LEARNING REPRESENTATIONS USING REINFORCEMENT LEARNING

by

SOURABH BOSE

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2019

To my Mom & Dad

ACKNOWLEDGEMENTS

First and foremost I would like to express my sincere gratitude to my supervisor Dr. Manfred Huber. Without his patience and continuous effort, this would not have been possible. I have learned a lot from him, and I am extremely lucky to have him as my mentor. Furthermore, I'd like to thank my thesis committee members, Dr. Gergely Záruba, Dr. Farhad Kamangar, Dr. Vassilis Athitsos and Dr. Heng Huang for their insightful discussions and valuable guidance over the years.

I would also like to thank Dr. Bob Weems for the wonderful years during which I worked as his teaching assistant. Finally, I would like to thank my family members for their love and support, and my labmates in Learning and Adaptive Robotics (LEARN) lab, who have become a part of my extended family over the years. Thank you.

April 22, 2019

ABSTRACT

LEARNING REPRESENTATIONS USING REINFORCEMENT LEARNING

SOURABH BOSE, Ph.D.

The University of Texas at Arlington, 2019

Supervising Professor: Manfred Huber

The framework of reinforcement learning is a powerful suite of algorithms that can learn generalized solutions to complex decision making problems. However, the applications of reinforcement learning algorithms to traditional machine learning problems such as clustering, classification and representation learning, have rarely been explored. With the advent of large amounts of data, robust models are required which can extract meaningful representations from the data that can potentially be applied to new unseen tasks. The presented work investigates the applications of reinforcement learning algorithms in the perspective of transfer learning by applying algorithms in the framework of reinforcement learning to address a variety of machine learning problems in order to learn concise abstractions useful for transfer.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

Introduction

Reinforcement learning allows extracting suitable priors from many general optimization problems. The presented work focuses on traditional machine learning problems, classification, clustering and representation learning etc., and applies algorithms from the framework of reinforcement learning in order to extract structured information suitable for transfer in such problems.

Artificial neural networks have been extensively used to solve many machine learning problems. Recent breakthroughs in deep learning techniques [1] allow neural networks to be extended to various new application domains. However, many real world problems do not possess the complete information required by the standard framework for training such networks. For example, in the case of lifelong learning systems, many algorithms act in dynamic environments where data flows continuously as streaming data. Such learning problems exhibit persistent changes in task requirements. Many classification problems also face such issues, often termed as a Concept Drift [2][3] of the task distribution. Concept drift refers to the change in relationships between input and output data in the underlying problem over time. Samples from such systems are often not from a strictly stationary distribution. Learned models for such problems thus become obsolete over time. To address this issue, reinforcement learning algorithms are used to learn representations that can adapt to the changing task specifications.

With the advent of large amounts of data, labeled data is usually expensive and sometimes requires human domain experts to annotate, while partially labeled data

is often relatively easier to obtain. Many real world problems are poorly labeled, often with sparse and partial supervision. Such issues are often termed as Weak supervision problems and are often faced in constrained clustering problems [4] or classification problems with partial labeling information [5]. In such cases, robust representations are required which can capture the underlying structural dependencies of the task. Learning systems in such situations must be able to learn structured representations defined by imposing external constraints when available. Learning structured representations also allows encoding the forward dynamics of a system. Many real world tasks can share an underlying model that is common across tasks in the same domain. In such cases discovering abstractions which define the underlying structural dynamics is beneficial as it adds to the expressive power of the model.

Applying neural networks to such problems requires dealing with sparse feedback and being able to learn representations that are robust with respect to tasks in dynamic environments. This dissertation develops algorithms to address these requirements by combining reinforcement learning algorithms with artificial neural networks and applies them to various machine learning problems. Here, reinforcement learning algorithms are used to control various facets of neural network training and their design, while focusing on hybrid learning techniques where elements from the reinforcement learning framework are combined with artificial neural networks for learning representations suitable for transfer.

1.1 Previous Work

A variety of approaches have been developed to address different aspects of the issues discussed here. Problems with sparse structural constraints such as constrained clustering problems often require forming a differentiable model for the constraints which includes clustering quality, leading to a complex overall application with the

need for a modified clustering algorithm[6, 7, 8, 9]. Such algorithms, however, are specific to the type of constraint. In such cases, introducing a new type of constraint requires a reformulation of the learning algorithm.

Ensemble learning algorithms are often designed for problems with large datasets with concept drift properties. Such algorithms often maintain a fixed size of the ensemble or use evolutionary algorithms to build ensemble networks [2, 10, 11]. This requires randomly re-exploring valid architectures from scratch for any new problem, which greatly affects the applicability of such algorithms.

Integrating structural information is another issue that often requires hand engineered context appended to the training data. Many constraints in the real world are often non-differentiable or discontinuous functions. Solving such problems often involve hill climbing algorithms, or versions of simulated annealing. SMO algorithms have also been used to solve such problems where the solution is optimized by considering one variable at a time[12]. Such solutions are often unstable and require carefully hand engineered hyper parameters[13]. Various neural network architectures have been proposed for discovering hidden structures in data.

Apart from correlation among dimensions, the observed data might also be spatially correlated. Learning effective representations for sequential data is an active area of research. In such cases the structures are often modeled outside the network architecture, using hidden Markov models or graph based modeling algorithms which encode the temporal structure of the data [14]. These issues indicate that uncertainty in real world data is an important challenge in various machine learning problems. A more detailed review of existing methodologies for the individual problem domains are presented in the corresponding chapters of this dissertation.

Neural networks and Reinforcement learning algorithms have been combined to form a powerful learning tool that can model various uncertainties in the observed

data. Such algorithms are often restricted to the domain of sequential decision making algorithms. Neural networks are used to learn complex representations from continuous high dimensional datasets, while reinforcement learning algorithms are used to model sparse feedback and solve problems in dynamic environments. This work examines various benefits of applying reinforcement learning techniques to other problem domains where issues of uncertainty arise. The remainder of this section provides a brief overview of artificial neural networks and reinforcement learning algorithms. Following this, a brief review of algorithms combining neural networks and reinforcement learning approaches is given.

1.1.1 Neural Networks

Artificial neural networks are a powerful modeling technique that can be used for representation learning in complex high dimensional input data. Recent breakthroughs in multi-layered neural networks or Deep learning mechanisms [1] illustrate the effectiveness of such algorithms at discovering intricate structures in high-dimensional data extending their application domains to many real world problems. Deep learning techniques have made major advances in solving complex supervised learning problems of science, business and government, ranging from image recognition [15], speech recognition and language modeling [16] [17], predicting the behavior of new drug molecules [18], and modeling biological systems as in reconstructing brain circuits [19] and predicting effects of gene mutations [20] [21]. Neural networks are capable of modeling arbitrarily complex functions in high dimensional continuous spaces and have been shown to generalize well for unseen data far from the training input. Furthermore, recent advances in deep learning demonstrate the complex representation learning capabilities of multi-layered architectures where a typical deep-learning

system, might have hundreds of millions of these adjustable weights, and hundreds of millions of labeled examples for training.

However, neural networks assume some preconditions over the type and distributions of training data allowed. In particular, the training data provided should be Independent and Identically Distributed (IID) with a fixed underlying distribution, and requires complete supervision in the form of ground truth or complete gradients with respect to individual instances in the data. Such requirements are often infeasible for many learning problems and thus restrict the application of neural networks to various problem domains where such uncertainties might arise in the observed real world data. Furthermore, neural network models have no inherent preference for discovering underlying structural information of the data which might prove beneficial and add to the expressive power of the learned representations. This is an open area of research where various architectural constraints have been proposed on the hidden representations in the form of lateral or recurrent connections among nodes. However, neural networks trained with traditional backpropagation algorithms and its variants which provide some convergence guarantees for standard feed-forward networks, often fail to converge when constrained by such recurrent connections.

1.1.2 Reinforcement Learning

The theory of sequential decision problems includes formulations of both deterministic and stochastic problems [22]. In such problems an agent interacts with a discrete time stochastic dynamical system, by observing the current system state and selecting an action at each time step. Sequential decision problems are popularly represented by the mathematical framework of Markov Decision Processes (MDP) in stochastic domains [23]. A *Markov Decision Process* is defined as a tuple $\langle S, A, \Psi, P, R \rangle$, where S is a finite set of states, A is a finite set of actions, $\Psi \subseteq S \times A$

is the set of admissible state-action pairs. $A_s = a|(s,a) \in \Psi \subseteq A$ defines the set of actions admissible in state s, assuming that $\forall s \in S$, A_s is non-empty. $P : \Psi \times S \mapsto [0,1]$ is the transition probability function with $P(s,a,s')$ being the probability of transition from state s to state s' under action a where $s, s' \in S$ and $a \in A_s$. $R : \Psi \mapsto \mathbb{R}$ is the expected reward function, with $R(s,a)$ being the expected reward for performing action a in state s.

Decision making problems are a class of *Credit Assignment Problems* and are often found in the real world. Solving such problems requires learning from sparse delayed feedback in stochastic dynamical environments. Reinforcement learning [24], [25] is a learning paradigm that acquires control policies without the need for extensive outside supervision. This formulates a framework for reactive control which learns from interactions with the environment utilizing supervision provided in the form of simple, scalar rewards and punishments defined by the learning task. Reinforcement learning algorithms use trial and error exploration to search for solutions to the credit assignment problem by approximately optimizing the expected reward of the system.

Most reinforcement learning algorithms can be categorized into offline and online learning algorithms. The Q-learning algorithm [26] is an offline model-free reinforcement learning approach and is a form of asynchronous Monte-Carlo dynamic programming. Here, the utility of a state action pair depends on the current observed reward and the utility of the greedy action choice for the next observed state. Given an state transition tuple $\langle s_t, a_t, s_{t+1}, r_t \rangle$, the Q-learning update step can be defined as :

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha(r_t + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t))$$

Offline Reinforcement learning approaches have the benefit that they can learn from existing data without the need to explicitly execute the policy to be learned. On the

other hand online learning algorithms have several advantages over offline learning and are potentially more robust to errors or omissions in the training set [27].

Instead of selecting the greedy action for the utility, online algorithms choose the next state action pair according to the current policy which allows online learning systems without requiring the storage of previously observed state action utilities. The SARSA algorithm [28], [29] is a modified connectionist Q-learning algorithm, where the update step for a given experience tuple $\langle s_t, a_t, r_t, s_{t+1}, a_{t+1} \rangle$ is defined as:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha(r_t + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t))$$

Additionally, reinforcement learning algorithms allow parameterized policies by applying Actor Critic techniques as policy gradient algorithms [30]. Here, a policy function is used to model the strategy and updated by estimating a gradient which optimize the utility. Allowing step updates and assuming an uncertain problem domain allow for robust learning algorithms that are suitable for solving many decision making problems in the real world.

Although rare, reinforcement learning algorithms have also been applied to problems in other learning domains. For example, in clustering problems [31] where the clustering problem is formulated as a reinforcement learning problem utilizing the quality of the cluster as feedback.

Like many learning algorithms, reinforcement learning is also susceptible to the curse of dimensionality while working on many real world problems that are usually represented in high dimensional and potentially continuous domains. The reinforcement learning framework defines various methods for addressing such issues. For problems with continuous spaces, the states are often aggregated to form an approximate factored representation of the space [32][29] [33]. In such cases the representations used need to respect the underlying dynamical system. Apart from

discretized state spaces, a latent representation is often learned for the observed system. Neural networks are most commonly combined with reinforcement learning algorithms to allow decision making and perform value function approximations in various real world problems with high dimensional input representations.

1.1.3 Combining Reinforcement Learning Algorithms and Neural Networks

Most existing research combines neural networks with reinforcement learning algorithms to address issues in the the domain of sequential decision making problems. Artificial neural networks are used to address the issue of space complexity in high dimensional, continuous space environments. Recently Reinforcement learning algorithms have been combined with deep neural networks to solve problems in the domain of sequential decision making problems. Deep Reinforcement learning algorithms (DEEPRL) [34] have been used to solve many tasks in complex and uncertain environments. Neural networks are used for function approximation, modeling the estimated utility of a sample. Deep Q networks (DQN) algorithms which combine deep learning models with reinforcement learning have been used for increasingly complex decision problems [35]. Furthermore, artificial neural networks are compatible with policy gradient algorithms which allow estimating gradients for learning a policy. The DDPG algorithm [36] applies policy gradient algorithms on deterministic deep networks. Training a critic network which models the feedback, it estimates the gradients for an actor network which maximizes the feedback.

Reinforcement learning algorithms solve for credit assignment problems with delayed feedback in uncertain environments. Using neural networks allows solving complex sequential decision making tasks with high dimensions or in continuous spaces. Thus the combination of Neural networks and Reinforcement learning algorithms forms a powerful learning technique.

Apart from traditional decision problems, the combination of neural networks and reinforcement learning algorithms have rarely been used to address issues arising in other learning problems. Recently, attention based mechanisms have performed extremely well [37] which applies reinforcement learning algorithms to manipulate the representations learned by a neural network. Here, the REINFORCE algorithm [38] has been used to decide the focus of attention over frames in sequential data even though the underlying model is non-differentiable. The agent actively controls the locations for deploying its convolutional sensor resources and thus influencing the representations used to reconstruct the data. The extracted attention information is potentially useful for other tasks in this domain. The algorithms proposed in this dissertation explore similar applications of reinforcement learning paradigms for feature learning problems focusing on various forms of uncertainty observed in samples from real world data. Here, reinforcement learning algorithms are used to impose control over the learned representation by utilizing some feedback provided by the task definition.

1.1.4 Summary and Contributions

In order to apply machine learning algorithms to various problem domains, the challenges posed by uncertainties in real world problems must be addressed. Such uncertainties might arise from changing task distributions over time, availability of sparse supervision and from underlying structural correlations which can introduce bias and instability in learning algorithms. These issues are inherent in the data provided, and thus can affect many learning problems with real world data. Addressing such issues requires robust learning techniques which can react to sparse feedback and underlying dynamics of the data.

This work combines reinforcement learning algorithms with artificial neural networks to solve for such issues. Here, reinforcement learning is used to control the learning and design aspects of neural networks in machine learning problems such as classification, clustering and representation learning. This addresses many important issues of learning problems with uncertain real world data and contributes techniques to the individual research areas.

For classification problems with large datasets exhibiting concept drift, an adaptive ensemble classifier is proposed. Here, Reinforcement learning algorithms are used to control the architecture of the classifier network, utilizing the solution accuracy as feedback. This allows adapting the learned representations to task distributions that change over time. In case of sparse structural supervision in constrained clustering problems, reinforcement learning algorithms are used to learn a policy that can automatically map the observed space to different representations with respect to different constraints. Furthermore, in order to discover underlying inter-dimensional correlations in the data, a novel sparse structured coding is proposed. Here, Policy gradient algorithms are used in order to impose the L0 norm which is a non-differentiable, discontinuous constraint objective and learn a structurally sparse, hierarchically related representation of the input data.

Apart from that, an adversarial network is designed from a model minimization framework in reinforcement learning. Here, the proposed architecture learns a representation which models the underlying dynamics of the observed sequential data.

