{\text{MTL}}$ are equal too, due to the sum in $\mathcal{L}_{\text{MTL}}$. Thus, these components of the output models will be equal to each other.

Notation. pr_1 denotes the projection map on to the first coordinate. More generally, pr_i denotes the projection to the $i-$th coordinate.

For the remaining parameters, $j = k_1 + 1$ to $j = d$, the update rules are,

$$\text{pr}_j \circ \mathfrak{L}_{\text{MTL}}(\mathfrak{t}_i) = \lim_{t\to\infty} \upsilon^j_{\partial_j \mathcal{L}_{\mathfrak{t}_i}}(\text{pr}_j(\theta_i^0), t).$$

Note that the partial derivatives used are task specific, since they are the partial derivatives of $\mathcal{L}_{\mathfrak{t}_i}$. Due to this, these components of the parameter are allowed to freely change.

Trivial foliation of MTL model

This shows the hard parameter sharing of MTL; the first k_1 components of the parameter vector are constrained to change together. The decomposition of U into $U_1 \times U_2$ is exactly the decomposition in Condition (a) of Definition 7.1.1. We do not need to worry about the second condition, since we do not carry out chart transitions in this setting. Thus, trivially, the MTL technique had proposed a foliation.

An implicit assumption of MTL is that $\{\mathfrak{t}_i\}_{i=1}^m$ is a collection of related tasks. Due to the parameter sharing, the $\{\mathfrak{L}_{\text{MTL}}(task_i)\}_{i=1}^m$ also belongs to a leaf of the foliation. If we assume that $\mathfrak{T}$ has the same structure as Θ, then the equivariance of the induced learning algorithm follows. This is because the learning algorithm is now an exact representation. Thus, each task has a unique model, and the assumed correspondence between the notions of relatedness in the task and model spaces implies the necessary result trivially.

8.2 Transfer Learning

These types of models are commonly seen in the transfer learning setting [74], and in some meta learning models, such as [86, 76, 36, 48]. These models simply take the direct definition of the local coordinates of a foliation, and use that as the definition of the model. Figure 8.1b shows an example neural network topology of such a model.

In inductive transfer learning, a common approach is to pretrain a model on a large dataset to obtain optimal parameters. Suppose these are θ. Then,

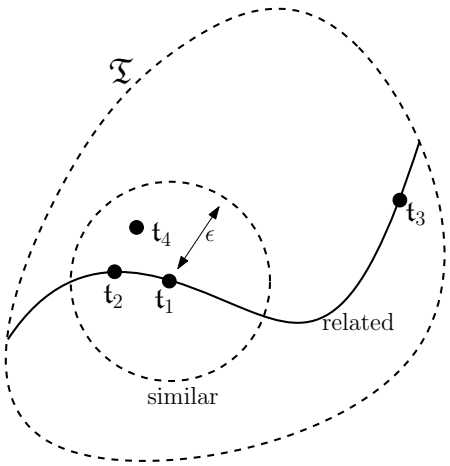


Figure 8.2. The distinction between similarity and relatedness. Similarity is defined in terms of an ϵ-ball around a particular element. Here, points $\mathfrak{t}_2$ and $\mathfrak{t}_3$ are similar to $\mathfrak{t}_1$. On the other hand, relatedness is defined in terms of a transformation set; this relationship is shown as a line joining them. Thus, tasks $\mathfrak{t}_1, \mathfrak{t}_2$ and $\mathfrak{t}_3$ are related to each other. However, $\mathfrak{t}_4$ is not related to $\mathfrak{t}_2$ and $\mathfrak{t}_1$ though they are similar; $\mathfrak{t}_3$ is not similar to $\mathfrak{t}_2$ and $\mathfrak{t}_1$, though they are related.

given a new dataset, the model is rectified into $(\theta_{fixed}, \theta_{retrain})$; the $\theta_{retrain}$ components are retrained while the θ_{fixed} components are kept fixed. We see immediately that this is simply describing the local coordinates of a foliation.

That is, in a local open set $U \subset \mathfrak{m}$, we have a chart ϕ such that $\phi(m) = (\theta_{fixed}, \theta_{retrain})$, where $\theta_{fixed} \in \mathbb{R}^a$, $\theta_{retrain} \in \mathbb{R}^b$ and $a + b = n$, the total number of parameters. Each θ_{fixed} defines a local plaque of dimension b. During transfer, since θ_{fixed} remains constant, the retraining occurs on this plaque.

In these methods, we learn the set of related models by simply solving the single task problem, and then taking the leaf on which it lies in the foliation of the model space. As with MTL, the equivariance follows trivially by the assumed structure of the task space.

The key difference between transfer learning and MTL is that MTL explicitly defines a transfer loss, whereas transfer learning simply learns over a single task. An alternative view of transfer learning is to see it as learning a large, single model that can act as an initialisation, where the solutions to new tasks are *close* to this model. This is another way of producing a notion of transfer.

Definition of a A Set of Similar Tasks

Definition 8.2.1 (A Set of Similar Tasks)**.** *A metric ρ on $\mathfrak{T}$ is called a notion of similarity. $\mathfrak{t}_1, \mathfrak{t}_2 \in \mathfrak{T}$ are said to be similar if $\rho(\mathfrak{t}_1, \mathfrak{t}_2) < \epsilon$ for $\epsilon > 0$. Then, given $\mathfrak{t} \in \mathfrak{T}$, the set $\{\mathfrak{s} \in \mathfrak{T} | \rho(\mathfrak{s}, \mathfrak{t}) < (\epsilon > 0)\}$ is a set of tasks similar to $\mathfrak{t}$.*

If we choose a countable number of reference tasks on $\mathfrak{T}$, then a notion of similarity can induce a partition on $\mathfrak{T}$ in the way of a Voronoi diagram [78].

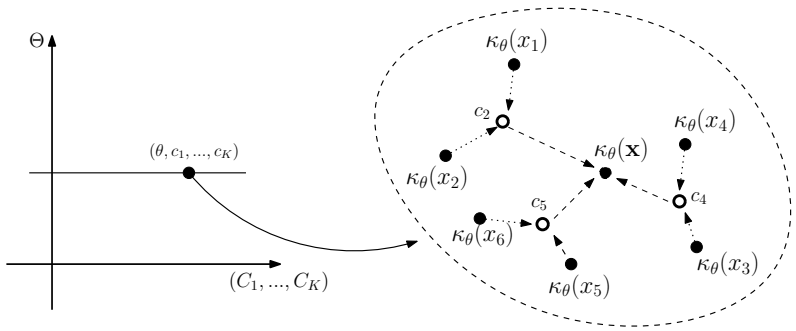


Figure 8.3. A foliation in the space $(\theta, c_1, ..., c_k, ..., c_K)$ gives us what we expect from a prototypical network. For a given dataset $\mathfrak{D}$, we find θ during the initial training phase. For a given task, which has classes from $\{2, 4, 5\} \subset \{1, ..., K\}$, the values (c_2, c_4, c_5) define the distribution over the class probabilities of a particular input x. These values are calculated from a small task-specific dataset $\mathfrak{D}_t = \{(x_1, 2), (x_2, 2), (x_3, 4), (x_4, 4), (x_5, 5), (x_6, 5)\}$, in this example.

That is, we can write an equivalence relation, given a set of reference tasks $R_{\mathfrak{T}} = \mathfrak{t}_i \in \mathfrak{T}_{i \in \mathcal{I}}$, $\mathfrak{s}_1 \sim \mathfrak{s}_2 \iff \arg\min_{\mathfrak{t}_i \in R_{\mathfrak{T}}} \rho(\mathfrak{t}_i, \mathfrak{s}_1) = \arg\min_{\mathfrak{t}_i \in R_{\mathfrak{T}}} \rho(\mathfrak{t}_i, \mathfrak{s}_1)$. Such a partition would look like the tesselation in Figure 7.1. Since this is a partition, there is a notion of relatedness that can be associated with it (for example, by trivially considering the permutation groups on each element of the partition).

Transfer via similarity occurs when we move to a point that is close to a reference starting point, whereas transfer by relatedness moves along a subset of all possible directions of variation. The differences between these methods of transfer is visualised in Figure 8.2

Transfer learning could be considered to transfer according to both methods.

8.3 Meta Learning

8.3.1 Prototypical Networks: Simple Meta-Learning

Prototypical Networks are used for few-shot classification [97]. It learns an embedding function which computes a distribution over the different classes of the problem. The embedding function is used to compute a prototype vector in the embedding space for each class. A distribution over the classes for a new query point is then computed by running a softmax over the distance from the prototypes to the embedding of the new data point.

This algorithm then carries out few-shot learning by generating the prototypes by applying the embedding function on the dataset given for the new task, and immediately computing the predictions on queries. During training, each iteration involves choosing a subset of the classes and their corresponding data, and optimising the embedding function to minimise the negative log likelihood of the true predictions.

To be more precise, suppose we are given a dataset $\mathfrak{D} = \{(x_i, y_i)\}_{i\in\mathcal{I}}$, where $x_i \in X$ and $y_i \in \{1, ..., K\}$ is a class identity. A classification task $\mathfrak{t}$ is given by choosing $n_\mathfrak{t} \leq K$ classes from $\{1, ..., K\}$, denoted as $Y_\mathfrak{t}$, to generate a dataset $\mathfrak{D}_\mathfrak{t} = \bigcup_{k\in Y_\mathfrak{t}}\{(x_i, y_i) : (x_i, y_i) \in \mathfrak{D}, y_i \in Y_t\}$.

The embedding function is a map $\kappa_{\theta\mathfrak{D}} : X \to \mathbb{R}^m$, where $\theta \in \Theta$ are the parameters of this function. For the k-th class in Y_t, the prototype is given by,

$$c_k = \frac{1}{\mid S_k \mid} \sum_{(x_i,y_i)\in S_k} \kappa_\theta(x)(x_i). \tag{8.1}$$

S_k is the set of data points given for the class $k \in Y_t$. The probability of a particular class, given a distance metric ρ on the embedding space, is:

$$p(y = k \mid x) = \frac{\exp(-\rho(\kappa_\theta(x), c_k))}{\sum_{k'\in\mathcal{Y}_t} \exp(-\rho(\kappa_\theta(x), c_{k'}))}. \tag{8.2}$$

In this model, the full parameters that specify a prediction are $(\theta, c_1, ..., c_K)$. θ corresponds to the global bias that specifies how each input should be transformed; this is independent of the task that is currently being solved. The coordinates given by the remaining $(c_1, ..., c_k, ..., c_K)$ specify a distribution over the particular classes; they are the task specific parameters. Each c_k therefore corresponds to a particular class.

Since for a given task, $n_t \leq K$, we would be ignoring the c_k coordinates that correspond to classes in the set $\{1, ..., K\}\backslash Y_t$. This can be done by setting c_k to very high or low values, since then, their contribution to the sum in the denominator of 8.2 would be negligible.

It should be noted that here, the adaption to the new task is deterministic and immediate; there is no *training* (optimisation) required. This approach is illustrated in Figure 8.3.

The foliation in this setting is given by $(\theta, c_1, ..., c_K)$, where θ identifies the leaves, and $c_1, ..., c_K$ corresponds to a task. Since K is finite, the number of different combinations of tasks is also finite. However, we assume that the leaves are a manifold. The values of $c_1, ..., c_K$ that correspond to remaining possible problems can be thought of as continuous, smooth variations of each of the classes. These might not be identifiable to humans as a particular type of object (such as a cat or a dog).

The equivariance follows once again by assuming the structure on the task space to be that on the model parameter space.