1.2 Document Overview

In the remainder of this document algorithms are developed which address the challenges posed by uncertain data in various learning problems. In particular, issues arising from partial supervision and inherent dynamics of data in clustering, classifica-

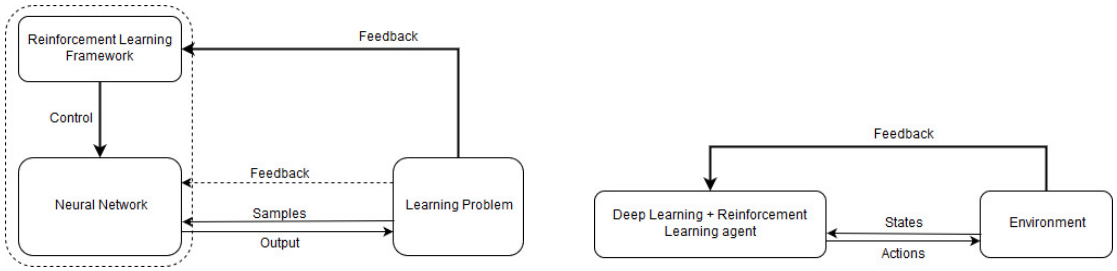
tion, and representation learning problems are explored. Chapter 2 gives an overview of the general learning technique used in this dissertation and discusses some of the main forms of uncertainty in real world data. Chapters 3 and 4 apply reinforcement learning algorithms to learn adaptive features. Chapter 3 is dedicated to uncertainties arising from interdependent and non-identically distributed (non-IID) input data in classification problems and illustrates the use of reinforcement learning algorithms in order to build feature sets that adapt to changing task requirements. Chapter 4 addresses the issues arising from sparse structural constraints in semi-unsupervised clustering problems. The algorithm described here demonstrates the use of Reinforcement learning techniques for constrained clustering on a given dataset by learning a policy which can automatically map the observed space to a different feature representation depending upon the constraints. Chapters 5 and 6 apply reinforcement learning algorithms in order to guide representation learning. Chapter 5 discusses some difficulties arising from partial and discontinuous supervision in classification and feature learning problems, and illustrates the use of policy gradient algorithms to address such issues. Chapter 6 formulates an MDP autoencoder architecture which applies the MDP homomorphism framework to discover underlying latent dynamics of an observed model. Finally, Chapter 7 concludes the work by summarizing the various proposed algorithms.

CHAPTER 2

Learning in Neural Networks Using Reinforcement Learning to Address Uncertain Data

This work applies elements of the reinforcement learning framework to control various aspects of representation learning and design of neural networks. This extends the application domain of neural networks and allows learning robust representations for uncertain and stochastic training data. Contrary to this, current Deep reinforcement learning techniques [35] learn value function approximations and control strategies by connecting a reinforcement learning algorithm to a deep neural network which operates directly on raw input and efficiently estimates the environment by using stochastic gradient updates. Such algorithms are effectively used to control the agent in a decision making environment by utilizing neural networks as function approximators for reinforcement learning algorithms.

Figure 2.1(b) shows a general Deep Reinforcement Learning architecture used for traditional decision problems, while Figure 2.1(a) outlines the general architecture followed by the algorithms proposed in this dissertation. This work applies reinforcement learning algorithms in order to extract abstractions in the form of meaningful priors for multiple machine learning problems. The algorithms proposed here impose control on the representations learned by a neural network to form robust representations useful for transfer and address various issues arising from uncertainty in real world problems.



(a) Learning technique used in this work (b) DeepRL architecture

Figure 2.1: (a) Outline of the learning techniques used in this work; (b) Architecture used by Deep Reinforcement learning algorithms for decision making problems.

2.1 Types of Uncertainty in Data

Reinforcement learning algorithms assume some uncertainty in the observed data and are thus extensively used in sequential decision making problems. Solving such problems requires robust algorithms that can adapt to noise and uncertainties in the real world.

Other learning algorithms also face similar uncertainty issues when applied to real world data. For example, the availability of partial labeling or imbalance in datasets are important areas of research in classification algorithms. This work focuses on three main sources of such uncertainties in data, and proposes various algorithms which address these issues.

2.1.1 Interdependent and Non-Identically Distributed Input Data

Most Neural Network algorithms assume that the given input data are samples from independent and identically distributed (IID) random variables. Such assumptions imply that the observed data sample does not depend on previous samples, do not exhibit drifting temporal influence, and are realized from a balanced distribution with respect to the task. Focusing on classification problems, there are many cases in the real world where this assumption is often violated. It is often restrictive to

gather data that can represent the task perfectly. Many classification problems favor imbalanced class data, where the training set representation is highly skewed towards some classes. Such issues are abundant in problems where gathering a balanced distribution for all classes is often not feasible. Furthermore, with the advent of larger datasets, real world data is often only available as streaming data, leading to the need for sampling based algorithms. However, such samples are often temporally or spatially correlated. In the case of streaming data, the task information might change as data arrives over time, resulting in a concept drift. Such class distributions result in poor performances when trained on standard neural network classification algorithms.

This work proposes an approach that uses reinforcement learning to learn a policy which can control the architecture of a neural network for any classification problem. Here, Reinforcement learning algorithms are used to decide the architecture of the network interactively by utilizing classification accuracy as feedback. This allows the network to learn features that can adapt to changing task requirements. It utilizes a sampling based approach and builds the learning system incrementally. Furthermore, it can decide to selectively enable or disable parts of the network and allows learning targeted representations for specific classes. This results in a robust neural network architecture that takes the current quality of the solution in consideration and adapts itself to data imperfections and distribution changes.

2.1.2 Imperfect Supervision

Training most learning algorithms requires complete supervision from the system. However, only sparse supervision is available for many real world problems. Sparse supervision problems are characterized as a special case of supervised learning problems where some samples are provided with extra information regarding their

representations, which is not available for the other samples. In the case of constrained clustering problems, in addition to the unsupervised distance metric, a few instance level constraints are provided which enforce cluster memberships of a pair of instances. These sparse constraints indicate whether two samples should be grouped to the same cluster or different clusters. Many learning algorithms including traditional clustering algorithms are ill-equipped to solve for such sparse supervision problems. Imposing such constraints often requires a reformulation of the learning algorithm which might not always be possible. This work proposes an approach which learns a representation to indirectly impose such constraints on a traditional K-means clustering algorithm. Reinforcement learning is used to model the feedback from the clustering algorithm and learn a policy which morphs the input representation with respect to the constraints. Here, the clustering algorithm acts as the environment, while the agent designs the input representation which optimizes constraint satisfaction as feedback. Using the clusters formed by the stochastic K-means algorithm, it learns an input representation which results in clusters which satisfy the constraints. This method allows incorporating sparse constraints directly into a learned input representation, resulting in a framework where modification of the base clustering algorithm is not necessary.

In addition to sparse supervision where only a few samples get the complete supervision, some real world problems only provide incomplete supervision for all samples. Training artificial neural networks for classification problems requires the target class in order to compute the gradient with respect to the output class. However, many problems can only provide a feedback of whether the current solution is correct or incorrect, without any guidance towards the true target. Such problems are often termed lock and key prediction problems [5]. In such cases, the true target is unknown and thus the feedback has no gradient. In this work, a classification

problem is designed which provides supervision in the form of a boolean feedback of either correct or incorrect. The solution for this is trivial for binary classification problems. However, in case of more than two classes, it becomes a credit assignment problem. Reinforcement learning algorithms allow modeling such feedback from the system and solve for such problems. Policy gradient algorithms are used to model the supervision. When combined with classification problems, such algorithms can estimate the gradients for the provided incomplete feedback with respect to each class output.

2.1.3 Underlying Structural Correlations

Neural networks can learn useful latent representations from input data. Such representations are often accomplished by learning a compressed representation of the input. However, such representations fail to capture inherent structure which is often present in real world data. For example, in case of reconstruction problems, it is beneficial to learn representations which encode correlations between the input features. In such cases a sparsity constraint is applied as a regularization term, such as KL-divergence based sparsity constraints, in order to discover some of those correlations. However, it is sometimes appropriate to enforce more structure on the latent representation than just sparsity. Structured sparse models encode information about the dependencies between code words. Enforcing such constraints allows to control the expressive power of the model without losing reconstruction accuracy.

Structured sparse coding for reconstruction tasks learns representations where a feature is influenced by activations of sibling features in the hidden layer. Imposing such constraints, usually termed structural constraints, requires lateral connections among the nodes in the layer. This results in a complicated, recurrent network architecture requiring gradient updates for a node to be affected by sibling nodes

connected in a distributed fashion. Such architectures increase the computational overhead and are unstable, often failing to converge to a solution. This shows the need for algorithms that can enforce such constraints without altering the architecture of the network. This work proposes a form of structured sparsity influenced by the L0 norm. The L0 norm is defined as the number of non-zero elements in a given vector. Unfortunately the L0 norm constraint is a discontinuous function which does not have a gradient anywhere. However, minimizing this constraint results in a sparse coding which requires as few non zero activations as possible in order to reconstruct a sample. This constraint thus emulates sparsity in a fully interconnected layer with lateral inhibitions. Here, Reinforcement learning is applied to influence the representation learning process in a neural network. This algorithm models the L0 norm as a penalty term, and estimates gradients in order to impose the constraint as a regularization term.

Apart from correlations between dimensions, structure in data can also arise in the form of temporal correlations between samples. In such cases the current observed sample depends upon the previously observed samples. In such cases, an underlying dynamics is assumed that can explain the observed data. Structured sequential data is often present in graph based problems such as learning representations for social networks, Markov decision making problems and natural language modeling problems. Reinforcement learning algorithms provide frameworks for minimizing such models into a latent or factored representation which respects the underlying dynamics of the sampled data. Here, an MDP autoencoder is formulated using the MDP homomorphism framework, in order to learn latent MDP models encoding the dynamics of the model with respect to temporal structural dependencies directly from observed samples.

CHAPTER 3

Incremental Learning of Neural Network Classifiers Using Reinforcement Learning

3.1 Introduction

With the advent of larger amounts of data in the presence of limited computational resources there is a significant need for sampling based algorithms, and with it arises the need for classification algorithms capable of incremental training. However, traditional algorithms, which are not designed around sampling, usually fail to scale well to large data sets and are often not suited to situations where only limited samples of the overall dataset are available at any point in time. The latter arises, for example, in the case of streaming data where data arises over time and the accumulation of a large data set a priori is infeasible either due to limitations in memory or computation resources, or due to privacy and data ownership limitations which make it impossible or inadvisable to store larger amounts of data over extended periods. Training a classifier with such a dataset, often requires incremental training and thus necessitates modifications of the existing structure or parameters of the classifiers to adapt to the changes which arise from concurrent samples, which are usually not possible for traditional classifiers. In such cases, Ensemble algorithms like boosting methods with resampling [39] and bagging methods like bagging predictors [40] are usually better suited as samples differing vastly from the learned distribution may be handled by different, weaker classifiers. However, training weak classifiers from samples from a larger dataset often results in an unnecessarily large ensemble leading to computational costs, and usually decrease in effectiveness [41]. One way to resolve

this problem is to choose selective classifiers as members of the final ensemble. This, however, results in a subset selection problem which is a combinatorial optimization problem[41].

Apart from this, additional complexities arise in ensemble classifiers if the member classifiers are neural networks, which have extra variables defining the structure of the network. This chapter presents an approach to incrementally learn neural network ensemble classifiers using a Reinforcement Learning based policy [42]. It shows that incrementally building classifiers from a small fraction of the overall dataset using this method achieves comparable performance to kernel SVM while achieving better results when compared to standard neural network classifiers.

3.2 Existing methodologies

Existing methods to solve the problem of ensemble classifiers for large scale datasets or streaming data usually involve a variation of online boosting algorithms [10], training individual member classifiers with randomly sampled data and creating a mega-ensemble of random forests [43], or integrating the collective results through meta-learning [44]. In order to avoid a large ensemble [2] maintains a fixed size ensemble, heuristically computing memberships.

Multiple techniques have been used to address the problem of ensemble memberships with neural network member classifiers, include genetic algorithms to evolve weights which are used as a metric for ensemble membership [11], tuning the member classifiers, making them as accurate and diverse as possible [45]. Finally, methods to dynamically modify the structure of the classifier involve recursively adding nodes as needed [46] retraining the entire network or learning new features iteratively similar to a decision tree.

3.3 Approach

The proposed approach is divided into three Markov Decision Problems (MDPs). These MDPs, upon training, will learn policies which, when applied to new classification problems build classifiers for them. Given the state of the system defined by its state attributes, the policy learned for the Ensemble Learning MDP iteratively decides on the membership of networks in the ensemble for the problem by adding or pruning networks from the final system. Adding a new network to the system involves creating a new network, a task which is delegated to the Type Selection MDP policy which determines the type of network to build and then hands construction of a new network to the Network Creation MDP policy. Given the current state of the classification process, the policy learned by the Type selection MDP decides upon the type of member classifier to build, which is either a One-vs-Rest or an All-vs-All classifier. It is often necessary to be able to choose to build a One-vs-Rest [47] classifier instead of an All-vs-All if the current system classifies some classes more poorly than others. This problem usually arises if the training data is highly imbalanced, resulting in lower preference of classifying an under-represented class compare to an over-represented class in the domain. Such problems require special attention, because the goal of general classification algorithms is to reduce the error function, which is representative of the error of the dominant class. Upon deciding the type of network to be created, the task of building the network is delegated to the policy for the Network Creation MDP. The policy learned for the Network Creation MDP builds the required network with a small sample of the training set, which may arise from sampling a large dataset or from streaming datasets. It iteratively decides to either increase the number of nodes in the current layer or increase the number of layers in the network. Training the network with a small sample provides the necessary variance for a member of an ensemble of networks. The membership utility of

a network thus created is then evaluated by the Ensemble Creation MDP policy for the final ensemble.

Subtasks defined over other MDPs behave as modules where each MDP policy solves a part of the whole problem [48]. Subtasks in this system are flattened, which means that the utility of the overall policy propagates through all the sub-MDPs, thus the policies learned by the sub-MDPs achieve a global optimum instead of greedily solving for the local maxima. In this approach, the Type Selection MDP is a subtask of the Ensemble Learning MDP, and the Network Creation MDP is a subtask of the Type Selection MDP. Given a state, when the Ensemble Learning MDP decides to add a new network, the next state is the starting state of the Type Selection MDP. Similarly in case of the Type Selection MDP, given the current state, upon deciding the type of network required, the next state is the starting state of the Network Creation MDP. The final state of the Network Creation MDP transitions to the Ensemble Learning MDP state. This ensures that the MDPs cooperate to learn a policy that is globally optimal since the utility of the entire ensemble is propagated through the sub-MDPs.

3.3.1 Ensemble Learning MDP

Given a classification problem, the policy learned by this MDP builds the final neural network ensemble classifier. The *current set* maintains the set of networks currently considered as members of the $final\ set$, while a *standby set* of networks is maintained, which consists of networks that were created, but are not members of the *current set*. Upon reaching the end of the policy the *current set* of networks is considered as the $final\ set$.

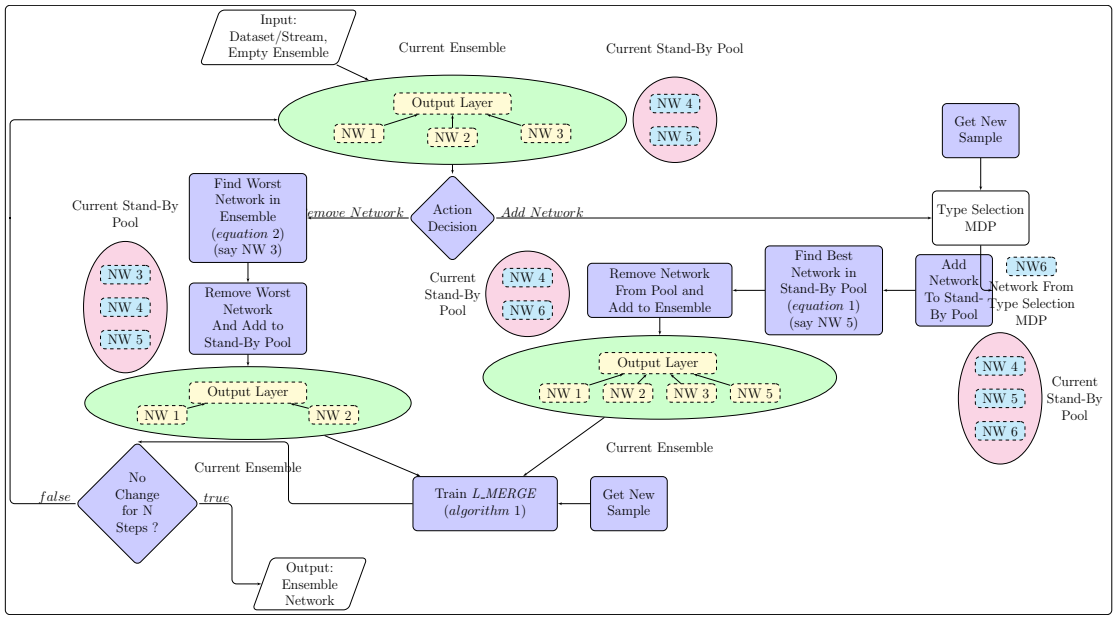


Figure 3.1: Ensemble Learning MDP Workflow

3.3.1.1 State Space of the MDP

The state space is defined by two state attributes, $Accuracy_{ensemble}$ and $Quality_{ensemble}$. To compute the state attributes of the current ensemble configuration, the networks in the *current set* are merged in a new output layer $Layer_{merge}$, which is trained with respect to a sampled training set and the accuracy, $Accuracy_{ensemble}$ is computed from the validation set.

The weights learned upon training the $Layer_{merge}$ layer are used as a measure for the utility of each network in the *current set*, along with the quality of the current ensemble. If the average of the set of weights in $Layer_{merge}$ for $network_i$ is $meanw_{network_i}$, the quality of the current ensemble $Quality_{ensemble}$ is computed as

$$Quality_{ensemble} = \frac{min_{network_i \in current\ set}(meanw_{network_i})}{max_{network_i \in current\ set}(meanw_{network_i})}$$

The policy learned with this method maximizes the accuracy of the final solution, while pruning out networks which do not contribute much to the final solution. This

$Quality_{ensemble}$ and the $Accuracy_{ensemble}$ are both in the range of [0..1]. The attributes are discretized into bins in order to reduce the complexity since state values closer to each other are related and thus can be grouped into a single state.

3.3.1.2 Action Space of the MDP

In order to modify the current configuration of network memberships, the agent can take two actions which add a new network or remove an existing network, respectively. Adding a new network to the system involves creating a new network using the policies learned for the Type Selection MDP and the Network Creation MDP with a small set of samples *N'* of the overall set of samples *N*, and including it in the *standby set*, heuristically computing the best candidate network in the set, and finally appending the best network to the *current set*. Similarly, removing a network from the *current set* involves heuristically choosing the network which might contribute the least in the final classifier, and subsequently removing it from the *current set* and appending it to the *standby set* for possible future use.

The agent is to decide, given the current state of the system identified by the parameters $Accuracy_{ensemble}$ and $Quality_{ensemble}$, between the two actions

- Adding a network to the *current set*.
  - Create a new network, appending it to the *standby set*.
  - find the best network to add from the *standby set*.
  - add best network to the *current set*.
- Removing a network from the *current set*.
  - Find the worst network in the *current set*.
  - remove the worst network from the *current set*, appending it to the *standby set*.

After each action $Layer_{merge}$ is retrained and the new state variables are recomputed. The *Ensemble Learning* workflow is shown in Figure 3.1.

3.3.1.3 Reward Function

The reward of the system after each action is computed as the change in accuracy achieved by the ensemble of networks in the *current set* over a sampled validation set.

3.3.1.4 Termination Criteria

The terminating state is achieved if the accuracy attribute in the state space does not change over consecutive S_e actions. This ensures small variance of the outcome.

3.3.1.5 Choosing the Candidate Network for Addition or Removal

The networks in the *standby set* are heuristically assessed to identify the best candidate for inclusion in the *current set* of networks. Given a set of samples S, let S' be the samples mis-classified by the *current set*. The utility of a network in the *standby set* is computed approximately as the degree of eligibility $Eligibility_{network_i}$ of the network in the candidate space.

$$\begin{aligned} &Eligibility_{network_i}(\forall network_i \in current\ set) = \\ &\qquad Accuracy_{network_i} * \frac{\#\, correctly\ classified\ in\ S'}{\#\ samples\ in\ S'} \end{aligned} \tag{3.1}$$

For removal, the network i is assumed to contribute the least in the current ensemble system and thus is chosen as the candidate for the action of removing the worst network in the *current set*.

$$i = \underset{i}{\operatorname{argmin}}(meanw_{network_i}) \tag{3.2}$$

3.3.1.6 Training the $Layer_{merge}$ Layer

In order to merge the networks in the *current set*, the activation outputs of the last hidden layer of the member networks are merged into a single $Layer_{merge}$ layer, with a common output layer. The weights of $Layer_{merge}$ define the voting power of each network, while identifying the least contributing network. When a new network is added to the *current set*, before retraining, $Layer_{merge}$ initial weights of the existing networks are left unchanged, while the $Layer_{merge}$ weights of the new network are initialized randomly. Similarly, retraining upon removal of a network involves keeping the $Layer_{merge}$ weights of the networks in the *current set* unchanged initially. $Layer_{merge}$ is trained with a small subset N' of the entire training set of N samples. Starting from a previous point in error space This approximates online batch training of the network with effectively a bigger set of training and validation samples, while using the small sampled set, thus avoiding the subset selection [49] and voting [50] problems faced by other ensemble selection methods.

3.3.2 Type Selection MDP

Given a state, the policy learned by the Type Selection MDP, which is a subtask of the Ensemble Selection MDP, is used to decide between a One vs Rest or an All vs All network. This is a 2-armed bandit system where, given a state, it chooses between one of two actions and then delegates the rest of the work to the Network Creation MDP.

Algorithm 1 Train layer L_MERGE

$W_L \leftarrow L\_MERGE\ Weights$

$K_{new} \leftarrow Newly\ Added\ Network$

Initialize Weights:

if $NetworkRemoved == true$ **then**
    $W_L \leftarrow RandomWeights$

else if $Networkadded == true$ **then**
    $\beta \leftarrow 0.5$

    $\forall W_L \in K_{new} \leftarrow RandomWeights$

    $W_O \leftarrow \forall W_L \notin K_{new}$

    $\forall W_L \notin K_{new} \leftarrow \beta * W_O$

Get New Sample: $S \leftarrow New\ Sample\ From\ Dataset\ Or\ Stream.$

Train The Layer:

$W_L \leftarrow backprop(W_L, S)$

3.3.2.1 State Space of the MDP

For a given problem with M classes, let the individual accuracies of each class be $acc_i\ \forall i \in M$. The measure of bias towards each class C_i is computed:

$$C_i = acc_i / max_i(acc_i)$$

This represents the uniformity in classification accuracy for each class in the given problem. The degree of classification bias towards classes in the current ensemble is computed as T

$$T = min(C_i)$$

and the least accurate class

$$minC = \underset{i}{\operatorname{argmin}}(C_i) \tag{3.3}$$

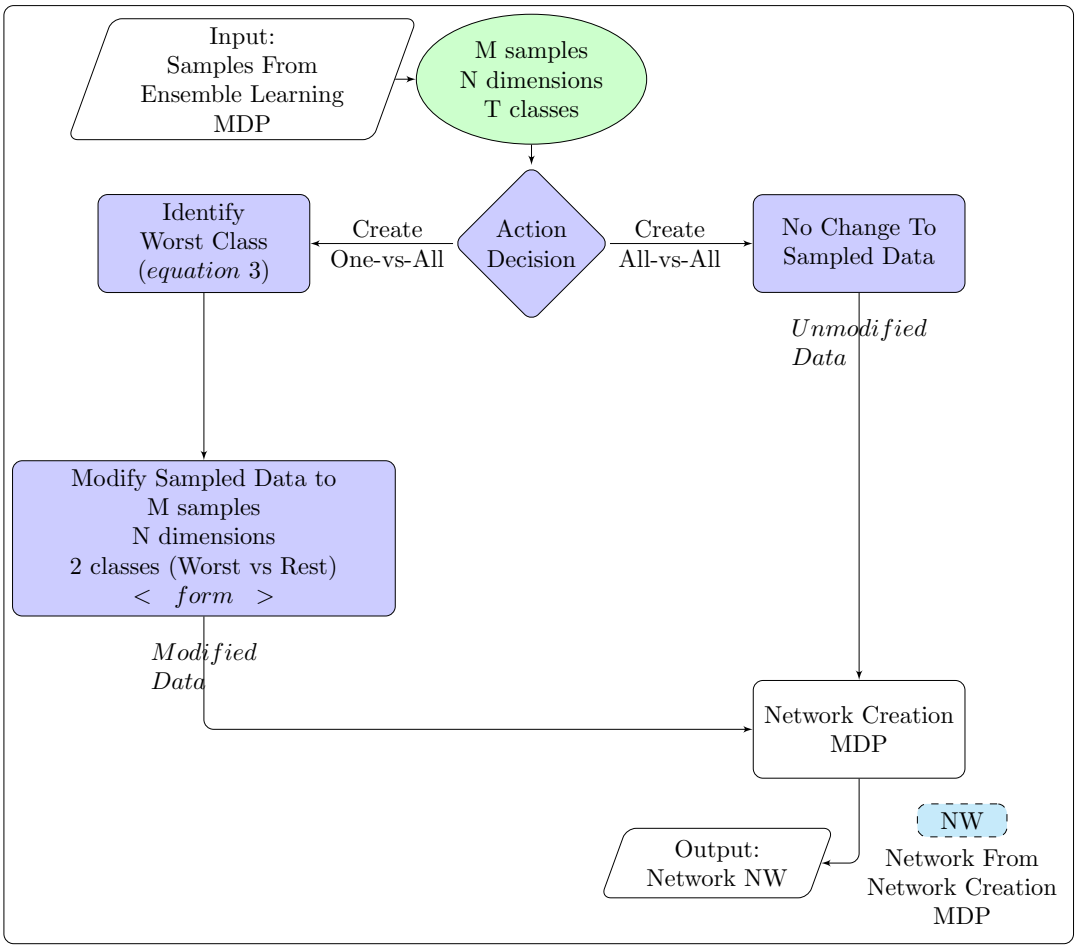


Figure 3.2: Flowchart for Type Selection MDP

T is in the range of [0..1], with value closer to 1 indicating that every class has close to uniform successful classification rate. The attribute T along with the attribute $Accuracy_{ensemble}$ from the ensemble selection MDP is used as state attributes for the Type Selection MDP.

3.3.2.2 Action Space of the MDP

Two types of networks are considered by the system, One vs Rest or an All vs All network. Given a specific class K in the problem with M classes, a One vs Rest network will give a Boolean output of class K or the rest of the classes in M

[47], while the All vs All network will classify among all classes at once, which is common in case of neural networks. In this case a One vs Rest network is a binary classifier with $minC$ vs Rest, assuming that this class needs the most improvement. Given a state, the agent is to decide between two actions, creating a One vs Rest or an All vs All network. Upon selecting the type, the Network Creation MDP policy is called as a subtask [48], which builds the network with the sampled dataset. The *Type Selection* workflow is shown in Figure 3.2.

3.3.2.3 Reward Function

There is no intrinsic reward defined in this system and the utility of an action taken is solely the reward propagated from the Ensemble Learning system.

3.3.3 Network Creation MDP

The policy learned by the Network Creation MDP, called as a subtask of the Type Selection MDP, is used to incrementally build a neural network structure given the sampled dataset from the overall problem.

3.3.3.1 State Space of the MDP

The network is built incrementally, adding nodes to the current layer or incrementing the number of layers by one as required, in an iterative manner. The measure of accuracy, given the current structure of the network being built and the sampled dataset, is computed as

$$Error_{Structure} = \frac{\#\ samples\ incorrectly\ classified}{\#\ total\ samples}$$

The error achieved by the structure in the layer previous to the current layer $Error_{Structure_{C}urrentLayer-1}$ along with the current structure error $Error_{Structure}$, de-

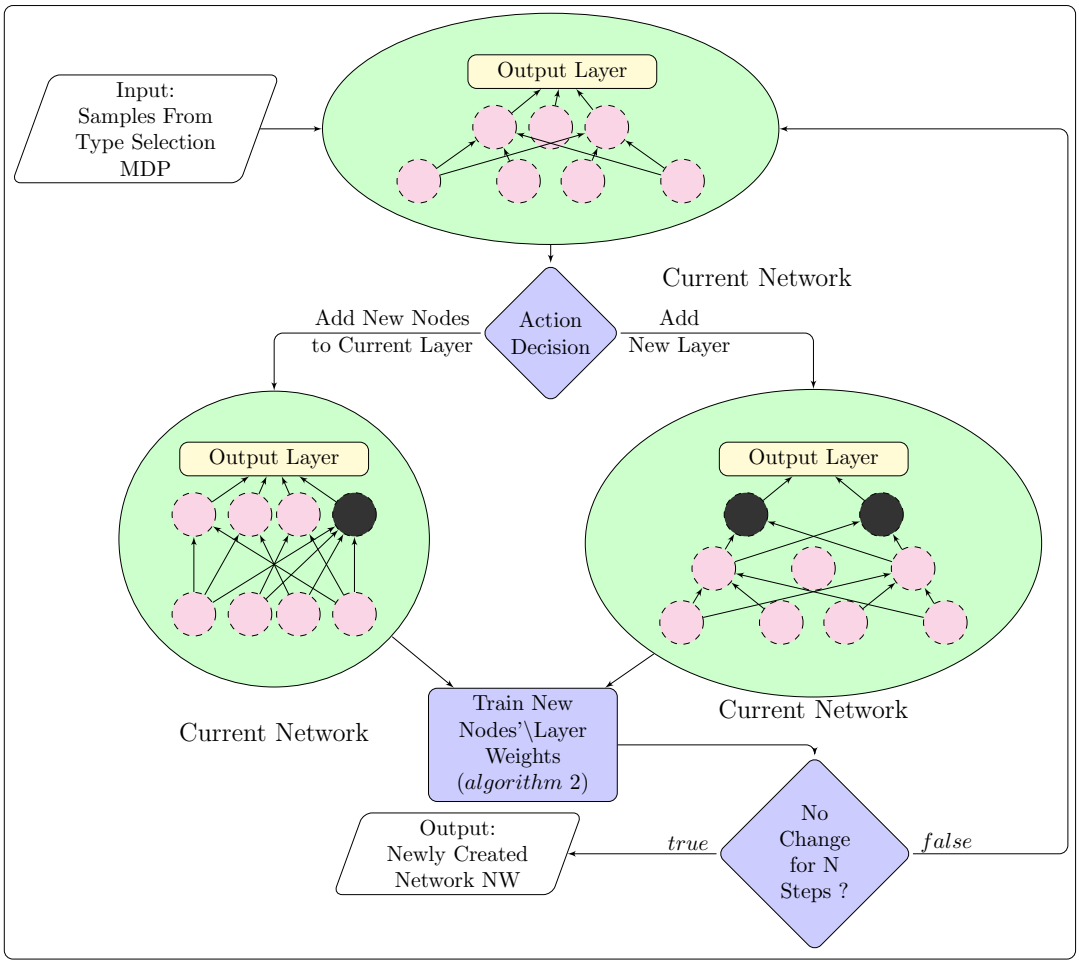


Figure 3.3: Flowchart for Network Creation MDP

fine the two state attributes of the system. $Error_{Structure_{C}urrentLayer-1}$ and $Error_{Structure}$ both are in the range of [0..1]. As before the attributes are discretized into bins in order to reduce the complexity, assuming that state values closer to each other are related. Initially, a layer with $numnodes = min(D, M)$ nodes, where D is the number of dimensions of the input and M is the number of outputs in the problem nodes, is added while $Error_{Structure_{C}urrentLayer-1}$ is being set to one.