8.3.2 Model Agnostic Meta Learning (MAML)

MAML [29] is a meta-learning technique that has been very popular since it was derived a few years ago; it is a good example of a method that makes use of 2 levels of optimisation. For MAML, we are given a set of tasks. Its goal is to find a common initialisation point, from where following gradient descent (or

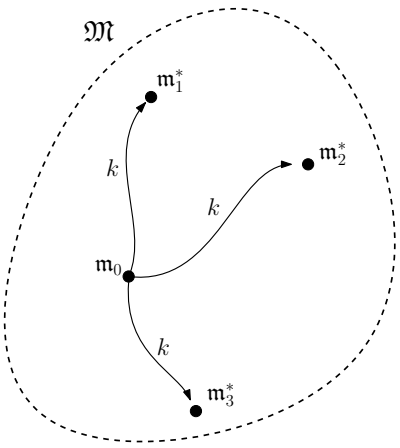


Figure 8.4. MAML finds an optimal initial model m_0 such that solutions to other tasks $\mathfrak{m}_1^*$, $\mathfrak{m}_2^*$ and $\mathfrak{m}_3^*$ are k gradient descent steps away.

another gradient based optimisation method) for a fixed number of iterations will yield a suitable model for each task. MAML does this by optimising the initial point in a higher level loop, and following gradient descent for a fixed number of steps starting from this point; see Figure 8.4.

Showing that this type of model uses the structures we have discussed is difficult. There are also several ways by which we could attempt to show that MAML implicitly uses a foliation. One such method could be to define the parameter space of MAML as $\Theta \times \Theta$. This is similar to the graphical model used by [32], where it is posed as a hierarchical bayes model. Alternatively, we could also view it in terms of transfer due to similarity. A more complicated view could be to make the following conjecture,

Conjecture 8.3.1. *We are given a task space $\mathfrak{T}$, a model space Θ_κ and a loss function $\mathcal{L}$.*

Let us assume that the gradient descent algorithm, that for any given initial model $\mathfrak{m}_0 \in \Theta_\kappa$, follows the flow $\upsilon_{d\mathcal{L}}(\mathfrak{m}_0, \mathfrak{t})$ of the negative loss differential $d\mathcal{L}$ till $t = k$ is well-behaved, for $\mathfrak{t} \in \mathfrak{T}$. For $\mathfrak{m}_0$, construct the subset $U_\mathfrak{t}$ of tasks for which,

$$|\mathcal{L}(\mathfrak{m}_\mathfrak{t}(k), \mathfrak{t}) - \mathcal{L}(\mathfrak{m}_\mathfrak{t}^*, \mathfrak{t})| = \epsilon,$$

where $\mathfrak{m}_\mathfrak{t}(k)$ is the model we obtain by following the flow for k time, $\mathfrak{m}_\mathfrak{t}^*$ is the fixed point of the dynamical system given by the gradient flow of $d\mathcal{L}$, and $\mathfrak{t} \in U_\mathfrak{t}$. Then, we conjecture that $U_\mathfrak{t}$ is a leaf of a foliation of at least $d-1$ dimension, where d is the dimension of $\mathfrak{T}$. Different leaves can be obtained by changing the number of time steps, and keeping ϵ fixed.

The collection of models $\mathfrak{m}_\mathfrak{t}(k)$ for each $U_\mathfrak{t}$ then is a leaf on the model space.

The proof of this very general statement is beyond the scope of the present thesis. We have made the accuracy fixed; in practice, we are okay with models

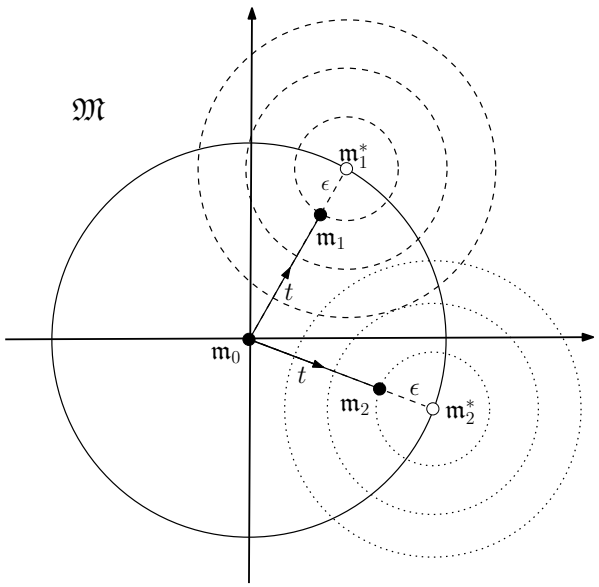


Figure 8.5. A pictorial representation of Theorem 8.3.2. The arrows show the flows, starting at m_0 along the positive t direction for each task. Centered at each $m_{\mathfrak{t}_i}^*$, we have drawn some of the level sets of the quadratic loss function, shown as dashed and dotted circles. The leaf generated by MAML is the solid circle.

that are *at worst* of a chosen accuracy; in the view of the foliations, we are saying that several, neighbouring leaves are good enough for our solution. The same can be said about stating that we want to reach a sufficiently suitable solution in at most k steps. It is possible that this requirement ends up creating a partition of a tesselation; that is we have found a region of similarity.

To study this, we would need to understand the system of differential equations given by,

$$\dot{\mathfrak{m}}_i = -\partial_i \mathcal{L}_{\mathfrak{t}}(m),$$

which is complicated in the general sense. We leave this study for future work.

First simple setting

However, in order to present an intuition as to why we think the the above would be correct, we present a very simplified example that illustrates Conjecture 8.3.1. Let us assume that the model space is locally 2-dimensional; that is $\phi(U_{\Theta_\kappa}) \subset \mathbb{R}^2$, for some chart on the $\mathfrak{m}$. Write the same for $U_{\mathfrak{t}}$. Further, make the very restrictive assumption that the loss function $\mathcal{L} : U_{\Theta_\kappa} \times U_{\mathfrak{T}} \to \mathbb{R}^+$ is homogeneously quadratic in both variables. That is, we can write the loss as,

$$\mathcal{L}((\mathfrak{m}_1, \mathfrak{m}_2), (\mathfrak{t}_1, \mathfrak{t}_2)) = \sum_{i=1}^{2} (\mathfrak{t}_i - \mathfrak{m}_i)^2 \tag{8.3}$$

Here, we see that changing the task merely changes the position of $\mathfrak{m}_{\mathfrak{t}}^*$, but keeps the shape of loss curve over $\mathfrak{m}$ fixed. The level sets of this system create circles centered at t; this is shown in Figure 8.5. The loss value at the optimal

model for a given model is 0. For such as system, we can show the following theorem:

Theorem 8.3.2. *For a given $\mathfrak{m}_0$, and a coordinate system centered at $\mathfrak{m}_0$, the subsets of $\mathfrak{T}$ for which the accuracy of the model is a fixed ϵ is given by,* MAML Foliation (simple)

$$\sum_{i=1}^{2} \mathfrak{t}_i^2 = \frac{\epsilon}{e^{-4k}}. \tag{8.4}$$

Proof. Since the loss function is given by $\mathcal{L}((\mathfrak{m}_1, \mathfrak{m}_2), (\mathfrak{t}_1, \mathfrak{t}_2)) = \sum_{i=1}^{2} (\mathfrak{t}_i - \mathfrak{m}_i)^2$, assuming that the coordinate system is centered at $\mathfrak{m}_0$, the vector field on $\mathfrak{m}$ is given by,

$$\mathrm{M}((\mathfrak{m}_1, \mathfrak{m}_2)) = (\dot{\mathfrak{m}_1}, \dot{\mathfrak{m}_2}) = 2\mathfrak{m}_1 \frac{\partial}{\partial \mathfrak{m}_1} + 2\mathfrak{m}_2 \frac{\partial}{\partial \mathfrak{m}_2}. \tag{8.5}$$

This is a system of differential equations which can be solved to give us a flow $\theta(k) = (\mathfrak{m}_1(k), \mathfrak{m}_2(k))$; the initial value condition is $\mathfrak{m}_i(0) = 0$, since we have centered the coordinate system at $\mathfrak{m}_0 = (0, 0)$.

The flow is given by,

$$\mathfrak{m}_i(k) = \mathfrak{t}_i e^{-2tk} + \mathfrak{t}_i. \tag{8.6}$$

Since $\mathcal{L}(\mathfrak{m}_\mathfrak{t}^*, \mathfrak{t}) = 0$, we have that for the $\mathfrak{m}(k)$ at which $\left| \mathcal{L}(\mathfrak{m}(k), \mathfrak{t}) - \mathcal{L}(\mathfrak{m}_\mathfrak{t}^*, \mathfrak{t}) \right| = \epsilon$, $\mathcal{L}(\mathfrak{m}(k), \mathfrak{t}) = \epsilon$. Thus,

$$\sum_{i=1}^{2} ((\mathfrak{t}_i e^{-2tk} + \mathfrak{t}_i) - \mathfrak{t}_i)^2 = \epsilon. \tag{8.7}$$

This can be rearranged to produce,

$$\sum_{i=1}^{2} \mathfrak{t}_i^2 = \frac{\epsilon}{e^{-4k}}. \tag{8.8}$$

■

Here, $(\mathfrak{t}_1, \mathfrak{t}_2)$, are the coordinates of each of the dimensions of a point $\mathfrak{t} \in \mathfrak{T}$. k denotes a variable that controls the radius; this k corresponds to time that one must follow the flow of a task starting at m_0 to end up at the model that is of ϵ accuracy. The following corollary can then be shown:

Corollary 8.3.3. *Given a task $t = (\mathfrak{t}_1, \mathfrak{t}_2) \in \mathfrak{t}$, a chosen accuracy $\epsilon \in [0, \infty]$, and an initial model m_0, the time k needed to get to a model $\mathfrak{m}_\mathfrak{t}$ for which the accuracy $\left| \mathcal{L}(t, \mathfrak{m}_\mathfrak{t}) - \mathcal{L}(\mathfrak{m}_\mathfrak{t}^*, \mathfrak{t}) \right| = \epsilon$ is given by,*

$$k_\epsilon = 4 \ln \left(\frac{\epsilon}{\mathfrak{t}_1^2 + \mathfrak{t}_2^2} \right), \tag{8.9}$$

and the model coordinates $\mathfrak{m}_\mathfrak{t} = (\mathfrak{m}_1, \mathfrak{m}_2)$ are given by,

$$\mathfrak{m}_i = -\mathfrak{t}_i \left(\frac{\epsilon}{\mathfrak{t}_1^2 + \mathfrak{t}_2^2} \right) + \mathfrak{t}_i. \tag{8.10}$$

Proof. Equation (8.7) can be solved for a k, which gives us,

$$k_\epsilon = 4 \ln \left(\frac{\epsilon}{\mathfrak{t}_1^2 + \mathfrak{t}_2^2} \right). \tag{8.11}$$

This can then be plugged into Equation (8.6) to give us:

$$\mathfrak{m}_i = -\mathfrak{t}_i \left(\frac{\epsilon}{\mathfrak{t}_1^2 + \mathfrak{t}_2^2} \right) + \mathfrak{t}_i. \tag{8.12}$$

■

These results show that the subsets that satisfy the criterion for a set of related tasks under the MAML scheme are *circles* in $\mathbb{R}^2$, for the given problem specification; this is a regular foliation on $\mathbb{R}^2/0$, or a singular foliation on $\mathbb{R}$. These results can be extended to higher dimensions, and to loss functions that aren't symmetric; for example, if the level sets look like ellipsoids, then we can expect that leaves would be ellipsoids too.

8.4 Reinforcement Learning

This section summaries some results from [76]. This paper studies a multi-task reinforcement learning (MTRL) setting, where we want transfer to occur in a specific way. The space of tasks consisted of reinforcement learning (RL) tasks, where the variation could occur in either the transition dynamics, or the reward function. A particular set of related tasks that we looked at were variations of the balancing cartpole.

The cartpole[1] system is a pendulum attached to a cart that can be moved along the x-axis (see Figure 8.7). To move the cart, a fixed force, either positive or negative is applied to the cart, causing the cart to accelerate, as well as swing the pendulum. In the particular variation of the problem we looked at, the cartpole begins with the pendulum in an upright position in the middle of the x-axis (see Figure 8.7), and the RL agent must continue to balance the pole for a period of time at a fixed point.

The variations to the transition dynamics we obtained by varying the mass of the cart. As can be imagined, a higher mass is harder to move, making a balanced position more stable, but also making a failed attempt harder to correct. The reward function was varied by changing the fixed point around which to balance the pole. Recall that the initial point is fixed, which means that the agent must move the balance point to the necessary area using its actions. The reward function counted the number of time steps for which the tip of the pole was within a small bubble around the balance point; in other words, this was a sparse reward.

[1] https://gym.openai.com/envs/CartPole-v1/

The set of all variations above created the set of related tasks for our setting. The particular space of tasks is assumed to be the tasks for which the chosen policy architecture could be a solution for. Instead of approaching this problem as a typical multi-task problem, where there is a single vector of parameters that are the task specific parameters, we used several task specific parameters that correspond to each direction of variations.

The goal here was to identify if exploiting known structure about how the tasks are distributed can lead to interpretable, and fast learning, and transfer.

8.4.1 Practical Implementation

Instead of approaching this from a manifold perspective, we opted to use the variational inference view of reinforcement learning [60]. We start by introducing a random variable R that denotes whether the trajectory[2] $\tau := (i, j, s_0, a_0, z_0, g_0, \dots s_T, a_T, z_T, g_T)$ is optimal ($R = 1$) or not ($R = 0$). T is the length of the episode; we assume this to infinite. The likelihood of an optimal trajectory is defined as $p(R = 1|\tau) \propto \exp\left(\sum_t^T r_j(a_t, s_t)\right)$. The trajectory assuming optimality is denoted by $p(\tau|R = 1)$. If τ is assumed to be a latent variable with prior probability $p(\tau)$, we specify the log-evidence as $\log p(R = 1) = \log \int p(R = 1|\tau)p(\tau)d\tau$. Introducing a variational distribution on trajectories $q(\tau)$, which combined with Jensen's inequality provides the Evidence Lower Bound (ELBO) $\mathbb{E}_{\tau\sim q(\tau)}\left[\log p(R = 1|\tau)p(\tau) - \log q(\tau)\right]$.

The generative model of a trajectory is given by Figure 8.6. The generative model is,

$$p(\tau) = p(i)p(j)p(s_0)\prod_{t=0}^{T-1} p(z_t|i)p(g_t|j)p(a_t|s_t, z_t, g_t)\mathcal{T}_i(s_{t+1}|s_t, a_t),$$

and the variational distribution is,

$$q(\tau) = p(i)p(j)p(s_0)\prod_{t=0}^{T-1} q_\delta(z_t|i)q_\omega(g_t|j)\pi(a_t|s_t, z_t, g_t)\mathcal{T}_i(s_{t+1}|s_t, a_t).$$

The conditional distributions $q_\delta(z_t|i)$ and $q_\omega(g_t|j)$ imply that these functions vary only if the transition dynamics, or reward function changes. Thus, we have 3 z_i and g_j each. These distributions were chosen to Gaussian, with diagonal covariance; thus the mean and covariance are the parameters ω and δ. π is the policy parameterised by θ. This was chosen to be a neural network. It is also conditioned on the latent variables z and g. Instead of these being fixed, if we were in a deterministic setting, they are instead sampled from a distribution. From a foliation point of view, the parameters of this setup would be (θ, ω, δ), where θ are the coordinates of the leaf.

The objective function is the maximization of the ELBO w.r.t. π, q_δ and q_ω,

[2] To keep with the technique of [60], we have omitted the rewards in this trajectory.

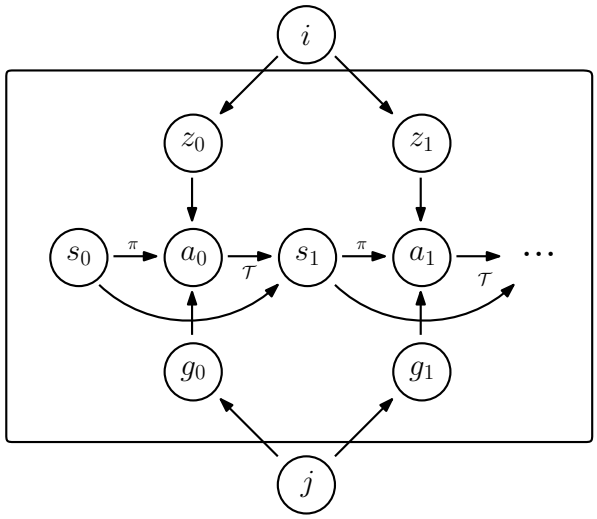


Figure 8.6. Generative model of trajectories.

which is given by,

$$J_{t_{ij}}(\theta,\omega,\delta) = \mathop{\mathbb{E}}_{q(\tau)}\Big[\sum_{t=0}^{\infty}\gamma^t\Big(r_j(s_t,a_t) - \frac{1}{\alpha_d}\log\frac{q_\delta(z_t|i)}{p(z_t|i)} - \frac{1}{\alpha_r}\log\frac{q_\omega(g_t|j)}{p(g_t|j)} - \frac{1}{\alpha_\pi}\log\frac{\pi_\theta(a_t|s_t,z_t,g_t)}{p(a_t|s_t,z_t,g_t)}\Big)\Big]. \tag{8.13}$$

Scalars $\frac{1}{\alpha_d}, \frac{1}{\alpha_r}$ and $\frac{1}{\alpha_\pi}$ to weight each information term, and γ is the discount factor, since we assume that $T \to \infty$. The first term of this objective can be interpreted as the typical RL objective to maximize the long-term returns. The next two terms correspond to objectives for latent variables.

The learning algorithm we chose to solve this objective for the cartpole problem was REINFORCE [123]. REINFORCE uses the reparameterisation trick to obtain Monte Carlo estimators for a RL objective. Applying this to 8.13, we obtain,

$$\mathcal{L}(\theta,\delta,\omega) := \frac{1}{MIJ}\sum_{m,i,j=1}^{M,I,J} R_0(\tau_m^{i,j}) - \frac{C}{\alpha_d}\mathbb{E}_i[\mathrm{KL}(q_\delta(\cdot|i)||p(\cdot|i))] - \frac{C}{\alpha_r}\mathbb{E}_j\left[\mathrm{KL}(q_\omega(\cdot|j)||p(\cdot|j))\right]$$

where we have used the following definition of the regularized discounted future returns,

$$\widetilde{R}_t(\tau_m^{i,j}) = \sum_{h=0}^{T-1}\gamma^{h-t}\Big(r_j(s_{h,m}^{i,j},a_{h,m}^{i,j}) - \frac{1}{\alpha_\pi}\log\pi_\theta\left(a_{h,m}^{i,j}|s_{h,m}^{i,j},z_{\delta_i}(\epsilon_{h,m}^{i,j}),g_{\omega_j}(\varepsilon_{h,m}^{i,j})\right)\Big),$$

and,

$$C = \frac{1-\gamma^{T-1}}{1-\gamma}.$$

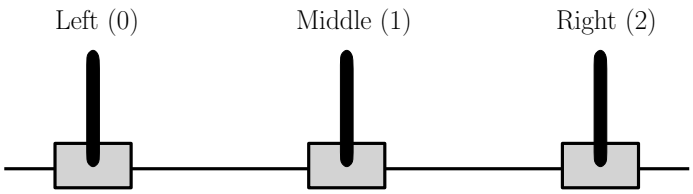


Figure 8.7. The different goal positions used in the training tasks of the cartpole problem for DSE. Note that the masses of the cart also change, but are not depicted here. The cartpoles always started in the middle position (with some noise).

As a practical note, we made use of Pop-Art [113, 40] to adaptively normalise the discounted rewards to have the standard Gaussian. This was because, summing over tasks with sparse rewards caused the algorithm to not try to solve all the tasks, but to only solve those that it detected a reward for faster.

The algorithm for DSE-REINFORCE is given below:

Algorithm 8.4.1 DSE-REINFORCE

For each i, j, initialize parameters δ_i, ω_j and θ
for each iteration **do**
  for each i, j **do**
    Collect trajectories $\{\tau_m^{i,j}\}_{m=1}^{M}$
  end for
  for each i, j **do**
    $\delta_i \leftarrow \delta_i + \nabla_{\delta_i} \mathcal{L}(\theta, \delta, \omega)$
    $\omega_j \leftarrow \omega_j + \nabla_{\omega_j} \mathcal{L}(\theta, \delta, \omega)$
  end for
  $\theta \leftarrow \theta + \nabla_{\theta} \mathcal{L}(\theta, \delta, \omega)$
end for

8.4.2 Experiments

We carried out 3 sets of experiments on the cartpole. For the training phases, we created 9 sets of variations using 3 of each dynamics and reward contexts. The masses chosen were $\{0.2, 1.0, 2.0\}$, and the goals were to go to the $\{\text{left}, \text{middle}, \text{right}\}$, as in Figure 8.7. These were implemented by extending the provided environment in Open AI gym. Each episode lasted for 300 time steps.

Learning performance

The first set of experiments was to study the learning behaviour, and the performance of the algorithm. For this, we trained all tasks defined above, and compared with the following baselines. The first was to train on all tasks using a single embedding, without any disentangling of the dynamics and rewards. Secondly, we trained each task independently, without parameter sharing using a vanilla REINFORCE method. Finally, we compared our algorithm to a personal implementation of Distral [107]. The hyperparameters

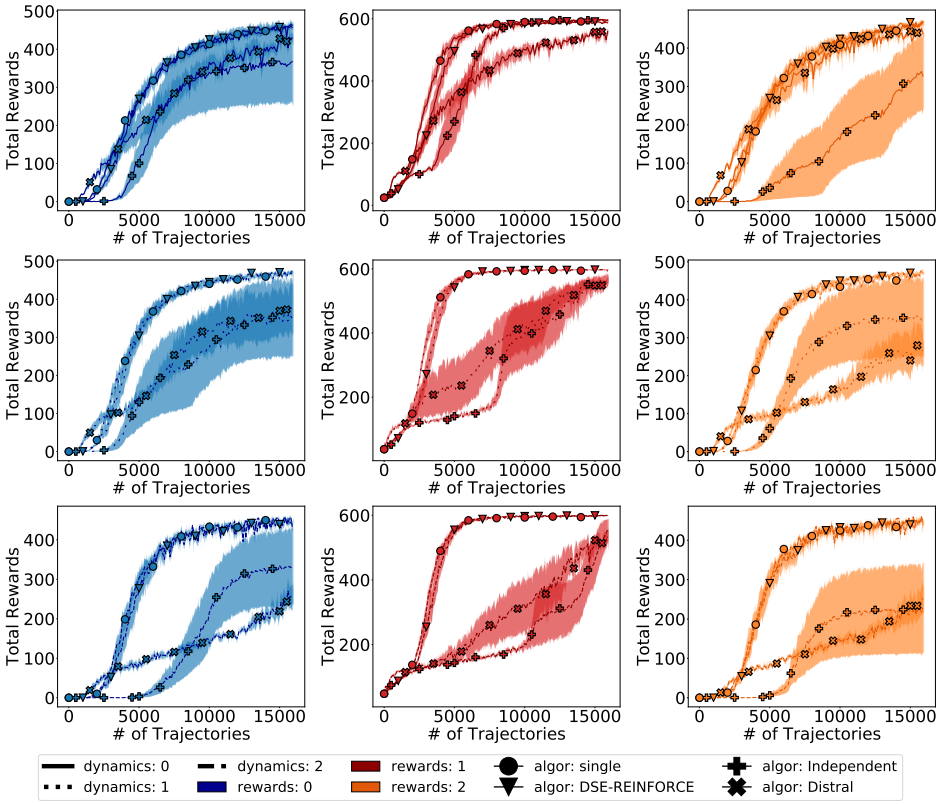


Figure 8.8. Comparison of DSE-REINFORCE against other algorithms. The columns represent the reward conditions, and the rows the dynamics conditions, all in ascending order.

and architectures for these can be found in Appendix C.