3.3.3.2 Action Space of the MDP

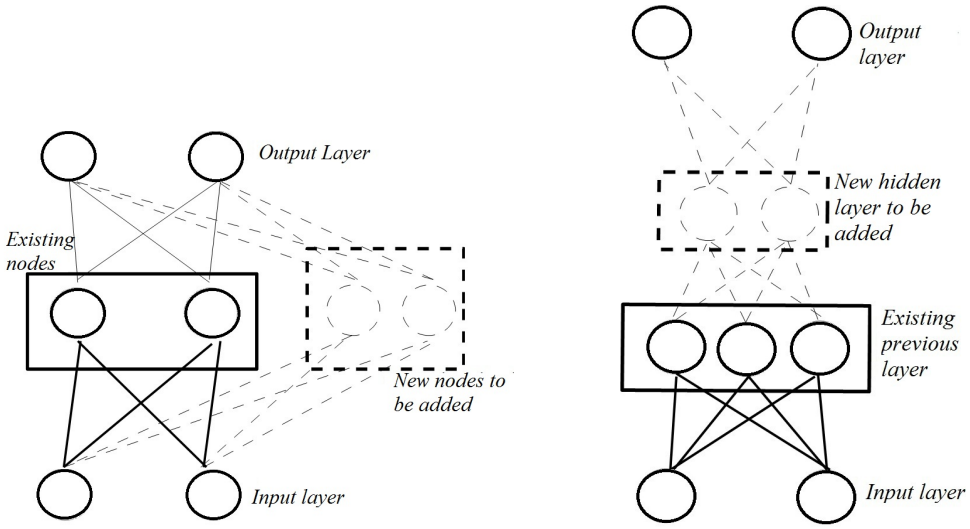
Extending the structure of the network can be achieved in different ways, one being to increase the number of nodes in the current layer by adding a set of V nodes at each time to the layer. A neural network with one hidden layer is capable of finding the solution to any given problem, however multiple layers find more complex features as combinations of features from the layers below, which are necessary to better define the problem at hand. While adding a new layer iteratively, the accuracy of the network will drop dramatically if the new layer has too few nodes compared to a previous layer with many nodes, which would in turn discourage multiple layers. Thus a new layer with H nodes with $H = min(V_{CurrentLayer}, M)$ where $V_{CurrentLayer}$ is the *number nodes in current layer* is added while incrementing the number of layers in the structure by one. Thus given the state of the system the agent can choose between two actions,

- Adding a new set of V nodes to the current layer.
- Adding a new layer with H nodes.

Figure 3.4 shows the basic mechanics of the two available actions on a simple example network. The overall $Network\ Creation$ workflow is shown in Figure 3.3.

3.3.3.3 Reward Function

The reward of the system after each action is computed as the change in error $Error_{Structure}$ achieved taking the action.



(a) Adding a new set of nodes to the current layer

(b) Adding a new hidden layer on top of the current layer

Figure 3.4: Modifying the network, bold lines indicate weights which are fixed and not relearned, fine lines denote the weights which are initialized to previous values and then relearned while dashed lines denote randomly initialized weights to be relearned

3.3.3.4 Termination Criteria

The terminating state is achieved if the $Error_{Structure}$ attribute in the state space does not change over consecutive Sn actions. This ensures minimal variance of the outcome and overcoming possible plateaus in the error space.

3.3.3.5 Training the Network

The goal of the network formation system is to incrementally identify the features required for the classification problem. Upon taking either action, the features already learned by the current structure are not relearned. Thus adding a new set of nodes as shown in Figure 3.4(a) to the current layer only requires the learning of input weights to the new set of nodes keeping the input weights of the old features fixed. The output weights of the layer comprised of the older and newer nodes are relearned since the output weights represent the contribution of each feature in the

classification problem, which changes given any change to the feature set. In order to approximately start in the same locality of the error space and forcing the output to be more stable, the output weights of the existing set of nodes in the new layer along with the bias weight are initialized to the last learned values, while the input and output weights with respect to the new set of nodes in the layer are randomly initialized. On the other hand, adding a new layer to the structure as shown in Figure 3.4(b), requires the learning of both the input and output weights of this new layer, which are initialized randomly.

In order to counteract further instability in the form of oscillations in the output accuracy, which results in a poorer network, the network prior to the onset of oscillations is returned as a solution to the problem.

In order to reduce computational complexity, this system does not prune out nodes which perform poorly, as extraneous nodes do not hinder the classification criterion. Additionally, pruning unwanted nodes would result in wasted time over a network which might possibly never be considered as part of the final ensemble.

Algorithm 2 Train new node or layer

$W_N \leftarrow WeightsOfTheCurrentNetwork$

$S \leftarrow Samples\ From\ Type\ Selection\ MDP$

$N_{new} \leftarrow New\ Nodes$

Freeze Old Weights: $Freeze(\forall W_N \notin Input, Output_{N_{new}})$

Initialize Weights: $\forall W_N \in Input, Output_{N_{new}} \leftarrow RandomWeights.$

Train New Node Weights: $W_N \leftarrow backprop(W_N, S).$

3.4 Experiments

The MDPs were trained with SARSA which is an online Reinforcement Learning algorithm [29], while the state attributes of the Ensemble Learning, Type Selection, and Network Creation MDPs were discretized into bins of size 0.06, 0.2, and 0.03, respectively, while the criteria for stopping was set as observing a state in the Ensemble Learning and Network Creation MDPs consecutively for four times. The first set of experiments were done with synthetic datasets to compare the performance of the proposed approach which has access to a small sample of the entire dataset at any given time, to existing algorithms which had access to the entire dataset. Another experiment was done to evaluate the performance of the proposed approach to existing real world problems. The final experiment over a standard synthetic dataset (SEA) was done to evaluate how fast the propsed approach adapts to datasets with concept drift.

3.4.0.1 Synthetic Dataset

Problems for training, validation and testing were pseudo-randomly generated as multivariate Gaussian distributions. The number of instances in the dataset being in the range of 2500 to 4000 samples in each problem with 65% used for training. The amount of samples N' for building the member networks was set to 500 samples. The performance results of optimized kernel SVMs computed using the libSVM, C-SVM algorithm and standard neural networks with varying predefined structures trained on the full dataset, were used as benchmarks to compare the results obtained in this work. In the case of standard neural networks the maximum accuracy achieved by the five predefined structures were used for comparison, $S = \{\{1200, 800, 300, 100, 30\}, \{700, 500, 200, 50\}, \{500, 300, 200, 100, 30\}, \{200, 150, 70, 50\}, \{100, 70, 30, 15\}\}$, where S_i defines the number of nodes in each layer. For example

S_1 defines a network with 5 layers and $1200, 800, 300, 100, 30$ nodes in each layer, respectively.

This set of diverse training problems was randomly partitioned in a training and test set with 50% of the problems used for learning the MDP policies and the rest for evaluation.

Figures 3.5, 3.6, and 3.7 show the learned policies for the individual MDPs. Once learned, these policies are used to build and maintain the network architecture for the various problems in the evaluation set.

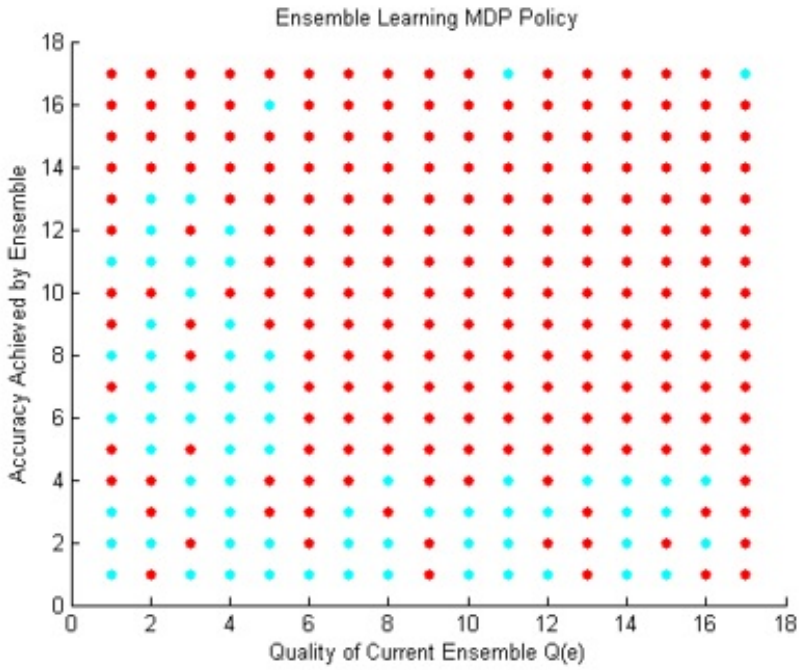


Figure 3.5: Policy learned by the Ensemble Learning MDP from the synthetic datasets

Figure 3.5 shows the policy of the Ensemble Learning MDP. Here, the *red* dots indicate states where a new network is added to the current ensemble, while *blue* dots indicate states where an existing network is removed from the ensemble. The Ensemble learning policy removes networks from the ensemble when either the

Quality or Accuracy of the ensemble degrades. Thus removing obsolete networks from the system, and allowing creation of better sub-networks.

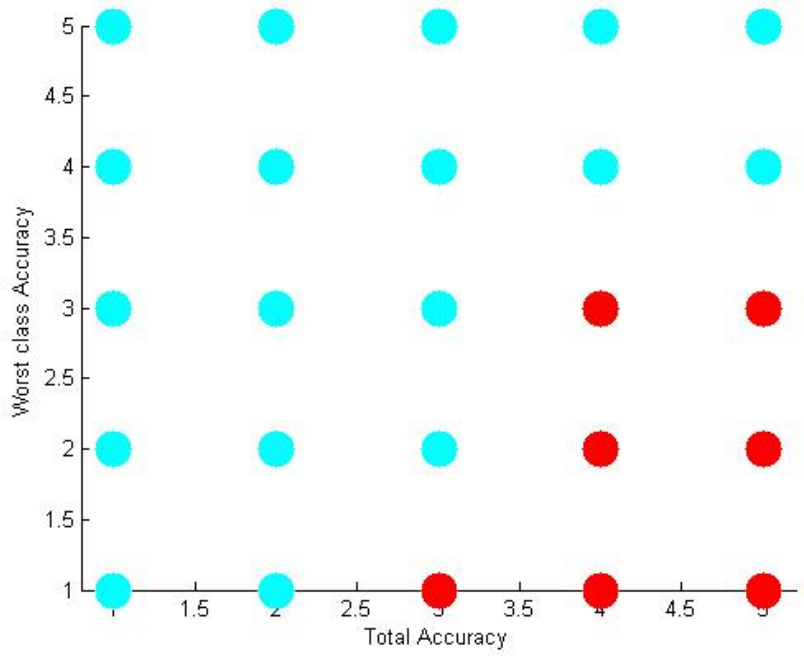


Figure 3.6: Policy learned by the Type Selection MDP from the synthetic datasets

The Type selection MDP decides over the type of classifier to be created. Figure 3.6 shows the policy for this MDP. Here, *blue* dots show states where the optimal policy is to create an $All - vs - All$ network, while *red* dots show states where $Worst - vs - Rest$ are created. This policy shows an interesting aspect, where the agent prefers to create $Worst - vs - Rest$ networks when the ensemble has a high classification accuracy, while a few classes perform poorly. In such situations it is intuitively better to create targeted classifiers in the form of $One - vs - All$ classifiers.

Finally, Figure 3.7 shows the policy learned for the Network Creation MDP. Here, states with *green* dots show states where the agent adds new node(s) to the current layer of the network, *blue* dots show states that add a new layer to the network,

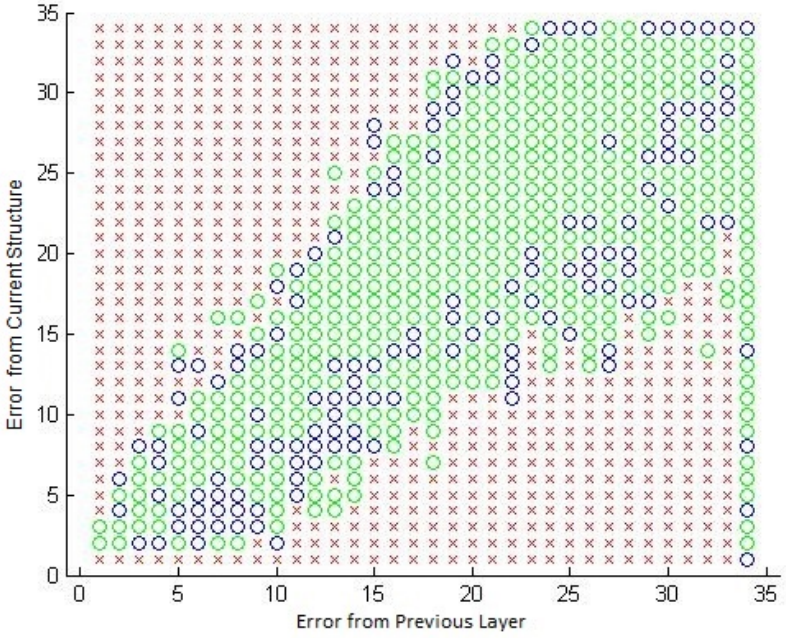
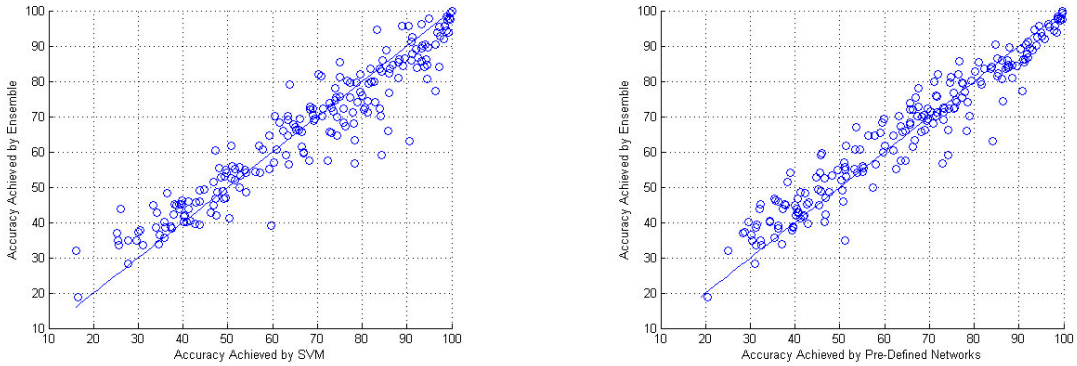


Figure 3.7: Policy learned by the Network Creation MDP from the synthetic datasets

while, *red* crosses depict states not seen during training. This policy shows that the agent in most situations prefers to add new nodes to a layer rather than adding a new layer. However, it also identifies situations where it extends the network by adding layers, in particular at the edges of the encountered errors when the differences between layer errors are largest.

Figure 3.8 shows comparison of the accuracy of the proposed method for each of the test problems, with (a) comparing with the kernel SVM and (b) showing the comparison with the single neural network. This data shows that while there is no significant performance difference compared to the SVM classifier ($p > 0.08$), it performs significantly better on average than the best neural network classifier ($p < 0.01$). This is particularly significant considering that both SVM and neural network classifiers had access to the complete training sets while the ensemble learner

only had access to consecutive random samples of 500 points both during policy learning and classifier formation.



(a) Accuracy compared to SVM

(b) Accuracy compared to pre-defined networks

Figure 3.8: Accuracies achieved on test set with generic policy, each point representing the test accuracy achieved for a single problem in the set of test problems

3.4.0.2 Real World Dataset

Experiments were performed on real world streaming datasets and compared to existing approaches. Table 3.1 shows the comparison of the proposed method against existing approaches [51, 52, 53] for imbalanced datasets and datasets that exhibit concept drift properties, presented as streaming datasets. The policies learned by the system trained previously with the synthetic datasets was used to evaluate performance on four real world datasets called poker-hand, forest covertype, shuttle datasets from the uci repository, airlines dataset from the MOA repository and another standard synthetic dataset (SEA).

Poker Hand dataset is an example of an imbalanced dataset where the training set representation is highly skewed towards few classes. Additionally there is a concept drift in the dataset, however the point of drift is unknown. Since the point

of drift is unknown this dataset achieves a poor performance when evaluated in a streaming environment compared to classifiers which have access to the full dataset. It consists of $1,000,000$ testing and 25,010 training instances with 10 attributes. A sample size of 4,000 was used to train the new networks. The results were compared to posted accuracies by [51].

Forest Covertype dataset consists of 581,012 instances with 54 attributes and has been tested in benchmarks for the evaluation of streaming data classification. A sample size of 3,000 was used. The results were compared to posted accuracies by [52].

Shuttle dataset is another imbalanced dataset where 80% of instances belong to a single class. It consists of 58,000 instances with 9 attributes. A sample size of 2,000 was used. The results were compared to posted accuracies by [53].

Airlines dataset consists of 539,383 instances and 7 attributes with 2 classes. A sample size of 3,000 was used. The results were compared to posted accuracies by [51].

Dataset	Accuracy of proposed method	Accuracy posted elsewhere
Poker hand	59.11%	52.00% [51]
Forest Covertype	75.03%	70.52% [52]
Shuttle	98.42%	95.33% [53]
Airlines	63.66%	60.74% [51]

Table 3.1: Accuracies achieved vs other approaches

Table 3.1 shows that the proposed method performs better compared to existing approaches on streaming data.

SEA dataset [2] is a synthetic dataset with 3 attributes whose values range from 0 to 10 and two classes. The data is divided into four blocks representing different

concepts. There are a total of $2,000,000$ instances generated. The concept of a block is defined as the membership of instances to the two classes as a function of the first two attributes $attribute_1 + attribute_2 \leq \theta$ while the third is irrelevant. The threshold θ dictates the concept of the block. The θ values used were 8,9,7 and 9.5 which was used to evaluate performance of the proposed method in case of concept drift. The approach was evaluated with a dataset from the SEA generator. Additionally 20% noise was also added to the system. This dataset simulates concept drifts in a system with streaming data. Figure 3.9 shows how the ensemble adapts to abrupt changes in concept. The proposed method performs extremely well in this scenario and adapts to the drift in the iteration after the drift is observed.

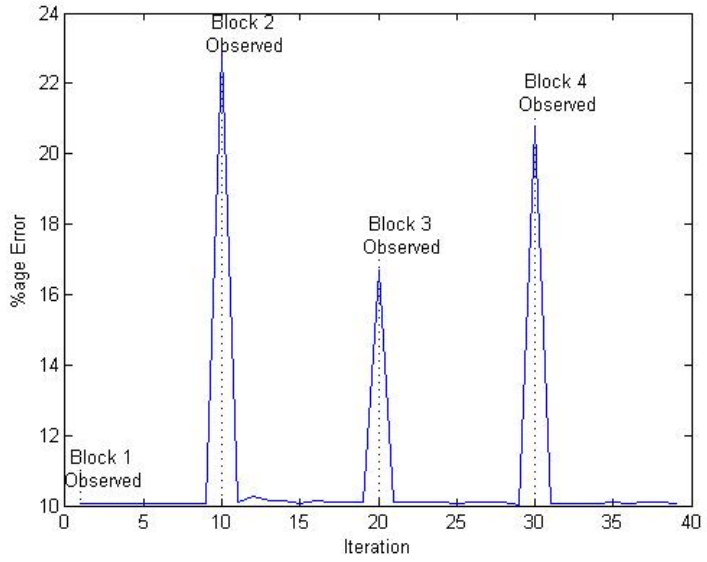


Figure 3.9: Adaptation rate for concept drift dataset SEA

3.5 Conclusion

This proposed approach presents an adaptive ensemble learning technique which incrementally builds the neural network classifiers. This algorithm first learns policies to construct networks using Reinforcement Learning and then uses them on a new problem to build a classifier. In both processes, learning uses incremental, small samples of the training data, making it applicable to streaming data as well as to very large data sets. Experiments show that incrementally building classifiers from a small fraction of samples of the overall larger dataset achieves comparable results to traditional classification algorithms trained with the entire data set. Furthermore, experiments evaluating the performance on real world datasets and the SEA dataset illustrate the approach's applicability to streaming datasets and datasets with the concept drift properties.

CHAPTER 4

Semi Unsupervised Clustering Using Reinforcement Learning

4.1 Introduction

Clustering of data into groups is an important task to perform dimensionality reduction and to identify important properties of a data set. A wide range of algorithms for clustering have been devised that all use some similarity measure that is built into the algorithm to establish groupings of the data with a range of properties [54]. For a given clustering problem, with a set of samples with dimension d, all these clustering algorithms would usually cluster the data based on the built-in similarity/ distance measure, attempting to maximize the distance between the clusters while minimizing the variance within the individual clusters. However, if some sparse information about the desired properties of the grouping of the input data is provided, traditional clustering algorithms fail to utilize such information and often require a reformulation of the algorithm. The need of semi-unsupervised clustering arises, for example, in data sets with large numbers of attributes where most of the attributes are not semantically relevant but will dominate any distance metric (due to their number), used by traditional clustering algorithms. In these cases, sparse information regarding the quality of clusters or regarding relations between a small number of data points might be available which could be used to modify the cluster formation process. For example, given a set of data points for cars that are to be clustered for subsequent use in performance characterization, secondary attributes such as color, year, registration state, etc, might lead to clusters that do not have desired characteristics. In these cases, sparse feedback either from a subsequent task or provided

as sparse feedback on the membership of some samples and their respective clusters could be used to improve the clustering results.

Semi unsupervised clustering formalizes a type of sparse constraints which might be applied to a clustering problem [55], thus allowing constraints on the input data in order to direct the clustering algorithm towards an answer which satisfies given constraints. This framework defines two possible types of constraints, same cluster constraints which indicate that points should be in the same cluster, and a different cluster constraints which indicates that points should be in different clusters. These constraints are often called $must-link$ and $must-not-link$ constraints, respectively, each constraint being defined over a pair of points in the problem set. Same cluster constraints state that for a given pair of points, their cluster membership should be of the same cluster. Similarly for different cluster constraints over a pair of points, the points should lie in different clusters. Note that this in no way specifies the cluster which a point should lie in but only identifies whether two points should or should not belong to the same cluster.

Given the input samples, it is often not possible to cluster the data according to the constraints in their original feature space using unmodified distance measures as indications for similarity. Thus we have to modify the feature space, usually by scaling the dimensions, so that an unmodified clustering algorithm is able to cluster based on it's own distance and variance constraints. In order to solve this problem, this work presents a novel approach [56] which, at first, learns a policy to compute the scaling factors using Reinforcement learning from a set of training problems and subsequently applies the learned policy to compute the scaling factors for new problems. The goal here is that by working on the scaled dimensions, the traditional clustering algorithm can yield results that satisfy the constraints.

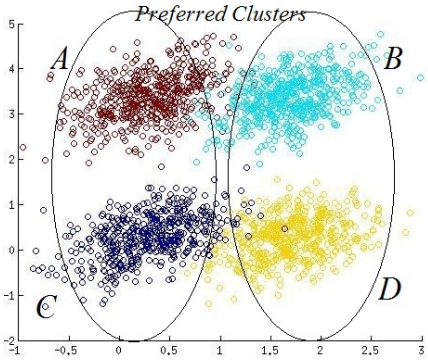
4.2 Existing Methodologies

Existing methods to solve the problem usually include a regularization term in the cluster optimization criteria, therefore, redefining the goal of the clustering algorithm in order to include the satisfaction of the given constraints. Using this regularization term [6], these approaches then perform the clustering algorithm within an expectation maximization framework [7] in order to solve for both the constraint satisfaction and the similarity optimization [57, 8, 9]. For such methods one must, along with satisfying the constraints, also satisfy the cluster validity requirements, leading to a complex overall application with the need for a modified clustering algorithm, yielding a framework that is limited to the specified constraint types [58]. Additionally, constraints often contain misleading information, which results in the miscalculation of the final cluster. To solve this problem, a subset of constraints are selected from the entire set as shown in [59] to achieve the required grouping.

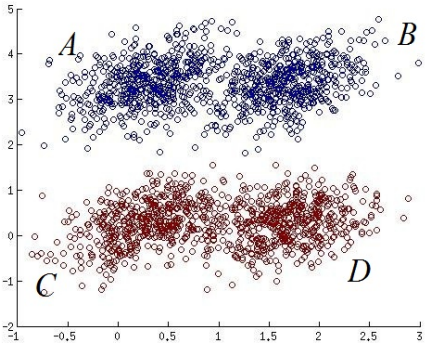
4.3 Approach

Consider the set of sample data shown in Figure 4.1(a) that is originally generated from four distinct distributions (A,B,C,D), to be grouped into two clusters. When a traditional clustering algorithm is applied on the data, it clusters the data into (A,B) and (C,D) as shown in Figure 4.1(b).

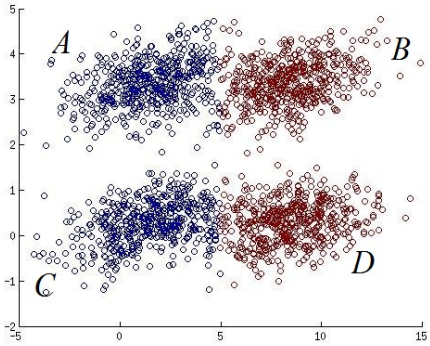
However, if constraints are provided over the data points indicating that some points in A and C should be clustered together and so should some points in B and D, a midification is necessary. In order for the unmodified clustering algorithm to produce the desired result, morphing the input representation is required. For example, scaling the data along the x-axis, increases the importance of similarity along the x-axis while keeping the y-axis variance the same. This would force the



(a) Original Data Resulting from Four Distributions A,B,C,D colored red,cyan,blue and yellow respectively



(b) Data Clustered Using Traditional Clustering Using Cartesian Distance



(c) Clustering of Re-Scaled Data with the data expanded along the x-axis by a factor of 5

Figure 4.1: K-means clusters. Solution without dimension scaling (Left); Solution with dimension scaling (Right)

clustering algorithm to now group the data into (A,C) and (B,D) as shown in Figure 4.1(c) which, in order to satisfy its own distance and variance optimization criteria, additionally also satisfies the point constraints. The constraints given to the system are often the result of human feedback, and are thus prone to conflicting constraints. Additionally, solving every constraint simultaneously in an analytic fashion would be computationally expensive.

Thus an iterative approach is taken, wherein reinforcement learning is applied. At each iteration an individual constraint, i.e., a constraint over a pair of points of either same cluster or different cluster type is chosen and the dimensions are scaled accordingly in order to better satisfy the given constraint.

For a same cluster type constraint, the dimensions are to be scaled in such a way that the distance between the two points is reduced. This is achieved by shrinking the dimension along which they are farthest apart. Similarly for a different cluster type constraint the dimensions are to be scaled in a way that increases the distance between the two points. This is achieved by expanding the dimension along which they are the closest. Thus as shown in Figure 4.2 for the two constraint points connected with a solid dark line, in case this is a same cluster type constraint the x-axis is shrunk because the points are farthest apart along this dimension; however if the constraint is of different cluster type, the y-axis would be expanded, since the points are closest along that dimension.

Instead of scaling a single dimension at a time a subset of the dimensions might be scaled simultaneously, i.e., for shrinking a subset of dimensions might be chosen that have the highest distances, and vice versa for the different cluster constraint type. If the dimensions along which scaling is performed are the same as those of the original inputs, the number and forms of the potential final clusters are limited. For example, in a two dimensional scenario, the final clusters formed with two clusters can only be separated along the axes. However, converting the input dimension to a kernel space with a distance metric to each point, the scaling process has more degrees of freedom as it can scale along a much higher number of dimensions, resulting in a more complicated and nonlinear separation plane with the caveat that it is much more computationally expensive to perform clustering and scaling in the resulting dimensions. To trade off the computation complexity versus the desired higher com-

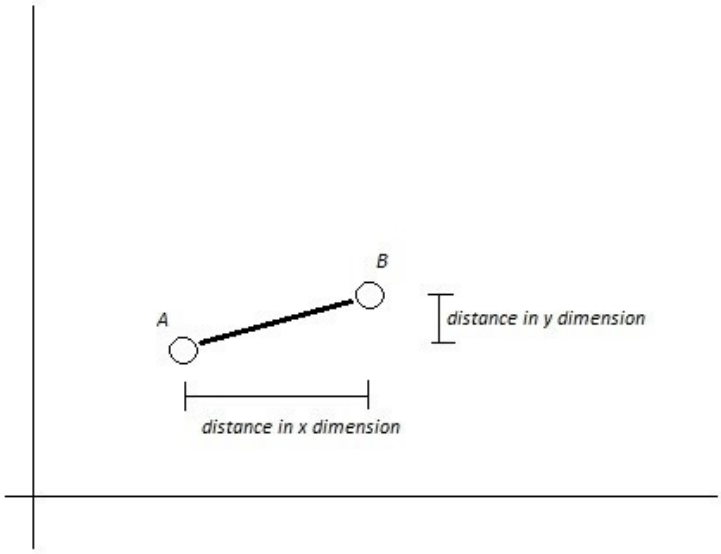


Figure 4.2: Distance Along a Pair of Constraint Points

plexity of the solution, the space is projected onto a higher dimension using kernels but only use the constraint points given as points along which the similarity metric of the kernel space is computed. A radial basis function (RBF) is used as a kernel function [60], which allows the kernel function to have a local property where points closer to each other should behave similarly. Thus, for input data with m samples and d dimensions, the data samples are represented in a $d + 2 * c$ dimensional space where c is the number of constraints provided, each specifying a pair of points over which the constraint is defined. In general, this is a much smaller number than the number of samples m. Since the new dimensions are functions of the distances from the constraint points, points closer to such a point lie closer together. For example, given a set of 10 dimensional data of 100 points and eight constraints, the resultant space after conversion would be of 26 dimensions where the 10 features in the original input are concatenated with the 16 new features computed, where the similarity met-

ric of the kernel is only computed with respect to the points in the constraints. This allows the shrinking or enlarging in the kernel space along a dimension to be applied more heavily to points closer to the pair of points in the constraint being satisfied.

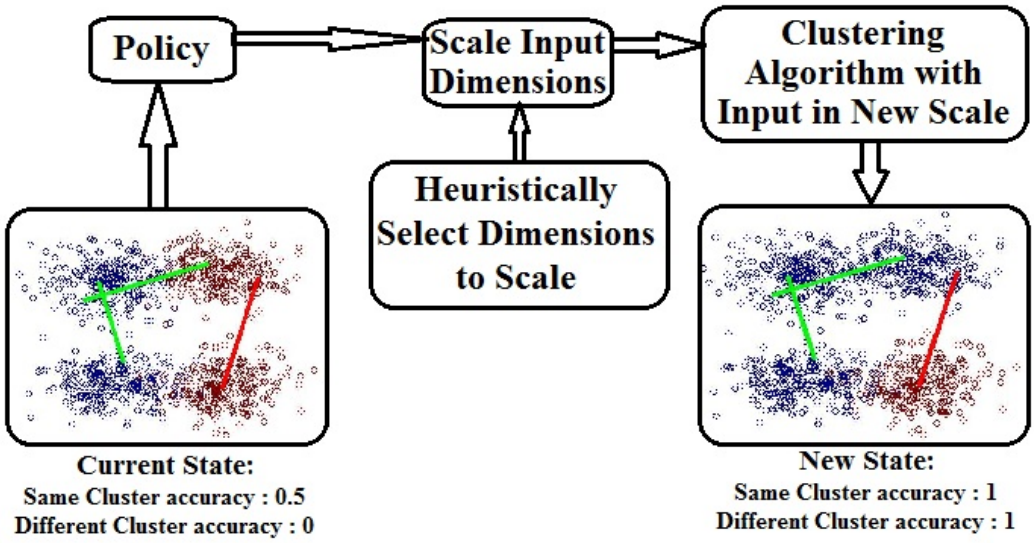


Figure 4.3: Using Reinforcement learning to scale dimensions

Figure 4.3 shows an overview where reinforcement learning is used to guide the clustering process. Upon mapping the dataset onto the new feature space, the new dimensions are scaled using a Markov Decision Process (MDP) policy learned by the Reinforcement Learning algorithm and subsequently a traditional clustering algorithm is applied on the new feature space, which now achieves the desired clusters.