The results for this set of experiments is shown in Figure 8.8. These show the averaged reward curves over 4 random seeds. We see that our method performs better than Distral and the independent methods. The speed of learning over the latter case shows empirically that the tasks are related, and can share their data usefully. On average, we performed similarly to methods using a single embedding.

Structure preservation of the latent space

An additional experiment we carried out was to take the mean values of the latent variable g, and interpolate between then, along the line of best fit. This is shown in Figure 8.9(d). We then took these interpolated values, and plugged then into the learned policy, and simulated the trajectories on different the training masses. Note that in this way, g is not sampled, and z was sampled from the appropriate latent variable.

The trajectory of the tip of the pendulum over time is plotted in Figure 8.9(e). We first note that the blue, red and orange paths correspond to moving left, staying in the middle, and moving right. Then, if we choose intermediate value, we see that the balancing point moves *continuously*. That is, there is a

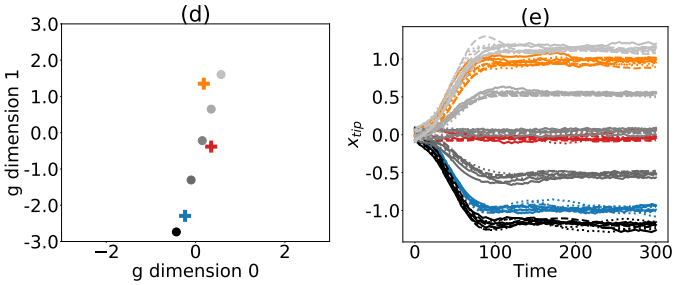


Figure 8.9. Manual generalisation of reward skill. by choosing the value of g. (e) contains the trajectories of the tip of the pole corresponding to the value of g chosen as per (d). The legend of Figure 8.8 applies. The colors in (e) correspond to the values of g in (d).

topological correspondence between the latent space of the rewards, and the goals. This is regardless of the fact that the space g is 2-dimensional; it found a 1-dimensional subspace that behaves in a structure preserving manner.

The final set of experiments tested the transfer capabilities of the learned solutions. We could only compare with the single embedding baseline, since the other algorithms did not allow for transfer; this is clearly not possible in the independent solutions, and the structure of Distral does not provide an obvious mechanism to identify a set of related models, as we need for transfer.

For this, we still used the same set of nine tasks. We held out a select number of tasks, and initially trained both algorithms on the remaining tasks. Then, keeping the parameters θ fixed, and initialising the parameters of the latent variables on the matching indices, we continued training the variational distributions on the remaining tasks.

Figure 8.10 shows the two configurations we had chosen. The (6-3) configuration had an equal number of example cases for each dynamics and goal cases. The latter is on the opposite extreme, where most cases had only one example.

The initial training results are shown in Figure 8.12. As expected, balancing in the middle scored the highest reward, since it is the easiest tasks. For the (6-3) configuration, the other goals behaved similarly, since they are symmetric goals. In (4-5), balancing to the right scored higher since there were more example tasks (2), compared to the single example of the left goal.

Figure 8.12 show us the results of retraining. We see here a clear benefit of DSE; since we have knowledge of which indices of dynamics and rewards contexts are useful, we can interpretably choose a good starting point for transfer. In fact, it could be argued that the retraining is actually carrying out minor corrections.

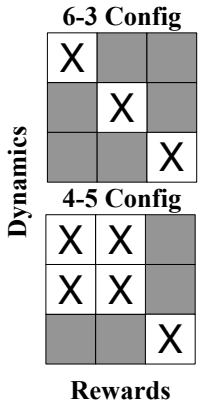


Figure 8.10. The different configurations for training and retraining. The boxes are ordered in ascending order. The crossed out boxes are those held out in the initial training.

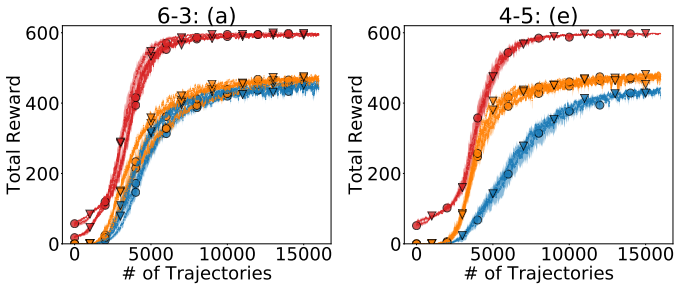


Figure 8.11. The training performances of the initial training sets for each configuration. The legend in Figure 8.8 applies.

8.5 Discussion

Having seen some practical examples of techniques that carry out transfer, we wish to make a few remarks. The framework from the previous chapters describes learning to transfer algorithms as mappings from entire sets of related tasks to sets of related models. In practice, we carry out the learning over a finite sample of these sets.

This provides some insight into the generalisation properties of a learning to transfer solution. An important fact about learning to transfer methods is that they implicitly assume that the set of training tasks contains *related* tasks. Having learned to transfer, we have access to a set of related models. By generalisation in the context of transfer, we mean to evaluate how good the solution to a new task, which is supposedly in the same set of related tasks, is. This varies depending on the tasks that were chosen.

Imagine the extreme case where a single task is used in transfer setting. Suppose there are two models that behave equally well, relative to the loss, and they each lie on a leaf. Each leaf can be thought of as a class of behaviours. Thus, if we assume that each leaf corresponds to a different behaviour, then, it could

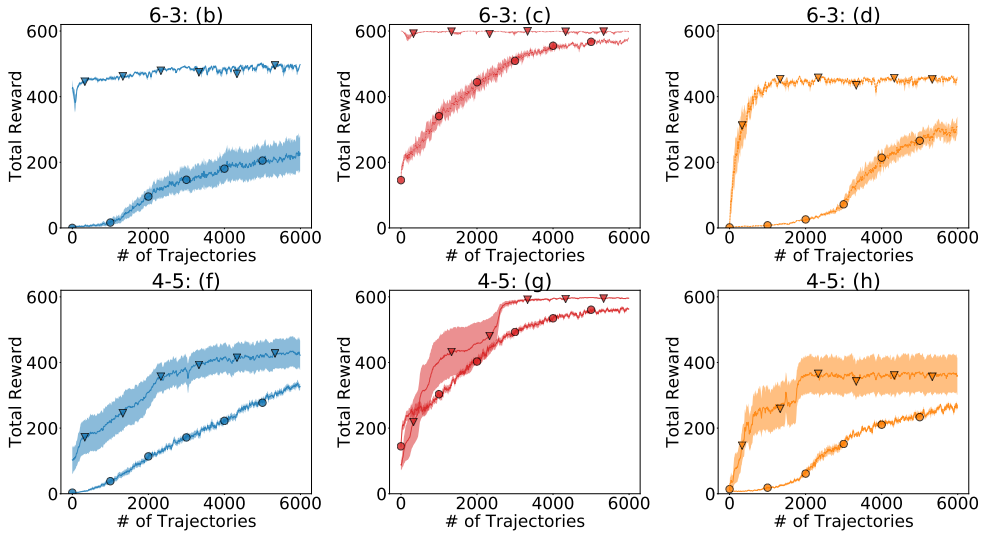


Figure 8.12. Transfer performance of each configuration in Figure 8.10. The legend in Figure 8.8 applies. The columns correspond to the goals left, middle and right, respectively.

be that a related task will behave differently if a solution to it is obtained by transfer.

Consider now the problem of balancing a pendulum of different masses using a constrained torque (i.e. it cannot exceed a maximum). Assume that this problem has a finite episode length, and that the reward function measures the length of time for which the pendulum balances at the top-most position. Given that RL is often in discrete time, it is possible to find a solution where the pendulum spins sufficiently fast that it crosses the top-most balance point at the sampling rate of the system. Thus, even though it is spinning quite fast, the pendulum appears to have solved the task. If we however increase the mass significantly, then the maximum torque of the controller might not be able to spin the pendulum fast enough. Thus, in this case transferring from this solution would not be effective.

This then shows that the tasks that were chosen in the initial learning to transfer can impact generalisation, and could allow us to find the most sensible solutions to learning problems. For example, if in the initial training set of tasks, we had pendulums on either side of the threshold above, then finding a behaviour (i.e. leaf) on which solutions to both exists might force the solution to be the one that actually balances, and is viable for a wider range of tasks. The converse of this however, is that for some tasks, their solution could be better if less tasks were forced to be on the set of related tasks.

This page was left intentionally blank.

Chapter 9

Conclusion

We presented a formulation of machine learning that centered around structure, and showed how we can define the typical components of learning to fit within this framework. Learning, when written in this way is about finding representations of structured systems that *preserve* the appropriate structure. The problem of learning a single task is about preserving the structure of a single system. The main benefit of writing learning in this way is that we can define the problem of learning to transfer as a similar problem, but applied over a set of systems.

Namely, transfer is defined on a set of objects; the objects are learning problems in the learning to transfer settings. We saw that in order to have a useful way of talking about transfer, we need to define a notion of transfer; a systematic way of identifying what it means to transfer. We defined this as a partitioning of our universe. Then, learning to transfer is the process of identifying the particular partition in which our task belongs.

We identified the theory of foliations as a useful way of parameterising these notions of transfer. When spaces of tasks and models are viewed as smooth manifolds, this theory naturally presents itself as a good candidate. We saw that foliations give us a way of identifying the notion of relatedness, which can allow us to interpret learning by transfer.

In order to view space of parametric models as manifolds, we investigated the effects of symmetries in the parameter space of a given architecture. We saw that, even for simple neural network models, trying to analyse these can become quite unwieldy; for this, we gave a brief introduction to GENNI, a novel method of visualising connected regions of symmetric models, with the intention of aiding such investigations.

Finally, we saw how a few examples of popular transfer techniques can be written under this framework. Unfortunately, it appears that these methods make quite simplifying assumptions that possibly avoid the rich properties of a foliated space.

9.1 Future Work

This work opens up a lot of work for the future. We list a few options below; these are in addition to any other possible future work we have mentioned previously in the thesis.

Using full foliation. In Chapter 8 we saw that the algorithms we described made use of local rectification, over a subset of a Euclidean space. This forgoes the complexities of a foliation as a whole. Thus, perhaps it is possible to devise loss functions and learning algorithms that can space the entire manifold of the model space.

In addition to this, perhaps we can take the learning another step higher. In this case, we want to somehow parameterise the foliation it self, and ask which notion of transfer is best suited for the current problem. In a setting like this, we won't assume that all tasks that we have are related, but asks the algorithm to produce the optimal (according to some loss function) related sets to which they belong to.

Evaluating transferrability. Once we have learned the set of related models on which a set of related tasks lie, an assumption that is made when carrying out learning by transfer is that the new task belongs to the same set of related tasks. A natural question that arises is the accuracy of this assumption.

The foliated view of transfer could provide principled methods for measuring this. For example, if loss functions and learning algorithms are well-behaved, then could we use the gradient of the loss from any learned solution, with data from the new learning problem as a measure? Does the assumption that if the optimal set of related models for the new model is close to the learned set, imply that the flow try to stay close to it? If so, measuring the size of the gradient in the transverse direction to the leaf might be a useful measure of transferability.

Alternatively, we could also investigate properties of loss functions that in addition to allowing transfer, provide easy measures of transferability; we expect that equivariant loss functions could play a role here.

Composition of models. The action of groups can be an interesting way to consider compositions. That is, given two groups $\mathcal{G}_1$ and $\mathcal{G}_2$, with actions ρ_1 and ρ_2, we can consider a composed action $\rho_1 \circ \rho_2$ on the semidirect[1] product group space $\mathcal{G}_1 \rtimes \mathcal{G}_2$. We know that foliations have a natural correspondence with groups and actions over manifolds. Could this be used to learn transferrable models that can have a useful, interpretable notion of composability?

A potential application of this, and thus interpretation can be seen in systems biology. A key goal in this field is to identify how complex behaviour can be

[1] Since a semidirect product [58] is more general than a direct product space, it could be used in more interesting cases.

generated by the composition of simpler behaviours [45, 44]. The question is, under some restrictions of potential behaviours, which simpler behaviours can be composed to create variations of the same complex behaviour. We previously said that each leaf can be thought of as a mode of behaviour. Thus, moving in the space of leaves changes the behaviour, and moving along a leaf is a variation of that behaviour. Thus, it could be possible that a foliated structure could be used to carry out the learning of this decomposition (under an assumed method of composability).

Extention to Category Theory. In Section 5.3, we alluded that there is a correspondence between category theory and the theory that we had developed here. It is possible that much of the theory here can be written much more simply in the language of categories. This, and the implications of such a rewriting leaves for an exciting piece of future work.

This page was left intentionally blank.

Bibliography

[1] Adams, J. A. Historical review and appraisal of research on the learning, retention, and transfer of human motor skills. *Psychological bulletin 101*, 1 (1987), 41.

[2] Al-Mualla, M., Canagarajah, C. N., and Bull, D. R. *Video coding for mobile communications: efficiency, complexity and resilience*. Elsevier, 2002.

[3] Andrychowicz, M., Raichuk, A., Stańczyk, P., Orsini, M., Girgin, S., Marinier, R., Hussenot, L., Geist, M., Pietquin, O., Michalski, M., et al. What matters in on-policy reinforcement learning? a large-scale empirical study. *arXiv preprint arXiv:2006.05990* (2020).

[4] Arnol'd, V. I. *Mathematical Methods of Classical Mechanics*, vol. 60. Springer Science & Business Media, 2013.

[5] Barrett, D. G., and Dherin, B. Implicit gradient regularization. *arXiv preprint arXiv:2009.11162* (2020).

[6] Baxter, J. Learning internal representations. In *Proceedings of the Conference on Computational Learning Theory* (1995), pp. 311–320.

[7] Baxter, J. A model of inductive bias learning. *Journal of Artificial Intelligence Research 12* (2000), 149–198.

[8] Ben-David, S., Blitzer, J., Crammer, K., and Pereira, F. Analysis of representations for domain adaptation. In *Advances in Neural Information Processing Systems* (2007), pp. 137–144.

[9] Ben-David, S., and Borbely, R. S. A notion of task relatedness yielding provable multiple-task learning guarantees. *Machine Learning 73*, 3 (2008), 273–287.

[10] Ben-David, S., and Schuller, R. Exploiting task relatedness for multiple task learning. In *Learning Theory and Kernel Machines*. Springer, 2003, pp. 567–580.

[11] Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006.

[12] Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. K. Occam's razor. *Information processing letters 24*, 6 (1987), 377–380.

[13] Camacho, C., and Neto, A. L. *Geometric Theory of Foliations*. Springer Science & Business Media, 2013.

[14] Caruana, R. Multitask learning. *Machine Learning 28*, 1 (1997), 41–75.

[15] Cederquist, J., and Negri, S. A constructive proof of the heine-borel covering theorem for formal reals. In *International Workshop on Types for Proofs and Programs* (1995), pp. 62–75.

[16] Chang, C. C., and Keisler, H. J. *Model theory*. Elsevier, 1990.

[17] Chaudhari, P., Choromanska, A., Soatto, S., LeCun, Y., Baldassi, C., Borgs, C., Chayes, J., Sagun, L., and Zecchina, R. Entropy-sgd: Biasing gradient descent into wide valleys. *Journal of Statistical Mechanics: Theory and Experiment 2019*, 12 (2019), 124018.

[18] Cohen, T., Geiger, M., and Weiler, M. A general theory of equivariant cnns on homogeneous spaces. *arXiv preprint arXiv:1811.02017* (2018).

[19] Cohen, T., Weiler, M., Kicanaoglu, B., and Welling, M. Gauge equivariant convolutional networks and the icosahedral CNN. In *Proceedings of the International Conference on Machine Learning* (2019), pp. 1321–1330.

[20] Cohen, T., and Welling, M. Group equivariant convolutional networks. In *International conference on machine learning* (2016), PMLR, pp. 2990–2999.

[21] Cohen, T. S., Geiger, M., Köhler, J., and Welling, M. Spherical CNNs. *arXiv preprint arXiv:1801.10130* (2018).

[22] Deisenroth, M., and Rasmussen, C. E. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)* (2011), Citeseer, pp. 465–472.

[23] Delange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., and Tuytelaars, T. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).

[24] Dinh, L., Pascanu, R., Bengio, S., and Bengio, Y. Sharp minima can generalize for deep nets. In *Proceedings of the 34th International Conference on Machine Learning* (2017), vol. 70, PMLR, pp. 1019–1028.

[25] Domingos, P. The role of occam's razor in knowledge discovery. *Data mining and knowledge discovery 3*, 4 (1999), 409–425.

[26] Ellis, H. C. *The Transfer of Learning.* Macmillan, 1965.

[27] Elsken, T., Metzen, J. H., and Hutter, F. Neural architecture search: A survey. *Journal of Machine Learning Research 20*, 55 (2019), 1–21.

[28] Esfeld, M. Ontic structural realism and the interpretation of quantum mechanics. *European Journal for Philosophy of Science 3*, 1 (2013), 19–32.

[29] Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the International Conference on Machine Learning* (2017).

[30] French, S., and Ladyman, J. In defence of ontic structural realism. In *Scientific structuralism*. Springer, 2010, pp. 25–42.

[31] Giraud-Carrier, C., and Provost, F. Toward a justification of meta-learning: Is the no free lunch theorem a show-stopper. In *Proceedings of the ICML-2005 Workshop on Meta-learning* (2005), pp. 12–19.

[32] Grant, E., Finn, C., Levine, S., Darrell, T., and Griffiths, T. Recasting gradient-based meta-learning as hierarchical Bayes. In *Proceedings of the International Conference on Representation Learning* (2018).

[33] Grunwald, P. A tutorial introduction to the minimum description length principle. *arXiv preprint math/0406077* (2004).

[34] Haefliger, A. Homotopy and integrability. In *Manifolds—Amsterdam 1970*. Springer, 1971, pp. 133–163.

[35] Hall, B. *Lie Groups, Lie Algebras, and Representations: An Elementary Introduction*, vol. 222. Springer, 2015.

[36] Hausman, K., Springenberg, J. T., Wang, Z., Heess, N., and Riedmiller, M. Learning an embedding space for transferable robot skills. In *Proceedings of the International Conference on Learning Representations* (2018).

[37] Hawkins, T. The mathematics of frobenius in context. *Sources and Studies in the History of Mathematics and Physical Sciences. Springer* (2013).

[38] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778.

[39] Hermann, R. On the accessibility problem in control theory. In *International Symposium on Nonlinear Differential Equations and Nonlinear Mechanics* (1963), Elsevier, pp. 325–332.

[40] Hessel, M., Soyer, H., Espeholt, L., Czarnecki, W., Schmitt, S., and van Hasselt, H. Multi-task deep reinforcement learning with popart. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2019), vol. 33, pp. 3796–3803.

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

[42] Hochreiter, S., and Schmidhuber, J. Flat minima. *Neural computation 9*, 1 (1997), 1–42.

[43] Hospedales, T., Antoniou, A., Micaelli, P., and Storkey, A. Meta-learning in neural networks: a survey. *arXiv preprint arXiv:2004.05439* (2020).

[44] Jaeger, J., and Monk, N. Dynamical modularity of the genotype-phenotype map. In *Evolutionary systems biology*. Springer, 2021, pp. 245–280.

[45] Jaeger, J., and Monk, N. Dynamical modules in metabolism, cell and developmental biology. *Interface focus 11*, 3 (2021), 20210011.

[46] Jones, D. M. The amsfonts package. *Versão 3* (2013), 14.

[47] Jurdjevic, V. *Geometric Control Theory*. Cambridge University Press, 1997.

[48] Kaddour, J., Sæmundsson, S., and Deisenroth, M. P. Probabilistic active meta-learning. In *Advances in Neural Information Processing Systems* (2020).

[49] Kingma, D. P., and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

[50] Kingman, J., and Taylor, S. *Introduction to measure theory and probability*. Cambridge, 1966.

[51] Kolmogorov, A. N., and Fomin, S. V. *Elements of the theory of functions and functional analysis*. Courier Corporation, 1957.

[52] Kuhn, T. S. *The Structure of Scientific Revolutions*. University of Chicago press, 2012.

[53] Lavau, S. A short guide through integration theorems of generalized distributions. *Differential Geometry and its Applications 61* (2018), 42–58.

[54] Lawson Jr, H. B. Codimension-one foliations of spheres. *Annals of Mathematics* (1971), 494–503.

[55] Lawson Jr, H. B. Foliations. *Bulletin of the American Mathematical Society 80*, 3 (1974), 369–418.

[56] Leberman, S., and McDonald, L. *The transfer of learning: Participants' perspectives of adult education and training.* CRC Press, 2016.

[57] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. *Neural computation 1*, 4 (1989), 541–551.

[58] Lee, J. M. *Introduction to Smooth Manifolds.* Springer, 2001.

[59] Lengyel, D., Petangoda, J., Falk, I., Highnam, K., Lazarou, M., Kolbeinsson, A., Deisenroth, M. P., and Jennings, N. R. Genni: Visualising the geometry of equivalences for neural network identifiability. *arXiv preprint arXiv:2011.07407* (2020).

[60] Levine, S. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909* (2018).

[61] Lisle, I. G., and Reid, G. J. Geometry and structure of lie pseudogroups from infinitesimal defining systems. *Journal of Symbolic Computation 26*, 3 (1998), 355–379.

[62] Mac Lane, S. *Categories for the working mathematician*, vol. 5. Springer Science & Business Media, 2013.

[63] Marr, D. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information.* Henry Holt and Co., Inc., 1982.

[64] McGehee, R. The stable manifold theorem via an isolating block. In *Symposium on Ordinary Differential Equations* (1973), Springer, pp. 135–144.

[65] McInnes, L., Healy, J., and Melville, J. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426* (2018).

[66] Mendelson, B. *Introduction to Topology.* Courier Corporation, 1990.

[67] Mitchell, T. Machine learning.

[68] Mitchell, T. M. *The Need for Biases in Learning Generalizations.* Department of Computer Science, Laboratory for Computer Science Research, 1980.

[69] Moerdijk, I., and Mrcun, J. *Introduction to foliations and Lie groupoids*, vol. 91. Cambridge university press, 2003.

[70] Nakahara, M. *Geometry, Topology and Physics.* CRC Press, 2003.

[71] Ng, A. Y., Harada, D., and Russell, S. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml* (1999), vol. 99, pp. 278–287.

[72] Olver, P. J. *Equivalence, Invariants and Symmetry*. Cambridge University Press, 1995.

[73] Pan, S. J., Tsang, I. W., Kwok, J. T., and Yang, Q. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks 22*, 2 (2010), 199–210.

[74] Pan, S. J., and Yang, Q. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering 22*, 10 (2009), 1345–1359.

[75] Petangoda, J., Deisenroth, M. P., and Monk, N. A. Learning to transfer: A foliated theory. *arXiv preprint arXiv:2107.10763* (2021).

[76] Petangoda, J. C., Pascual-Diaz, S., Adam, V., Vrancx, P., and Grau-Moya, J. Disentangled skill embeddings for reinforcement learning. *arXiv preprint arXiv:1906.09223* (2019).

[77] Rasmussen, C., and Ghahramani, Z. Occam's razor. *Advances in neural information processing systems 13* (2000).

[78] Reddy, D., and Jana, P. K. Initialization for k-means clustering using voronoi diagram. *Procedia Technology 4* (2012), 395–400.

[79] Reeb, G. Sur certaines propriétés topologiques des variétés feuilletées. *Act. Sc. et Ind.* (1952).

[80] Rosenberg, H., and Roussarie, R. Reeb foliations. *Annals of Mathematics* (1970), 1–24.

[81] Royer, J. M. Theories of learning transfer. *Center for the Study of Reading Technical Report; no. 079* (1978).