4.4 Learning the Similarity Function

To apply Reinforcement Learning to the learning of the effective scaling of the feature dimensions, an MDP is defined that uses two state variables, namely, the accuracy for same cluster constraints and the accuracy for different cluster constraints. Since the variables are continuous, the state space is discretized into bins of size B.

It should be noted that, as with all such discretizations, the choice of discretization level is a trade-off between the fine grained accuracy of the current state status and the computation complexity of the algorithm.

State Space of the MDP : To compute state attributes, constraints are divided into two groups, ones that are satisfied in the current clusters and ones that are not. For the ones that are not satisfied, a degree of them being satisfied is calculated. This degree is used as a metric to identify how close to satisfied a constraint is. Using a probabilistic view on cluster membership, the probability of a point, x, being in a cluster, K_i, is defined as

$$P(K_i|x) = \frac{distance(x, K_i)}{\sum_j distance(x, K_j)}$$

The degree of a *same cluster* constraint, $C_{j,k}$, being satisfied is a function of the probabilities of the two member points, x_j and x_k, in the constraint. For each cluster, the likelihood of them lying in the same cluster, K_i, is defined as

$$P(K_i|C_{j,k}, type(C_{j,k}) = same\ cluster) = P(K_i|x_j) * P(K_i|x_k)$$

and the degree of a *same cluster type* constraint to be fulfilled is then calculated as the maximum product of the probabilities for each point regarding every cluster. Thus the degree of a constraint being satisfied is the maximum likelihood for any class that both points are in that cluster. The maximum (rather than the sum) is used here to achieve a stronger tie across multiple constraints containing the same data point since it forces a single cluster to be picked for each constraint. Similarly, for a *different*

cluster type constraint, $C_{j,k}$, the probability of a point belonging to cluster K_i while the other does not, is defined as

$$P(K_i|C_j, k, type(C_{j,k}) = different\ cluster) = P(K_i|x_j) * (1 - P(K_i|x_k))$$

and the degree of the constraint to be satisfied is again calculated as the maximum across all clusters of this probability. The accuracy of a constraint type as a state variable is defined in Equation (4.1).

$$P_t(satisfied) = \frac{\#satisfied\ constraints\ of\ type\ t}{\#constraints\ of\ type\ t} \tag{4.1}$$

$$Accuracy_t = P_t(satisfied) + (1 - P_t(satisfied))* E_{i \in unsatisfied\ of\ type\ t}[P(K_i|C_{j,k}, type(C_{j,k}) = t)] \tag{4.2}$$

This results in two state attributes, $Accuracy_{same\ cluster}$ and $Accuracy_{different\ cluster}$, which are both in the range of [0..1]. The attribute is then discretized into bins of size B in order to reduce the complexity since state values closer to each other are related and thus can be grouped into a single state, the size of the group determined by the bin size B.

Action Space of the MDP : Given the degrees of satisfaction of the individual constraints of any type, the unsatisfied constraints closest to satisfaction and the constraints farthest from satisfaction are identified, i.e. for each constraint type the unsatisfied constraints with the highest and the lowest probabilities are selected. There are 4 actions which the agent can take in every state in order to satisfy the constraints.

- For *same cluster* constraint type, fix unsatisfied constraint farthest from satisfaction

- For *same cluster* constraint type, fix unsatisfied constraint closest to satisfaction
- For *different cluster* constraint type, fix unsatisfied constraint farthest from satisfaction
- For *different cluster* constraint type, fix unsatisfied constraint closest to satisfaction

Upon choosing the specific constraint to fix, the dimensions to scale are chosen. For *same cluster* constraint type, a subset of the dimensions with the highest distance is selected and scaled down, bringing them closer in the new feature space. Similarly for *different cluster* type, the dimensions with the lowest distance are selected and scaled up, pushing them farther apart.

Reward Function : The reward function is simply defined as the improvement in the satisfaction of the constraints as measured by the sum of the difference between each of the two accuracy variables of the new state and the current state.

Goal State or End Criteria : The goal state is defined as the state where all the constraints are satisfied. However it is often not possible to reach this state due to many factors like human error in constraint selection or other contradicting constraints. Thus, the condition where the state does not change for a given fixed number of steps is allowed as a termination condition.

4.5 Experimental results

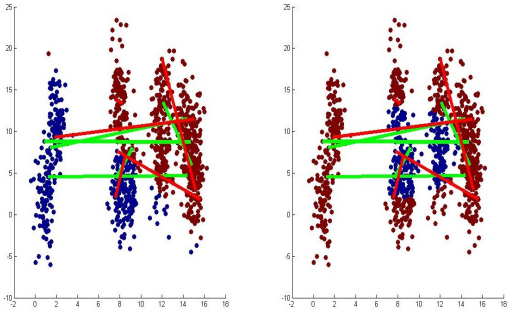
Problems for training, validation and testing were pseudo-randomly generated as multivariate Gaussian distributions. Parameters controlling the individual cluster size, number of distinct clusters, mean and co-variance of each cluster, along with the number of input dimensions were seeded randomly, with a range of 400 to 800

samples in each problem. Furthermore, a set of constraints for each problem were randomly chosen from the data points. Reinforcement learning was used to learn 5 different policies over a set of 70 training problems. Subsequently, the policies were used to compute the dimension scaling factors for 30 test problems similar to their respective training problems which were unseen during the policy learning phase.

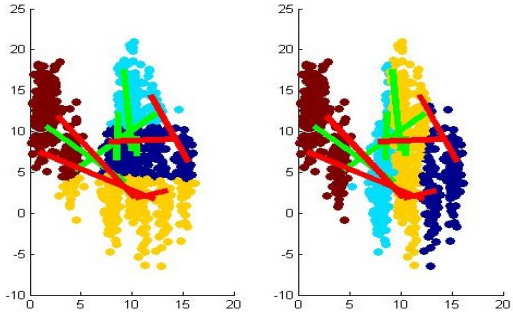
For the experiments, a bin size of 0.05 was chosen, i.e. state variable values are discretized into the accuracy intervals $[0..0.05), [0.05..0.1), ..., [0.95..1]$. For the actions, scaling steps of 0.7 and 1.3 are chosen. For *same cluster* constraints the top third of the dimensions with the highest distance are scaled down by multiplying the dimensions by 0.7, and for *different cluster* constraints the bottom third dimensions are scaled up by multiplying the selected dimensions by 1.3. The algorithms used were K-means clustering [61] for the clustering process and SARSA[28] as the reinforcement learning algorithm [24] for training the MDP.

The policies were trained with problems similar to each other with respect to the number of dimensions in the original dataset and the number of proposed constraints on the problem. Finally a policy was also trained with a broad range of problems, for comparison with the performance of the other specialized policies.

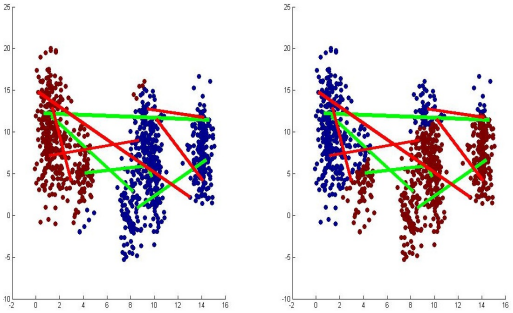
Figure 4.4 shows the result of a cluster output of K-means on the original dataset and the output upon scaling the dimensions using the policies learned. Red lines indicate *different cluster type* constraints while green lines indicate *same cluster type* constraints. Figures 4.4(a-d) are solutions to four 2-dimensional problems, each with 10 constraints. Although, a few constraints might be contradictory, the proposed approach is able to learn a representation which satisfies most constraints. By contrast, Figure 4.4(e) is a 20-dimensional clustering problem, the solution to which was mapped to 2 dimensions using t-sne [62] for visualization. Table 4.1 shows the performance of the individual policies on 30 test problems and the range of dimen-



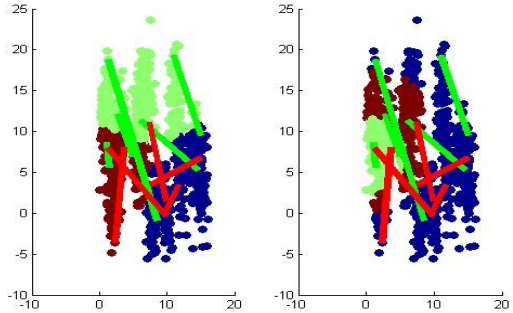
(a) problem with 2 potential clusters, left: initial clustering has 3 satisfied constraints right: the final solution is able to solve for 7 out of 10 constraints



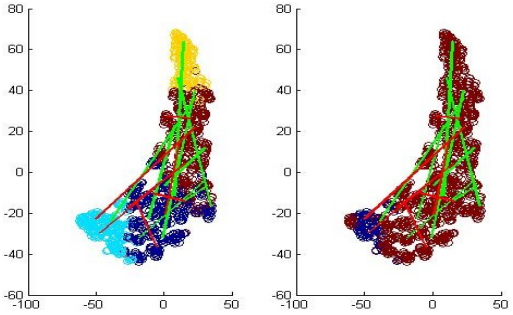
(b) problem with 4 potential clusters left: initial clustering has 4 satisfied constraints right: the final solution is able to solve for 8 out of 10 constraints



(c) problem with 2 potential clusters left: initial clustering has 4 satisfied constraints right: the final solution is able to solve for 8 out of 10 constraints



(d) problem with 3 potential clusters left: initial clustering has 3 satisfied constraints right: the final solution is able to solve for 7 out of 10 constraints



(e) Solutions for 20-dimensional dataset mapped to 2 dimensions using t-sne for visualization

Figure 4.4: K-means clusters for a range of generated problems and constraint sets. Left figures show the initial clusters and the right figures show the final clusters after scaling with the learned policy.

sions and constraints learned upon. Policies $1-4$ are learned on specific types while policy 5 is a generic policy.

policy	# dimensions	# constraints	% satisfied
1	2 - 7	4 - 10	81.2%
2	5 - 10	20 - 25	84.8%
3	7 - 15	20 - 25	72%
4	15 - 20	10 - 15	73.33%
5	2 - 20	10 - 25	40%

Table 4.1: Performance analysis of individual policies

Since the constraints were randomly generated, it is often impossible to solve for all of them because of conflicting constraints. However, the solutions computed from the policies above solve for the highest possible number of constraints while maximizing the accuracy of unsatisfied constraints as shown in Equation (4.2). Table 4.1 also shows that policies learned for a specific type of problem (policies 1 to 4) perform much better than a generic policy (policy 5).

4.6 Conclusion

The proposed framework shows that semi-unsupervised clustering is possible by re-mapping and scaling the original input dimensions, without modifying the clustering algorithm. Thus, this method can be applied to any traditional clustering algorithm. Additionally experiments show that performance of the generic policy is poor compared to policies from specific types of problems.

CHAPTER 5

Training Neural Networks with Policy Gradient

5.1 Introduction

A neural network is a powerful modeling tool which has the ability to approximate arbitrarily complex functions, provided the optimization function has a gradient. However, it falls short if the function to be optimized does not have an inherent gradient or if the function to be optimized is a black box where the equation is unknown.

There are many problems where such scenarios are evident. For example, In a classification task where instead of the target, a function outputs if the classification was correct or not. Traditional classification neural networks fail to work in these situations where a target is not explicitly specified,.

A similar problem arises where lateral inhibitions on a sparse autoencoder are to be imposed on the features. Imposing such constraints, usually termed as structural constraints, is difficult since they require lateral connections among the nodes in a particular layer. This results in a complicated, recurrent network architecture requiring gradient updates for a node to be affected by sibling nodes connected in a distributed fashion. However, imposing such a constraint without lateral connections among the hidden nodes is not possible as the gradient is non-computable without the error and activation information from the neighbor nodes.

The proposed method solves such problems where the gradient is non-differentiable or simply non-computable by modeling an approximate smooth representation of the

optimization function, and subsequently deriving the gradient from that approximation instead.

A similar gradient problem is present in the case of solving reinforcement learning problems with neural networks where there is no inherent target but only a system of rewards which is a qualitative utility of the output. In this case a secondary critic network is used to approximate the gradient of the system of rewards with respect to the output actions at each state. This method is called policy gradient where one network learns which actions to take while another network approximates the required gradient of the rewards with respect to the output actions[63].

The proposed method uses a modification of the policy gradient algorithm and represents the problem as a context based n-armed bandit problem in order to enforce the non-differentiable or non-computable constraints from a system of rewards. The functions are considered to be a black box, modeled in a critic network where the state is the input sample and the actions are the activations from nodes on which the constraint is to be applied. The constraints are represented as a reward function which is modeled by the critic network. Given any reward function which might be non-differentiable the critic network learns a smooth approximation of the constraint. The gradient of the output of this network with respect to the activations is then fed into the actual actor network.

Experiments carried out show that the proposed method is able to solve problems which impose arbitrary structural constraints on the networks in addition to minimizing the reconstruction error. Additionally it is shown that it is able to solve multi-label classification problems without an actual target but with a correct or wrong outcome indicator. Finally, a novel form of structural lateral sparsity is introduced which results in much sparser representations without degrading the reconstruction error. Lateral inhibitions are shown to learn more concise and an order of

magnitude sparser features with better generalizations compared to traditional KL sparsity[13]. However, this form of sparsity is rarely used in the real world due to their high computational complexity and complex architecture design. The proposed approach negates the need for interconnected nodes in the hidden layer for lateral inhibitions, thus reducing the complexity, while learning better laterally sparse features, thus facilitating their use in real world applications which was not possible earlier.

5.2 Existing Methodologies

Non-differentiable constraints are usually optimized by converting the function into a multitude of convex piecewise linear functions or solving the function with a hill climbing algorithm like simulated annealing. The function is then solved based on the piece the function currently lies on [64]. However, this is not possible for functions like the L0 norm which do not have a gradient for any piece. On the other hand solving such constraints using simulated annealing is infeasible for higher degrees of freedom of the constraint.

Another approach to solving such problems is to consider one variable at a time of the constraint and solve for it using sequential minimal optimization (SMO)[12]. The constraint is formed as a dual problem, for example using Moreau-Yosida regularization [65], for the SMO algorithm.

Existing methods of imposing a sparsity constraint involve forcing a feature to fire as few times as possible for all samples [66]. KL-divergence between the current average activation and a given target average activation is used to compute the gradient for the sparsity constraint. However, another type of sparsity in the form of lateral inhibitions is often imposed. This form of sparsity tries to minimize the number of active features required to reconstruct a sample.

Existing methods achieve laterally inhibited sparse features using lateral connections among nodes in a hidden layer [13]. Such connections are needed in order to share information about which nodes have fired in the locality. The gradient for a feature node is computed with respect to the laterally connected nodes. Although resulting in a tighter and more compact representation of the data, it is computationally expensive and requires a complex network architecture. Thus, only a few neighbor nodes are locally connected in order to reduce the complexity while retaining some lateral sparsity information.

5.3 Overview of Reinforcement Learning, Actor-Critic, and Policy Gradient

Reinforcement learning is a class of algorithms which learn decision making systems[67]. Given a state, which represents the current status of the problem, an agent needs to learn which action to take in order to maximize a reward function. However, the agent does not know the correct action for each state and has to learn this from intermediate or delayed rewards, possibly at the end of the entire sequence of actions. The agent learns the best possible action to take by randomly exploring all actions for the states and computes a utility value for each state action pair. This utility function is known as the value function. This utility value is then used to determine the best action to take at each state.