[82] Ruder, S. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098* (2017).

[83] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning internal representations by error propagation. Tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[84] Russell, S. J. Preliminary steps toward the automation of induction. In *AAAI* (1986), pp. 477–484.

[85] Sacksteder, R. On the existence of exceptional leaves in foliations of co-dimension one. In *Annales de l'institut Fourier* (1964), vol. 14, pp. 221–225.

[86] Sæmundsson, S., Hofmann, K., and Deisenroth, M. P. Meta reinforcement learning with latent variable Gaussian processes. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence* (2018).

[87] Saemundsson, S., Terenin, A., Hofmann, K., and Deisenroth, M. Variational integrator networks for physically structured embeddings. In *International Conference on Artificial Intelligence and Statistics* (2020), PMLR, pp. 3078–3087.

[88] Schaffer, C. A conservation law for generalization performance. In *Machine Learning Proceedings 1994*. Elsevier, 1994, pp. 259–265.

[89] Schmidhuber, J. On learning how to learn learning strategies. Tech. rep., TODO: FINDOUT, 1995.

[90] Schmidhuber, J., Zhao, J., and Wiering, M. A. Simple principles of metalearning. *Technical Report IDSIA 69* (1996), 1–23.

[91] Schunk, D. H. *Learning Theories, an Educational Perspective*. Pearson, 2012.

[92] Shalev-Shwartz, S., and Ben-David, S. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[93] Shipley, J. O., and Dolan, S. R. Binary black hole shadows, chaotic scattering and the cantor set. *Classical and Quantum Gravity 33*, 17 (2016), 175001.

[94] Silver, D. L., and Mercer, R. E. The parallel transfer of task knowledge using dynamic learning rates based on a measure of relatedness. In *Learning to learn*. Springer, 1996, pp. 213–233.

[95] Skinner, B. The science of learning and the art of teaching. *Harvard Education Revue, vol. 24* (1954).

[96] Smith, S. L., Dherin, B., Barrett, D. G., and De, S. On the origin of implicit regularization in stochastic gradient descent. *arXiv preprint arXiv:2101.12176* (2021).

[97] Snell, J., Swersky, K., and Zemel, R. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems* (2017), pp. 4077–4087.

[98] Stefan, P. Accessible sets, orbits, and foliations with singularities. *Proceedings of the London Mathematical Society 3*, 4 (1974), 699–713.

[99] Steiner, G. Transfer of learning, cognitive psychology of. In *International Encyclopedia of the Social and Behavioral Sciences*, N. J. Smelser and P. B. Baltes, Eds. Pergamon, Oxford, 2001, pp. 15845–15851.

[100] Strogatz, S. H. *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. CRC press, 2018.

[101] Subedi, B. S. Emerging trends of research on transfer of learning. *International education journal 5*, 4 (2004), 591–599.

[102] Sussmann, H. J. Orbits of families of vector fields and integrability of distributions. *Transactions of the American Mathematical Society 180* (1973), 171–188.

[103] Sussmann, H. J. Uniqueness of the weights for minimal feedforward nets with a given input-output map. *Neural networks 5*, 4 (1992), 589–593.

[104] Sutton, R. S., and Barto, A. G. *Reinforcement Learning: An Introduction*. MIT press, 2018.

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

[106] Tamura, I. *Topology of Foliations: An Introduction: An Introduction*, vol. 97. American Mathematical Soc., 1992.

[107] Teh, Y., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N., and Pascanu, R. Distral: Robust multi-task reinforcement learning. *Advances in neural information processing systems 30* (2017).

[108] Thorndike, E. L. *Educational psychology, Vol 2: The psychology of learning.* Teachers College, 1913.

[109] Thrun, S., and Mitchell, T. M. Learning one more thing. Tech. rep., Carenegie Mellon University, 1994.

[110] Thrun, S., and Mitchell, T. M. Lifelong robot learning. *Robotics and Autonomous Systems 15*, 1-2 (1995), 25–46.

[111] Trench, W. F. *Elementary differential equations with boundary value problems.* 2013.

[112] Valentine, J. W. *On the origin of phyla.* University of Chicago Press, 2004.

[113] van Hasselt, H. P., Guez, A., Hessel, M., Mnih, V., and Silver, D. Learning values across many orders of magnitude. *Advances in Neural Information Processing Systems 29* (2016).

[114] Vapnik, V. N., and Chervonenkis, A. Y. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of Complexity*. Springer, 2015, pp. 11–30.

[115] Vilalta, R., and Drissi, Y. A perspective view and survey of meta-learning. *Artificial Intelligence Review 18*, 2 (2002), 77–95.

[116] Vlačić, V., and Bölcskei, H. Neural network identifiability for a family of sigmoidal nonlinearities. *Constructive Approximation* (2021), 1–52.

[117] Vondrick, C., Pirsiavash, H., and Torralba, A. Anticipating visual representations from unlabeled video. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 98–106.

[118] Wang, M., and Deng, W. Deep visual domain adaptation: A survey. *Neurocomputing 312* (2018), 135–153.

[119] Wang, Y., Yao, Q., Kwok, J. T., and Ni, L. M. Generalizing from a few examples: A survey on few-shot learning. *ACM Computing Surveys 53*, 3 (2020), 1–34.

[120] Watkins, C. J., and Dayan, P. Q-learning. *Machine learning 8*, 3-4 (1992), 279–292.

[121] Webb, G. I. Further experimental evidence against the utility of occam's razor. *Journal of Artificial Intelligence Research 4* (1996), 397–417.

[122] Weyl, H. *Symmetry*, vol. 104. Princeton University Press, 2015.

[123] Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning 8*, 3 (1992), 229–256.

[124] Wolpert, D. H. The lack of a priori distinctions between learning algorithms. *Neural computation 8*, 7 (1996), 1341–1390.

[125] Wolpert, D. H., and Mcreader, W. G. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation 1*, 1 (1997), 67–82.

[126] Worrall, J. Structural realism: The best of both worlds? *Dialectica 43*, 1-2 (1989), 99–124.

This page was left intentionally blank.

Appendix

This page was left intentionally blank.

Appendix A

Mathematical Preliminaries

We provide some background details of mathematical principles that were used in the thesis.

A.1 Topology

A topology is a simple structure one can place on a set $\mathcal{M}$.

Definition of a Topology

Definition A.1.1 (Topology)**.** *Given a set X, a topology $\mathcal{O}_X$ on X is a set of subsets of X that satisfy the following conditions:*

a) $X \in \mathcal{O}_X$ *and* $\varnothing \in \mathcal{O}_X$*,*

b) for $U_i \in \mathcal{O}_X$*,* $\bigcup_{i\in\mathcal{I}} U_i \in \mathcal{O}_X$ *(closed under* arbitrary *unions),*

c) $U_1, U_2, ..., U_n \in \mathcal{O}_X \implies \bigcap_{i=1}^{n} U_i \in \mathcal{O}_X$ *(closed under* finite *intersections).*

Simply put, a topology $\mathcal{O}_\mathcal{M}$ is a set of subsets of a set which is closed under (finite) intersections and (arbitrary) unions, and also contains the null set $\varnothing$, and the set $\mathcal{M}$ itself. An element of a topology $U \in \mathcal{O}_M$, which is a subset of M is called an *open set*, the difference of which with M is called a *closed set*. Despite these names, it is incorrect to think of an open set as an open interval; particularly, some sets can be both open and closed (for example, the full set $\mathcal{M}$). The tuple $(\mathcal{M}, \mathcal{O}_\mathcal{M})$ is called a topological space.

Closed sets can have other characterisations. One such is that is a set which contains all its limit points; that is, the limits of all convergent sequences in a closed set is contained in the closed set.

An intuitively familiar example of a topology is the called the standard topology $\mathcal{O}^S_{\mathbb{R}^d}$ on $\mathbb{R}^d$. To define the standard topology, one defines its elements, the open sets implicitly as follows. An open set $U \in \mathcal{O}^S_{\mathbb{R}^d}$ iff $\forall p \in U$, $\exists r \in [0, \infty]$: $\mathcal{B}^r(p) \subseteq U$, where $\mathcal{B}^r(p) := \{x \mid \sqrt{\sum_{i=0}^{d}(x_i - p_i)^2}\}$ are open balls centred at p. This topology in $\mathbb{R}^2$ is depicted in Figure A.1.

An important idea a topology affords us is a neighbourhood of a point. Different

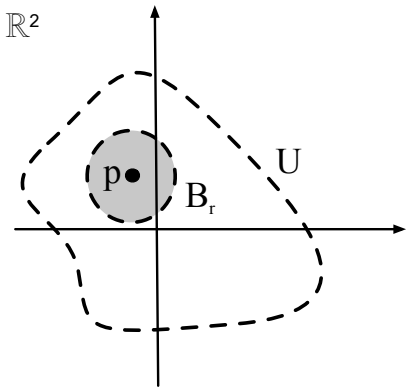


Figure A.1. An open set in the standard topology on $\mathbb{R}^2$. Note that the dashed lines imply that the boundary is not included. Furthermore, the open set U is populated with other open balls at all other points $p \in U$.

authors define this slightly differently, but these differences are inconsequential to our discourse. [58] defines a neighbourhood of a point $p \in \mathcal{M}$ as an open set $U_p \in \mathcal{O}_\mathcal{M}$ that contains p; on the other hand, [66] define $A_p \subseteq \mathcal{M}$ to be a neighbourhood of p if it completely contains an open set $U_p \in \mathcal{O}_\mathcal{M}$ that contains p. Presently, we will stick to the former definition by [58], since it is simpler to consider.

At this point, it is worth noting the relationship between topological spaces and metric spaces. Loosely, a metric space is a set equipped with a metric, which is a numerical measure of distance between points; this is defined more formally in Definition A.1.2.

Definition of a Metric

Definition A.1.2 (Metric)**.** *Given a set X, a metric is a map $\rho : X \times X \to \mathbb{R}^+$ that satisfies:*

a) $\rho(x, y) = 0 \iff x = y$,

b) $\rho(x, y) = \rho(y, x)$ *(symmetry),*

c) $\rho(x, y) \leq \rho(x, z) + \rho(z, y)$ *(triangle inequality),*

for $x, y, z \in X$.

Every metric space defines a topological space through the metric topology; this is a topology similar to the standard topology, where the open balls are defined w.r.t. the metric on the metric space. In fact, the standard topology is the metric topology on $\mathbb{R}^d$ defined w.r.t. the Euclidean metric. There are, however topological spaces which do not have a corresponding metric; those that do, are called *metrizable* [66]. In this way, a topological space is more general. That is, as in the case of the circle example above, we are allowed to consider *closeness* in terms of a metric, but it is not the only allowed way. Choosing a metric fixes the *shape* of a space; for example, topologically, a geometric circle and a geometric ellipse are topologically equivalent to the

circle space $\mathbb{S}^1$.

Now, we can imagine transforming a circle into an ellipse by simply stretching or squashing actions (imagine a spherical balloon being squashed between 2 poles). Importantly, we do not tear (or create holes) the space. Such a transformation, that maps between 2 topological spaces is called a continuous map. More specifically, a continuous map is the topology preserving map; a map $f : X \to Y$ between 2 topological spaces $(X, \mathcal{O}_X)$ and $(Y, \mathcal{O}_Y)$ is *continuous* if the pre-image of open sets in Y is open in X. If such a map is invertible, f is said to be a *homeomorphism*, which is the special name given to topological isomorphisms. Thus, the map between a circle and an ellipse is a homeomorphism.

We note that in this way of defining a continuous map, the only necessary information about X and Y are their topologies. A typical way of defining a continuous map is in terms of the $\epsilon - \delta$ notation. Such a definition is only possible on a metric space, and we state that the definition given above is therefore more general. Of course, for a metrizable topological space, the $\epsilon - \delta$ definition follows once a suitable metric is found.