There are many Reinforcement learning algorithms which are used to learn this utility value or Q-value. The SARSA algorithm [29] updates a Q-value for a state action pair as a function of the reward for the current state action pair, and the utility of the next state and action to be taken [29]. This is known as the Bellman equation.

$$\begin{aligned} Q(S_t, a_t) = Q(S_t, a_t) + \alpha(R(S_t, a_t) \\ +\gamma(Q(S_{t+1}, a_{t+1})) - Q(S_t, a_t)) \end{aligned} \tag{5.1}$$

To solve such a system, which inherently does not have a gradient towards the best possible action, using neural networks, the policy gradient algorithm is used [68] [69]. This method, uses an actor-critic model, where the actor learns the best action to take as an output for a given input state [70]. The actor network learns a probability distribution over the set of actions for a given state. An action is chosen based on this distribution [36]. The critic network learns the value function with the current state and chosen action as input. The gradient of the critic network value output with respect to the actions is then used to train the actor network [34].

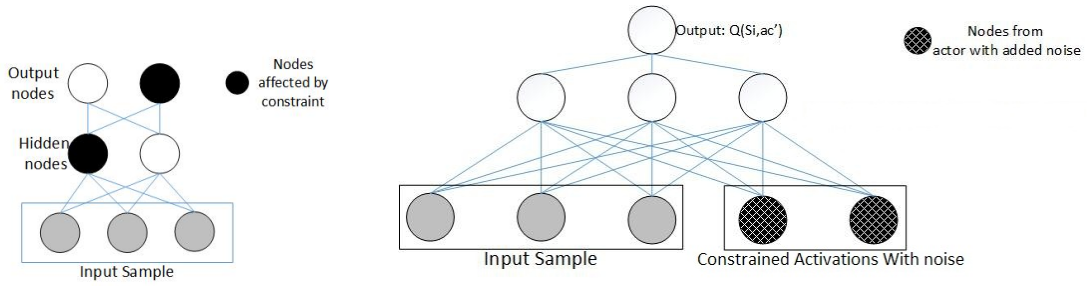
However, there are some simpler problems where there is no next state. In such cases, taking an action from the state results in termination and a reward. The agent has to learn which action results in the highest reward. Such systems are usually known as n-armed bandit systems [71].

Additionally, there are problems where there exist multiple states, but taking an action in any state results in a termination and a reward. In such cases the agent, similar to an n-armed bandit, has to learn which action results in the maximum reward for every state in the system. This is a slightly different problem than n-armed bandit, and is usually known as a context-based n-armed bandit system [72].

5.4 Approach

The proposed approach uses policy gradient to impose constraints on the network instead of learning actions. It solves the problem of constraint satisfaction using an actor critic model. The given constraint C, is encoded as a system of rewards $R(S_i, a_c)$ which is a metric representing the satisfaction of the constraint. The critic network approximates this reward function and provides the gradient with respect to the activations to the actor network. In contrast to policy gradient where the input to the critic network is the action taken by the actor network, the network models the

reward function from the sample and activations of the nodes where the constraint is to be applied. The overall architecture of the Actor and Critic Networks with respect to the constraint nodes is shown in Figure 5.1.



(a) Actor network (b) Critic network

Figure 5.1: General network architecture. (a) Actor network, shaded grey nodes indicate input nodes while black nodes specify the nodes over which the constraints are to be applied. (b) Critic network, checkered black nodes indicate values from constrained nodes with added noise. Shaded grey inputs and activations of affected nodes used as input to critic network, the output being the utility for the activations produced

Let A be the set of activations in the actor network for all nodes in the input layer a_i, hidden layer activations a_h and the output layer activations a_o.

$$A = \{\{a_i\}, \{a_h\}, \{a_o\}\}$$

For a given sample S_i, a_c is defined as the activation of the set of nodes affected by the given constraint C over the set of all activations A in the actor network as shown in Figure 5.1(a).

$$a_c \subseteq A$$

For example, in case of the classification problem a_c is the set of all activations in the output layer, a_o. In genral, however, the activations where the constraint is to be applied can be a combination of a_h and a_o activations from different layers.

Similar to policy gradient systems, in order to train the critic network, some exploration is required to approximate the space defined by the reward function. In order to achieve this, a small zero-mean gaussian noise is added to the activation outputs of the affected nodes. This process is similar to exploring actions in policy gradient and allows the system to explore new nearby activation values which might lead to better rewards during training.

$$a'_c = a_c + \mathcal{N}(0, \sigma^2)$$

The new activations a'_c so formed, in addition to the given sample, is then used as an input to the critic network as shown in Figure 5.1(b). Given the reward from the constraint C the critic network then approximates the utility of the activations for the given sample.

For each sample, the activations a'_c with the added noise in the actor network are used as inputs to the critic network along with the input values for the sample. The critic network is trained with the reward from the constraint as the target. This reward might be a one or zero as in the case of the classification problem without a target dataset or in the case of sparsity with lateral inhibitions, the reward might be the L0 norm of the constraint nodes per sample.

The proposed model represents the problem as a context based n-armed bandit system, where there is no next state, and for each sample there exists a set of activations which maximizes the reward. The target of the critic network drifts during training and depends on the current output of the network. The target value of the network is given by the Bellman equation. The utility $Q(S_t, a'_{c_t})$ of a set of activations a'_{c_t} for a given sample S_t, ignoring the next state is computed as

$$Q(S_t, a'_{c_t}) = Q(S_t, a'_{c_t}) + \alpha(R(S_t, a'_{c_t}) - Q(S_t, a'_{c_t})) \tag{5.2}$$

The critic network is trained by minimizing the mean-squared error of the output of the critic network and the target. The target of the network maintains an adaptive average and over time converges to the expected reward $E[R(S_t, a'_{c_t})]$, for activations a'_{c_t} and sample S_t. Given the error function as the mean-squared error where the output y is given as the current utility learned by the network for sample S_t and activation values a'_c

$$y = Q(S_t, a'_{c_t}),$$

the new target τ computed as a function of the current output y and the reward incurred is given as

$$\tau = Q(S_t, a'_{c_t}) + \alpha(R(S_t, a'_{c_t}) - Q(S_t, a'_{c_t}))$$

The mean squared error to train the critic network is computed as

$$Error = \frac{1}{2}(\tau - y)^2$$

$$= \frac{1}{2}(R(S_t, a'_{c_t}) - Q(S_t, a'_{c_t}))^2$$

The gradient of the critic network is computed as

$$\frac{\partial Error}{\partial Q(S_t, a'_{c_t})} = R(S_t, a'_{c_t}) - Q(S_t, a'_{c_t})$$

which is the Bellman error for n-armed bandit systems[73]. This gradient is then backpropagated to train the critic network. The weight w update is then computed as

$$\frac{\partial Error}{\partial w} = \frac{\partial Error}{\partial y}\frac{\partial y}{\partial w}$$

The gradient for the layers is backpropagated for the entire network while training. The only difference is that the target is drifting over time and the gradient is the Bellman error.

The critic network is trained using this gradient and therefore tries to approximate the reward function and output as closely as possible. If the reward function is non-differentiable, it learns a smooth approximation for the function. On the other hand, the actor network tries to maximize the reward function. This is a different gradient than the one used to train the critic network which in contrast tries to get consistent reward function outputs. To obtain the constraint related error for the actor network, a separate gradient is computed during backpropagation in the critic network which is the gradient without the Bellman error. This function is the gradient of the output y of the critic network which is $Q(S_t, a'_{c_t})$ with respect to the activations a'_c.

$$\frac{\partial Q(S_t, a'_{c_t})}{\partial a'_{c_t}} \tag{5.3}$$

Analogous to the policy gradient system, the critic network learns the value function whose output is defined by the rewards from the constraint system. The probabilistic actions are replaced by the noisy activations of the actor network.

Thus, in order to train the system, the actor network is forward propagated to record the activations for the nodes affected by the constraint C. A zero mean gaussian noise is added to these activations a'_c. Subsequently, the samples and their respective noisy activations a'_c are fed as input to the critic network. The Bellman error is computed for the critic network from the output of the network and the new target computed from the output and the reward. This error is used to train the critic network. A separate gradient is computed as the derivative of the output with respect to the activation inputs. This gradient maximizes the reward function, i.e., the output of the critic network, with respect to the activations. Therefore the actor network has to learn to maximize the reward function by modifying the activations.

This gradient may be applied to the actor network in two ways. It can be used as the task error gradient in case of the classification problem with incomplete target data. In this case the output layer activations a_o are the affected nodes. The actor network has to learn the activation class outputs for which the reward is maximized. In this case the gradient from the critic network is the sole gradient applied to the actor network.

In contrast, to solve the problem of lateral inhibitory sparsity constraint, this gradient is applied to the hidden layer nodes. The gradient from the critic network is combined with the task error or reconstruction error, backpropagated from the output layer, as a regularization term in the hidden layer.

$$\frac{\partial E}{\partial w} + \theta\left(\frac{\partial Q(S_t, a'_{c_t})}{\partial a'_{c_t}}\right)$$

where $\partial E/\partial w$ is the gradient element from the task error, and θ controls the weight of the sparsity term from the critic network. In this case the actor network has to learn activations in the hidden layer which, in addition to the actual reconstruction error, also have to maximize the reward accrued from the imposed structural constraint.

5.5 Classification with Incomplete Target Data

In traditional multi-label classification problems the network tries to minimize the mean squared error E between the output y and the target τ. The gradient is defined as

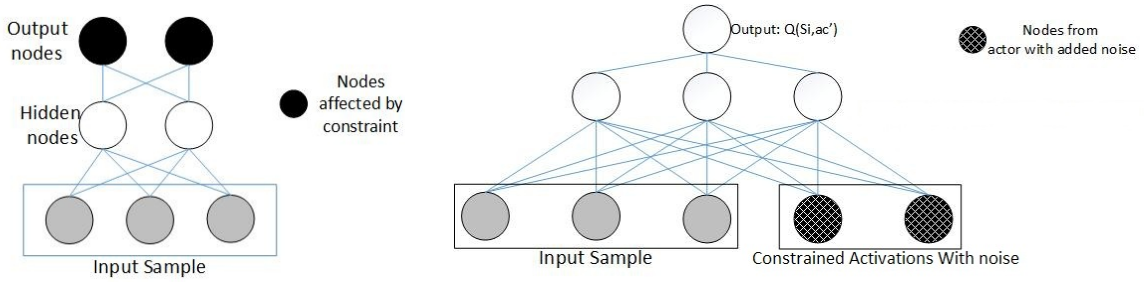
$$\begin{aligned} E &= (\tau - y)^2 \\ \frac{\partial E}{\partial y} &= \tau - y \end{aligned} \tag{5.4}$$

However, this is not possible for problems where the target τ is not defined. For example, there are problems where, instead of the actual target, there exists a correct

or wrong boolean indicating whether the sample was correctly classified or not. The function can be represented as

$$f(y) = \begin{cases} 1 & : C_y = C_\tau \\ 0 & : C_y \neq C_\tau \end{cases} \tag{5.5}$$

Here C_y is the class from the output y and C_t is the correct class of the input. In such cases the only information available is non-differentiable. The network is to be rewarded for correct classification and no reward for wrong classification. For such problems the system does not know what the actual class is nor does it know which output nodes correspond to which class. For the proposed model in this case the constraint to be satisfied is simply given by Equation 5.5 which provides the reward for the critic network. To model this, the network architecture is instantiated as in Figure 5.2.



(a) Actor network (b) Critic network

Figure 5.2: Target-less classification network. (a) Actor network, shaded grey nodes indicate input nodes while constraints are applied on the output layer nodes. (b) Critic network, with the input and noisy actor output node activations as input

Here the nodes affected by the constraint are the output node activations, $a_c = a_o$. The critic network has to learn the constraint function such that the gradient in Equation 5.3 approximately equals

$$\frac{\partial Q(S_t, a'_{c_t})}{\partial a'_{c_t}} \approx \frac{\partial E}{\partial y}$$

5.5.1 Experiments

Experiments were performed to compare the classification accuracy and mean-squared error achieved by classification using the proposed model and a traditional neural network. Training was done with a reward of 1 if correctly classified and zero if incorrectly classified and compared to the traditional classification network which had access to the actual target data. The classifier network was trained with 70% of the number of input nodes as hidden layer, and sigmoid activation functions for both networks. Since the reward term is either one or zero, sigmoid activation functions were also used in the critic network. A larger network might be used to increase performance of the task, for example increasing the number of layers and nodes would boost the performance of these tasks. However, a single hidden layer with few nodes was chosen in order to demonstrate the validity of the proposed approach.

The model was tested on three real world problems, by dividing the data into training, validation and testing segments.[1]

5.5.1.1 Iris dataset

The iris dataset has four input features and 150 samples in the dataset with three classes.

5.5.1.2 Thyroid dataset

The thyroid dataset has 21 input features and 7200 samples in the dataset with three classes.

[1]All datasets were accessed from UC Irvine Machine Learning Repository http://archive.ics.uci.edu/ml/

5.5.1.3 White Wine Quality dataset

The white wine quality dataset has 11 input features and 3918 samples and 9 classes

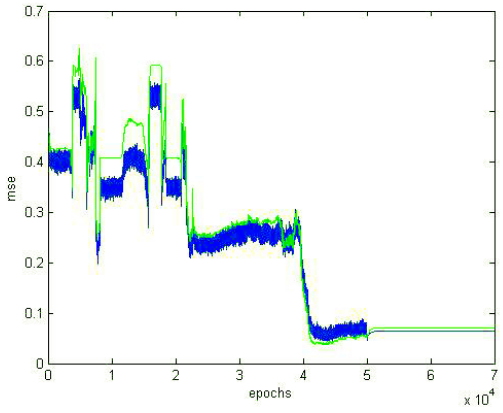
Table 5.1 shows the comparisons of accuracy and mean squared error achieved for the classification problems with the proposed model with incomplete data and traditional neural network classifiers with the entire dataset. The first two columns show the results from the proposed model while the traditional classifier is shown in the next two columns.

	Proposed		traditional	
Dataset	accuracy	MSE	accuracy	MSE
Iris	**96.67%**	0.0664	96.67%	0.0096
Thyroid	93.2%	0.037	**94.2%**	0.031
white Wine quality	**56.63%**	0.071	56.4%	0.067

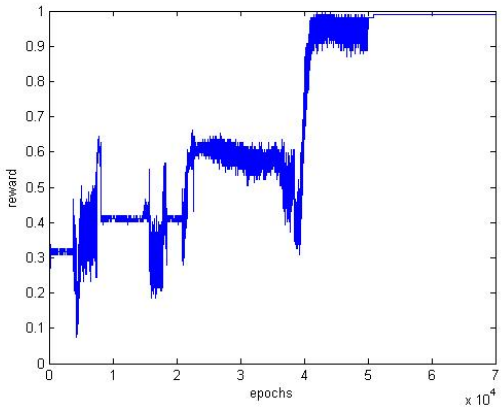
Table 5.1: Results from incomplete target data vs traditional network with complete target data

This data shows that despite having less information the proposed approach is capable of learning a similar quality solution.

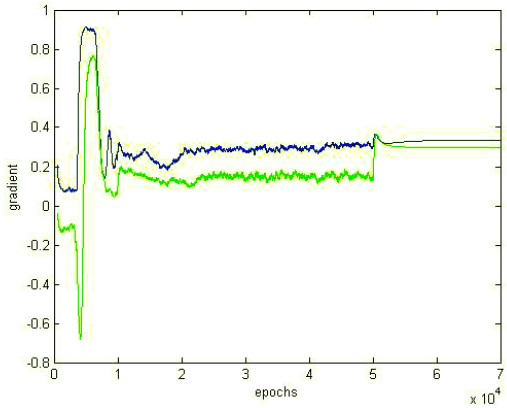
For the proposed model, Figure 5.3(a) shows the mean-squared classification error plot for the iris dataset. It takes some time for the critic network to model the reward function in order to provide an approximately correct gradient. There may be multiple solutions in the reward space of the critic network. Maximizing the reward function in the actor network might lead to going down towards a single solution and then flipping the class outputs if a new, better space is found via exploration. This might require redoing the gradient function. Thus, in the initial training stages, the



(a) MSE



(b) Reward



(c) Gradient Comparison

Figure 5.3: Proposed approach with incomplete target data. (a) Mean-squared error for the iris dataset, where the green line shows the validation error while the blue line shows training error. (b) Reward accrued over epochs. (c) Comparison of the approximated gradient vs the true gradient for a single sample and a single node over epochs, where the green line shows the approximated gradient from the critic network while the blue line shows the true gradient.

actor network performs poorly. However, once the critic network models the reward, the mean-squared reconstruction error from the actor network is minimized.