A topology allows us to then define a host of useful and important other notions that are preserved under continuous mappings. One such is the notion of compactness. Intuitively, one can think of a compact subset as a subset that is closed and bounded; for $\mathbb{R}^d$, the Heine-Borel Theorem [15] says that this is exactly true. A compact set remains compact under a continuous map. Another such property is that of (path)-connectedness. A subset A is (path)-connected if one can draw a continuous line $\gamma_{pq} : [0, 1] \to A$ between any two points $p, q \in A$, where $\gamma(0) = p$ and $\gamma(1) = q$. Generally, path-connectedness and connectedness are distinct properties, but this distinction is not necessary in the present work, and we mean path-connectedness if we state that a set is connected.

A.2 Measure Theory and Probability

It is useful in measure theory to define an *extended* real line,

Definition of a Extended Real Line

Definition A.2.1 (Extended Real Line)**.** *An extended real line, denoted by $\mathbb{R}^*$ is a compactification of the real number field, by adds $-\infty$ and ∞ to it. In this field, the following rules for addition and multiplication are satisfied for $x \in \mathbb{R}$:*

a) $-\infty < x < \infty$

b) $x \pm \infty = \infty$, $x \pm -\infty = -\infty$,

c) if $x > 0$, $x \times \pm\infty = \pm\infty$,

d) if $x = 0$, $x \times \pm\infty = 0$,

e) if $x < 0$, $x \times \pm\infty = \mp\infty$, *and*

f) $\infty \times \infty = \infty$, $-\infty \times -\infty = \infty$ *and* $\infty \times -\infty = -\infty$.

It is assumed that $\mathbb{R}$ is endowed with the standard topology. The *non-negative* extended real line, as the name suggests, places the addition constraint that all elements of it must be non-negative. That is,

Definition of a Non-negative Extended Real Line

Definition A.2.2 (Non-negative Extended Real Line)**.** *The non-negative extended real line, denoted as* $\mathbb{R}^+$ *is the set,*

$$\mathbb{R}^+ = \{x | x \in \mathbb{R}^*, x \geq 0\}.$$

The elementary structure that we place on a set in measure theory [50, 51] is a $\sigma-$algebra, which is defined as follows;

Definition of a $\sigma-$algebra

Definition A.2.3 ($\sigma-$algebra)**.** *Given a set* X*, a* σ*-algebra* Σ_X *on* X *is a set of subsets of* X *that satisfy:*

a) $X \in \Sigma_X$,

b) if $U \in \Sigma_X$*, then* $U^c \in \Sigma_X$,

c) if $\{U_n\}_{n\in\mathcal{I}}$ *is a sequence of elements in* Σ_X*, then* $\bigcup_{n\in\mathcal{I}} U_n \in \Sigma_X$.

Note that Condition (b), together with Condition (1) implies that $\varnothing \in \Sigma_X$.

Definition of a Measurable Space

Definition A.2.4 (Measurable Space)**.** *A set* X*, together with a* σ*-algebra* Σ_X *is called a measurable space.*

A special kind of σ-algebra is the Borel σ-algebra;

Definition of a Borel σ-algebra

Definition A.2.5 (Borel σ-algebra)**.** *Given a topological space* $(X, \mathcal{O}_X)$*, a Borel* σ*-algebra on* X*, denoted by* $\mathfrak{B}_X$ *is the smallest* σ*-algebra that contains the open sets of* $\mathcal{O}_X$.

One can define maps between measurable spaces that *respect* the measure-theoretic structures in the following way;

Definition of a Measurable Map

Definition A.2.6 (Measurable Map)**.** *Given measurable spaces* (X, Σ_X)*, and* (Y, Σ_Y)*, a measurable map is a map,*

$$f : X \to Y,$$

that for $U \in \Sigma_Y$*, the preimage w.r.t* f *satisfies,*

$$f^{-1}[U] \in \Sigma_X$$

A measurable space can then be endowed with the key structure of measure theory, a measure;

Definition of a Measure

Definition A.2.7 (Measure)**.** *A measure* μ *on a measurable space* (X, Σ_X) *is a map,*

$$\mu : \Sigma_X \to \mathbb{R}^+$$

that satisfies,

a) $\mu(\varnothing) = 0$,

b) for $\{U_n\}_{n\in\mathcal{I}}$, *where* $U_n \in \Sigma_X$ *and* $U_i \cap U_j = \varnothing$ *for* $i, j \in \mathcal{I}$,

$$\mu\Big(\bigcup_{n=1}^{N} U_n\Big) = \sum_{n=1}^{N} \mu(U_n).$$

A measurable space, together with a measure makes a measure space;

Definition A.2.8 (Measure Space)**.** *A tuple* (X, Σ_X, μ) *of a set, a* σ*-algebra on this set, and a measure is called a measure space.* Definition of a Measure Space

If a measurable function is defined between a measure space and a measurable space, then it induces a measure on its codomain as,

Definition A.2.9 (Induced Measure)**.** *Given a measure space* (X, Σ_X, μ_X), *a measurable space* (Y, Σ_Y), *and a measurable map* $f : X \to Y$, *a measure is induced on* (Y, Σ_Y) *by* f, *where,* Definition of a Induced Measure

$$\begin{aligned} \mu_Y : \Sigma_Y &\to \mathbb{R}^+ \\ U &\mapsto \mu_Y(U) = \mu_X(f^{-1}[U]). \end{aligned}$$

We can prove that μ_Y is a valid measure;

Theorem A.2.1. μ_Y *as defined in Definition A.2.9 is a measure.* Induced measure is a measure

Proof. We need to show that μ_Y satisfies the conditions of a measure. This can be shown as follows:

a) since $f^{-1}[\varnothing] = \varnothing$, $\mu_Y(\varnothing) = \mu_X(f^{-1}[\varnothing]) = \mu_X(\varnothing) = 0$,

b) since for any set of subsets $\{U_n\}_{n\in\mathcal{I}}$,

$$f^{-1}\Big(\bigcup_{n\in\mathcal{I}} U_n\Big) = \bigcup_{n\in\mathcal{I}} f^{-1}(U_n),$$

we can say that for $\{U_n\}_{n\in\mathcal{I}}$, where $U_n \in \Sigma_Y$ and $U_i \cap U_j = \varnothing$ for $i, j \in \mathcal{I}$,

$$\begin{aligned} \mu_Y\Big(\bigcup_{n\in\mathcal{I}} U_n\Big) &= \mu_X\Big(f^{-1}\Big(\bigcup_{n\in\mathcal{I}} U_n\Big)\Big) = \mu_X\Big(\bigcup_{n\in\mathcal{I}} f^{-1}(U_n)\Big) \\ &= \sum_{n\in\mathcal{I}} \mu_X(f^{-1}(U_n)) \\ &= \sum_{n\in\mathcal{I}} \mu_Y(U_n). \end{aligned}$$

■

A probability space is a special kind of measure space, where the measure satisfies the conditions of a probability measure;

Definition of a Probability Measure

Definition A.2.10 (Probability Measure)**.** *Given a measurable space* (X, Σ_X)*, a probability measure* ρ *is a map,*

$$\rho : \Sigma_X \to [0, 1],$$

that satisfies,

a) for $\rho(\varnothing) = 0$*,*

b) for $\{U_n\}_{n=1}^N$*, where* $U_n \in \Sigma_X$ *and* $U_i \cap U_j = \varnothing$ *for* $1 \leq i, j \leq N$*,*

$$\rho\Big(\bigcup_{n=1}^{N} U_n\Big) = \sum_{n=1}^{N} \rho(U_n),$$

and,

c) $\rho(X) = 1$*.*

Thus,

Definition of a Probability Space

Definition A.2.11 (Probability Space)**.** *A probability space is a tuple* (X, Σ_X, ρ) *of a set, a* σ*-algebra on this set, and a probability measure.*

A random variable is a measurable map between a probability space and the extended real line, endowed with a Borel σ-algebra;

Definition of a Random Variable

Definition A.2.12 (Random Variable)**.** *Given a probability space* (X, Σ_X, ρ) *and the measurable space* $(\mathbb{R}^*, \mathfrak{B}_{\mathbb{R}^*})$*, a random variable* x *is a measurable map,*

$$x : X \to \mathbb{R}^*.$$

A.3 Manifolds

A relatively simple, yet powerful object that we can start with is a topological manifold:

Definition of a Topological manifold

Definition A.3.1 (Topological manifold)**.** *A tuple* $(\mathcal{M}, \mathcal{O}_\mathcal{M}, \mathcal{A}_\mathcal{M})$*, is called a d-dimensional topological manifold if:*

a) $\mathcal{M}$ *is a set,*

b) $\mathcal{O}_\mathcal{M}$ *is a topology on* $\mathcal{M}$*, where* $(U \in \mathcal{O}_\mathcal{M}) \subseteq \mathcal{M}$ *is called an open set. The topology must be Hausdorff and paracompact.*

c) $\mathcal{A}_\mathcal{M}$ *is a collection of charts, where a chart* $(U, \phi) \in \mathcal{A}_\mathcal{M}$ *consists of an open set and a homeomorphism* $\phi : U \to (U_{\mathbb{R}^d} \subseteq \mathbb{R}^d)$*.*

Here, $\mathbb{R}^d$ *is the usual real space endowed with* $\mathcal{O}^S_{\mathbb{R}^d}$*, the standard topology.*

A topological manifold is created by endowing a topological space with a set of charts, called an atlas. A chart, from Definition A.3.1 consists of a tuple of an open set $U \in \mathcal{O}_\mathcal{M}$ and a map ϕ that is a homeomorphism between U and a subset $U_{\mathbb{R}^d}$ of $\mathbb{R}^d$. This then is a generalisation of what we think of

as the topological space $\mathbb{R}^d$; that is such a d dimensional manifold is *locally* homeomorphic to $\mathbb{R}^d$. The key aspect here is that the set $\mathcal{M}$ can be given any topology $\mathcal{O}_\mathcal{M}$ on it, but we can still locally relate it to a space we are familiar with (the real space); see Figure A.2. Each map ϕ that does this is called the *coordinate map*. An atlas is a collection of such charts, such that the domains of each of the chart maps is a cover of M.

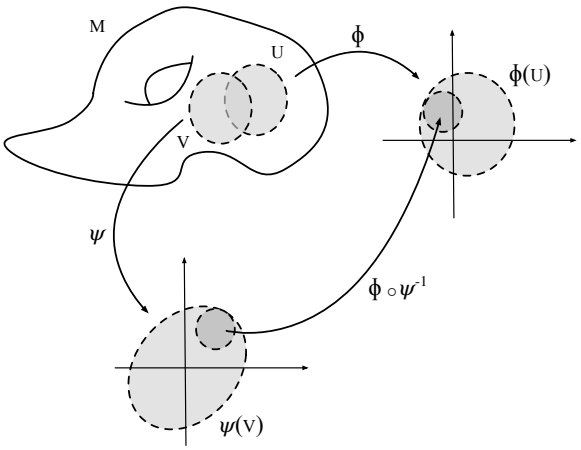


Figure A.2. A topological manifold $\mathcal{M}$ with examples of its coordinate charts. We see here that despite being a complicated topological entity (See the hole in the middle), locally, it is equivalent to $\mathbb{R}^2$. Further, note that the chart transition map $\phi \circ \psi^{-1} : V_{\mathbb{R}^2} \to U_{\mathbb{R}^2}$ exists because a homeomorphism is invertible, and is continuous since the composition of continuous maps remains so.

We note here that because open sets of the manifold are homeomorphic to $\mathbb{R}^d$, which is endowed with the standard topology, our manifold is locally compact. This means that every point on the manifold has a neighbourhood that is contained in a compact subset of X; intuitively, compactness gives us a notion of being closed (contains limit points) and bounded [15]. In addition to this, our manifolds are *Hausdorff* [58]. This means that if we take any 2 points on a manifold, there exists at least one neighbourhood of each point such that these subsets are non-intersecting. These properties give our manifolds nice properties that make them easy to work with. As an example, a trivial topology that can be given to a set X is $\{X, \varnothing\}$. It is possible to show that every map onto X is continuous, and as such this topology doesn't provide us with any useful structure; the Hausdorff property ensures that there are enough open sets to reason about interesting things.

In Figure A.2, there is a map that is defined to take elements in the intersection of two open sets from one chart to another. More precisely, given $U, V \in \mathcal{O}_\mathcal{M}$ and charts on these sets $(U, \phi), (V, \psi) \in \mathcal{A}_\mathcal{M}$, where $\phi : U \to (U_{\mathbb{R}^d} \subset \mathbb{R}^d)$ and $\psi : V \to (V_{\mathbb{R}^d} \subset \mathbb{R}^d)$, and if $U \cap V \neq \varnothing$, then $\phi \circ \psi^{-1} : \psi(U \cap V) \to \phi(U \cap V)$. Such a map is called a chart transition map. These maps are important in

defining differential properties on a manifold.

Definition of a C^k - Atlas

Definition A.3.2 (C^k - Atlas)**.** *An atlas $\mathcal{A}_\mathcal{M}$ of a topological manifold $\mathcal{M}$ is called a C^k-atlas if all chart transition maps are k times continuously differentiable.*

In this way, if our $\mathcal{A}_\mathcal{M}$ atlas satisfies these conditions, we call $\mathcal{M}$ a C^k-differentiable manifold. A smooth manifold then is a C^∞-manifold; that is, it is infinitely differentiable. Such properties are defined to enable us to carry out calculus on manifolds in an invariant way; topologies don't provide enough restrictions because it is possible to construct maps that are homeomorphisms, but don't preserve differentiability.

Furthermore, requiring that smoothness is preserved at the intersection of open sets also exposes another key aspect of the philosophy of manifolds, differential or otherwise. A manifold attempts to describe its properties in a coordinate independent way. That is, the manifold exists, with its properties, without the need for chart maps; the charts are our ways of interacting with the manifold. However, if the chart is valid, and respects the conditions we put on it, then our choice of a chart shouldn't change the properties of the underlying manifold.

This is because if there are two charts in our atlas, $(U, \phi), (U, \psi) \in \mathcal{A}_\mathcal{M}$, which are based on the same open set $U \in \mathcal{O}_\mathcal{M}$, then doing either chart transition $\phi \circ \psi^{-1}$ or $\psi \circ \phi^{-1}$ should not alter any properties we see on either coordinate. Thus, the condition in Definition A.3.2 ensures that smoothness properties are preserved between coordinates, and allows us to measure whether they can be preserved through maps between differential manifolds. A map f on M is said to be smooth, if $f \circ \phi^{-1}$ is smooth, for any $(U, \phi) \in \mathcal{A}_\mathcal{M}$. A map that does preserve smoothness, and is invertible is called a *diffeomorphism*. In this way, the chart transition maps are therefore perhaps the most obvious examples of diffeomorphisms.

As a slight tangent, this philosophy contributes to the beauty of differential geometry. It allows us to define properties intrinsic to a space without worrying about how we choose to represent them; these properties are then independent of the coordinate system we choose. This is of course a very desirable attribute, since we don't want to worry about whether our theories are the way they are because of a particular choice in coordinates that we made.

A.4 Groups and Group Actions

Definition of a Group

Definition A.4.1 (Group)**.** *A set $\mathcal{G}$ with an operation $\bullet : \mathcal{G} \times \mathcal{G} \to \mathcal{G}$ is called a group $(\mathcal{G}, \bullet)$ if:*

a) $\bullet$ is associative,

b) there exists $e \in \mathcal{G}$ such that $e \bullet g = g$ for any $g \in \mathcal{G}$, and,

c) $\forall g \in \mathcal{G}$ there exists $g^{-1} \in \mathcal{G}$ such that $g \bullet g^{-1} = e$ and $g^{-1} \bullet g = e$.

A group is said to be commutative or *Abelian* if $g \bullet h = h \bullet g$.

The key way in which we will be using a group is through its action on another set.

Definition A.4.2 (Group action (left))**.** *The (left) action of a group* $(\mathcal{G}, \bullet)$ *on a set* $\mathcal{X}$ *is a map* $\rho : \mathcal{G} \times X \to X$ *that satisfies:* Definition of a Group action (left)

a) $\rho(h, \rho(g, x)) = \rho(h \circ g, x)$,

b) $\rho(e, x) = x$,

for $g, h \in \mathcal{G}$, $e \in \mathcal{G}$ *is the neutral element in* $\mathcal{G}$, *and* $x \in X$.

A right action reverses the order of the input; we brevity, we will only consider left group actions, and therefore won't qualify which we use. A group action gives us a way to traverse a set in terms of a set of transformations. That is, we can think of a group as a set of transformations, and the action of an element of a group is applying a particular transformation an element of another group. An example of a group is SO(3), which is group of all spherical rotations; an element of SO(3) can act on an element of $\mathbb{R}^3$ by rotating it, in the intuitive sense.

Groups play an important role in the idea of symmetries and invariances [122]. A particular reason for that are the two conditions that a group action must satisfy. These conditions ensure a sort of consistency in the behaviour of the action. Consider the following scenario. We have three points $x, y, z \in X$. Suppose $y = g_{xy}(x)$, $z = g_{yx}(y)$ and the $z = g_{xz}(x)$. The first condition ensures that $g_{xz} = g_{xy} \bullet g_{yz}$, that we can blindly choose from $\mathcal{G}$ whether to move from x to z directly, or to do so through an intermediate point y. The second condition ensures this consistency holds for the neutral element of the group too.

Definition A.4.3 (Orbit)**.** *Given a group* $\mathcal{G}$ *and its action* ρ *on a set* X, *the orbit of* $\mathcal{G}$ *centered at a point* $x \in X$, *which is denoted by* $O_{\mathcal{G}}(x)$, *is a set defined as:* Definition of a Orbit

$$O_{\mathcal{G}}(x) = \{y \mid y = \rho(g, x)\, \forall g \in \mathcal{G}\}.$$

Since $\mathcal{G}$ includes the neutral element, $x \in O_{\mathcal{G}}(x)$. The consistency requirements of a group action, together with the requirements of a group imply that an orbit forms a well defined equivalence class. That is, if $x, y \in O_{\mathcal{G}}(x)$, then $O_{\mathcal{G}}(x) = O_{\mathcal{G}}(y)$. In other words, there is an equivalence relation, where $x \sim y$ if $\exists g \in \mathcal{G}$ such that $x = g(y)_{\mathcal{G}}$, which is of course used in the definition of an orbit. With an equivalence relation comes a quotient space $X/\sim$; this is the space of equivalence classes. The quotient space defined by orbits of a $\mathcal{G}$ is denoted by $X/\mathcal{G}$ and is called the *orbit space*.

A group action is called *free* if its isotropy groups are trivial, and only contain the neural element. This implies that for a free group action, if $x, y \in O_{\mathcal{G}}(x)$, then $O_{\mathcal{G}}(x) = O_{\mathcal{G}}(y)$. This means that the orbits of a free action are well defined equivalence classes.

An important class of groups that we consider are Lie Groups [70, 58, 35]. Lie groups are simply groups that are also smooth manifolds.

We define a proper action [58, Chapter 21] as,

Definition of a Proper action

Definition A.4.4 (Proper action)**.** *Given a Lie group $\mathcal{G}$ and a manifold $\mathcal{M}$, with a (left) action ρ, the action is called a proper action if,*

$$(g, p) \mapsto (\rho(g, p), p)$$

is proper.

Note that this is different, and weaker to requiring that the action itself is a proper map.

Appendix B

Miscellaneous Definitions and Proofs

B.1 Intersection of Subsystems

Definition of a Intersection of subsystems

Definition B.1.1 (Intersection of subsystems)**.** *Suppose we are given subsystems* $\widetilde{\mathfrak{z}}_1 = (\mathfrak{R}_{\widetilde{\mathfrak{z}}_1}, \mathfrak{S}_{\widetilde{\mathfrak{z}}_1})$ *and* $\widetilde{\mathfrak{z}}_2 = (\mathfrak{R}_{\widetilde{\mathfrak{z}}_2}, \mathfrak{S}_{\widetilde{\mathfrak{z}}_2})$ *of a system* $\mathfrak{z} = (\mathfrak{R}, \mathfrak{S})$*. Furthermore,* $\forall(U_1 \in \mathfrak{R}_{\widetilde{\mathfrak{z}}_1}) \subseteq (U \in \mathfrak{R})$*, and* $\forall(U_2 \in \mathfrak{R}_{\widetilde{\mathfrak{z}}_2}) \subseteq (U \in \mathfrak{R})$*,*

$$U_1 \cap U_2 \neq \varnothing,$$

Then,

$$\widetilde{\mathfrak{z}}_1 \cap \widetilde{\mathfrak{z}}_2 = (\mathfrak{R}_{\widetilde{\mathfrak{z}}_1 \cap \widetilde{\mathfrak{z}}_2}, \mathfrak{S}_{\widetilde{\mathfrak{z}}_1 \cap \widetilde{\mathfrak{z}}_2}),$$

where,

$$\mathfrak{R}_{\widetilde{\mathfrak{z}}_1 \cap \widetilde{\mathfrak{z}}_2} = \{U_1 \cap U_2 \mid \forall(U_1 \in \mathfrak{R}_{\widetilde{\mathfrak{z}}_1}) \subseteq (U \in \mathfrak{R})\}, \forall(U_2 \in \mathfrak{R}_{\widetilde{\mathfrak{z}}_2}) \subseteq (U \in \mathfrak{R})\},$$
$$\mathfrak{S}_{\widetilde{\mathfrak{z}}_1 \cap \widetilde{\mathfrak{z}}_2} = \mathfrak{S}|_{\mathfrak{R}_{\widetilde{\mathfrak{z}}_1 \cap \widetilde{\mathfrak{z}}_2}}.$$

The intersection of subsystems is defined only when the intersection of their relata do not result in a nullset. This then ensure that the following is true.

Intersection of subsystems is a subsystem

Theorem B.1.1. *If we are given subsystems* $\widetilde{\mathfrak{z}}_1$ *and* $\widetilde{\mathfrak{z}}_2$ *of a system* $\mathfrak{z}$*. Then* $\widetilde{\mathfrak{z}}_1 \cap \widetilde{\mathfrak{z}}_2$ *is also a subsystem of* $\mathfrak{z}$*.*

Proof. The proof follows by inspection, as in Theorem 2.1.1. ■

This page was left intentionally blank.

Appendix C

Hyperparameters

C.1 Cartpole

The policies for these problems were composed of neural networks with H hidden units in a single hidden layer. The non-linear component of the hidden layer was the TANH function; the final output passed through a SOFTMAX layer. The input for these networks were the concatenated vector (g_t, s_t, z_t). For the MTRL case, we preprocessed the input by computing the outer product between state vector s and the concatenation of the latent variables z and g. We then flattened the outer product and concatenated the original input vector once more.

The hyperparameters for the single-embedding and independent algorithms are the same as the MTRL values from Table C.1. The dimension of the single embedding was equal to the sum of the dimensions of the reward and dynamics latent variables.

C.2 Distral

The hyperparameters for Distral were chosen as $\beta = 10$, $\alpha = 0.5$ and all learning rates were 10^{-4}, whereas the network architecture consisted of two layers with 50 hidden neurons and ReLU non-linearity.

Table C.1. Cartpole hyperparameters

Parameter	MTRL	HRL
dim z	2	-
dim g	2	-
γ	0.99	0.99
H	2	100
α_d	50000	-
α_r	1000	-
α_π	∞	50
π learning rate	0.002	0.002
q_δ, q_ω learning rates	0.002	-
β_{art}	0.02	0.02
Max episode length	300	2000
Number of tasks	9	1
Batch size per task	4	10
Extended policy input	Concat and outer product	-

List of Figures

This page was left intentionally blank.

List of Definitions

This page was left intentionally blank.

List of Theorems, Lemmas and Corollaries

This page was left intentionally blank.

List of Propositions and Conjectures