Figure 5.3(b) shows the reward from the actor network structure over epochs. This shows the jumps made by the critic network whenever a space with a higher reward is identified via the added noise exploration.

Figure 5.3(c) compares the gradient from the critic network to the true gradient of the traditional classifier network.

$$\frac{\partial Q(S_t, a'_{c_t})}{\partial a'_{c_t}} \approx \frac{\partial E}{\partial y}$$

The gradient for a single sample and a single node over the training epochs is shown. The green line shows the gradient learned by the critic network while the blue line shows the true gradient computed with the entire target dataset. Figure 5.3(c) shows that the critic network initially provides a wrong gradient, however, after some time approximates the true gradient closely even with the incomplete target dataset.

5.6 Proposed "Lateral" Autoencoder Sparsity

Sparsity in an autoencoder forces the feature activations to be as close to zero as possible. This is usually achieved over the activation of a node over all samples[66]. The average activation of a hidden unit j over the training set of m samples is defined as

$$\hat{\rho}_j = \frac{1}{m}\sum_{i=1}^{m}[a_j^h(x^{(i)})]$$

A sparsity parameter ρ is defined as the target average activation of each node. In order to achieve sparsity, an additional penalty term in the form of KL-divergence is thus used.

$$KL(\rho||\hat{\rho}_j) = \rho log\frac{\rho}{\hat{\rho}_j} + (1-\rho)log\frac{1-\rho}{1-\hat{\rho}_j}$$

The gradient computed from this function is then used as a regularization parameter in addition to the gradient backpropagated for minimizing the reconstruction error.

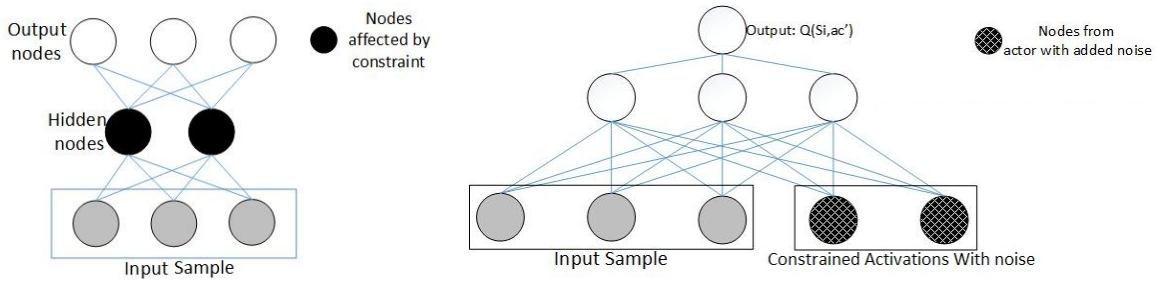
$$\delta^h = \sum_{i=1}^{n}(W_i^h\delta^o)f'(a^h) + \beta\left(-\frac{\rho}{\hat{\rho}_j} + \frac{1-\rho}{1-\hat{\rho}_j}\right)$$

Here f' is the derivative of the activation function and W are the weights.

The goal of sparsity is to represent the data using as few values as possible. However traditional KL sparsity does not enforce any conditions on the maximum number of activations allowable to reconstruct a given sample. Although it achieves a similar form of sparsity indirectly as most features may fire for a few samples and remain close to zero for the rest while still achieving the desired average activation, it looses the essence of dimensionality reduction. Multiple features firing at the same time indicate that some discriminative property of the data is divided among all the features in order to keep the average activation of each node as low as possible, thus features learned overlap over the dataset. This results in inefficient and redundant features. The resulting feature set is temporally sparse but lacks important structural insights of the data.

In order to learn non-overlapping discriminative and sparse features, it is often necessary to laterally inhibit features [74] [75]. However implementing such a constraint requires interconnections among nodes in the hidden layer. This results in additional computational complexity and a complex network architecture which requires the update of a node to include the activations of other nodes in the layer. In such architectures the networks have minimal lateral connections, usually between immediate neighbors, in order to reduce the complexity but also preserve some lateral inhibition [13]. The proposed model, however, allows us to impose a novel structural constraint where the output of a node is influenced by every other node in the hidden layer. This allows for a much stricter constraint, which is not possible for traditional methods for lateral inhibitions. This constraint therefore enforces as few high activations as possible in the entire hidden layer to reconstruct the sample as opposed to lateral inhibitions among immediate neighbors. In contrast to traditional sparsity which strives to achieve close to zero average activation for a given node over all samples, this form of sparsity tries to minimize the number of features required to

reconstruct a sample. This results in highly discriminative features and achieves the desirable property of locally expert features which fire while most other nodes remain close to zero.



(a) Actor network (b) Critic network

Figure 5.4: Sparse autoencoder networks. (a) Actor network, shaded grey nodes indicate input nodes while constraint applied on hidden nodes. (b) Critic network, with input and noisy hidden node activations as input

Figure 5.4 shows the adaptation of the Actor-Critic architecture to this problem. The nodes affected by the constraint are the hidden layer nodes of the autoencoder, $a_c = a_h$, as shown in Figure 5.4. A node is considered to be active if the activation value is higher than a threshold parameter ϕ. The constraint is then encoded as a system of penalties, where the penalty for sample S_t and activations a'_c is given by

$$R(S_t, a_c) = \sum_{i=1}^{n} [a'_{c_i} > \phi]$$

which is the number of nodes 'firing' for a given sample in the hidden layer with n nodes. In addition to the constraint being non-computable because of the lack of lateral connections among hidden nodes, the reward function is an L0 norm which is usually impossible to optimize directly. The critic network models a smooth approximation of the reward function which now has a gradient and can be optimized by the actor network. The gradient from the critic network output with respect to the

hidden layer activations is used as a regularization parameter similar to traditional sparse autoencoders.

$$\delta^c = \sum_{i=1}^{n} (W_i^c \delta^o) f'(a^c) + \theta \left(\frac{\partial Q(x, a'_c)}{\partial a'_c} \right)$$

Here f' is the derivative of the activation function, W are the weights, and x are the input training samples.

5.6.1 Experiments

Experiments were performed to compare the reconstruction accuracy and achievable sparsity of the proposed form of sparsity and traditional sparse methods using KL-divergence. The autoencoder network was trained with 70% of the number of input nodes as hidden layer. A larger network is needed to increase performance of the reconstruction task, however, the number of nodes for the hidden layer was chosen in order to demonstrate the validity of the proposed approach. The sigmoid function was used as activation functions of all nodes in the autoencoder. Since the penalty term is the number of nodes firing for a given sample, which can be more than one, softplus activation functions were used in the critic network for the hidden and output layers. The softplus activation function has the property of non-linear functions, whose output lies in the range of $(0, \infty)$.

Weight regularization is an important aspect of any sparse autoencoder system, since the activations can be learned to arbitrarily small values by simply learning higher weights. To force the system to learn proper sparse features, a weight reglarization penalty term was added to both models in order to keep the weights low. The model was tested on five real world problems, by dividing the data into training, validation and testing segments. Weight parameters $0.1 - 2$ as the range of θ and $0.2 - 16.8$ as β value ranges were used.

5.6.1.1 Abalone dataset

The abalone dataset is a highly imbalanced dataset with over 80% of the samples belonging to the same class. The dataset has 8 input features and 4,177 samples. The autoencoder network was trained with six hidden nodes.

5.6.1.2 White Wine dataset

The wine white dataset has 11 input features and 3,918 samples in the dataset. The autoencoder network was trained with eight hidden nodes.

5.6.1.3 Wine dataset

The wine dataset has 13 input features and 178 samples in the dataset. The autoencoder network was trained with 10 hidden nodes.

5.6.1.4 Glass dataset

The glass dataset has 9 input features and 214 samples in the dataset. The autoencoder network was trained with seven hidden nodes.

5.6.1.5 Thyroid dataset

The thyroid dataset has 21 input features and 7,200 samples in the dataset. The autoencoder network was trained with 15 hidden nodes.

The results of reconstruction and the minimum achievable sparsity for the problems are shown in Table 5.2. The proposed model is able to achieve better reconstruction accuracies for many cases compared to the traditional form of KL-divergence sparsity while achieving much more compact and sparser representations. For the proposed model, the first column represents the minimum sparsity achieved

	Proposed sparsity		Traditional sparsity	
Dataset	minimum sparsity achieved	MSE	minimum sparsity achieved	MSE
Abalone	0.1663	**0.0076**	0.4285	0.0119
Wine white	0.11	**0.0035**	0.4908	0.0037
wine	0.0972	**0.0089**	0.4514	0.0124
glass	0.00	0.0087	0.2716	**0.0086**
thyroid	0.01	0.0123	0.5495	**0.0118**

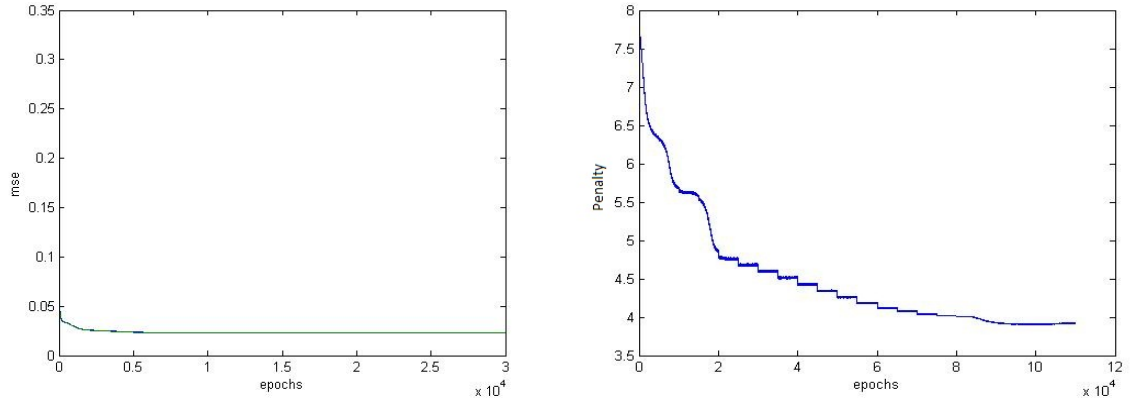
Table 5.2: Comparison of the two forms of sparsity

which is the percentage of times that a hidden node fires over the entire test sample set for every node, while the second column shows the mean-squared reconstruction error achieved for the sparsity. Similarly, the third and fourth column represent the minimum sparsity achieved and mean-squared reconstruction error for the traditional form of sparsity.

A separate experiment was done using the thyroid dataset, which has 21 input dimensions, with eight hidden nodes. This forces the autoencoder networks to learn some features with higher activation values since the feature vector is not over complete. This was done to compare the type of features learned by the two approaches.

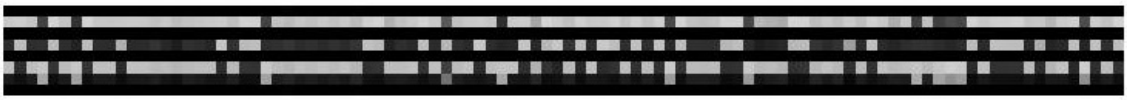
For the proposed model, Figure 5.5(a) shows the mean-squared reconstruction error plot for the thyroid dataset. Figure 5.5(b) shows the decrease in penalty of the actor network over epochs.

Figure 5.6 shows the features learned for 100 samples in the dataset. Figure 5.6(a), shows the features learned from the proposed approach while Figure 5.6(b) shows the features learned for the traditional sparse model. As shown in Figure 5.6(a) the features learned by the proposed model are much more discriminative as evident in features from node pairs (four, six) and (two, seven) where each pair never fires simultaneously.



(a) MSE (b) Penalty

Figure 5.5: Proposed form of sparsity. (a) Reconstruction mean-squared error for thyroid dataset (b) Penalty accrued over epochs



(a) Features learned by proposed lateral inhibitions approach

(b) Features learned by traditional sparsity constraint

Figure 5.6: Thyroid dataset features. x-axis represents sample activations, y-axis represents individual nodes, intensity represents value of activations of a node for the given sample with white being almost one while black denoting non-firing nodes. (a) Proposed lateral inhibition sparsity constraint approach (b) Traditional KL-divergence sparsity constraint approach

In contrast, the traditional sparse system as shown in Figure 5.6(b) learns features which fire together, i.e., the discriminative property of the dataset is divided into multiple features in order to keep the average activation value of each feature as low as possible. Figure 5.6(a) shows that the proposed approach is able to push down the activations of unneeded features, namely one, three, five and eight, towards zero in order to keep the number of activations per sample low.

5.7 Conclusion

An actor critic model is employed to solve problems without an inherent gradient. Policy gradient algorithms are used to estimate gradients for tasks with a non-differentiable or discontinuous performance metric. The gradient is learned from a system of rewards from the imposed constraints function and is subsequently used to train the actual network. A novel form of lateral inhibition sparsity is also proposed which learns sparse features using as few activations as possible to reconstruct a given sample. It does this without interconnections among nodes in the hidden layer which made it unusable earlier with regards to computational complexity. Experiments evaluated on real world data show that this form of lateral sparsity is able to reconstruct the data better and with a much sparser representation.

5.8 Future Work

Given the applicability of the proposed approach, further exploration is necessary in order to assess the performance of the proposed form of sparsity in deep neural networks.

CHAPTER 6

MDP Auto-encoder

6.1 Introduction

Humans possess an intuitive understanding of the physical dynamics of the world. For example, humans recognize correct dynamic behavior of planes flying and balls rolling. This allows predicting the outcomes of actions and solve complex tasks much faster, and would thus also be a useful ability in machine learning systems [76]. Learned models for the dynamics of the world can be used to simulate the environment and perform multi-step lookaheads for planning. Model-based reinforcement learning (RL) algorithms learn to model the observed data and then use it to solve the task. Such models predict future observations as an outcome of executing actions from the current observed state. Solving a task with the ability to predict the future outcomes of actions has a fundamental appeal for reinforcement learning. There is a vast variety of possible applications, from learning policies offline with a given model [77, 78], solving a new task with better sample efficiency [25], encouraging exploration in order to reduce model uncertainty [79], or using model predictability as an intrinsic reward to learn models for better planning [80, 79].

However, most model-based approaches rely on predicting the raw observations [76, 79, 81], which makes prediction very complex and might not be necessary to solve a given task. For example, when solving a task from observed images, much of the information in the image is irrelevant to the task at hand. Predicting the future frame of an image might require encoding the color of a rainbow colored car even if it behaves similar to other cars, irrespective of the color, when viewed in terms of

tasks that require driving. Furthermore, learning models which can predict the future outcomes while relying on current observations might be difficult when observations are not temporally and spatially correlated. Building an accurate prediction model for a large and complex observation space is often challenging. This shows the need for learning models that can extract information pertaining to the underlying dynamics and rewards of the system by encoding information that is essential in solving the task. This provides the ability to predict future rewards and identify the terminal states, without the need for predicting future observations. Such models can learn simpler representations for complex domains and are potentially more flexible when dealing with stochastic environments.

Thus solving a task requires learning accurate models for the dynamics of the system independent of the specific task such that it can potentially be transferred to other tasks. Such models need to address several key difficulties in encoding system dynamics in a latent space, and require the ability to capture multiple future outcomes and predict rewards and chance of failure or termination, while minimizing compounding errors from modeling inaccuracies for multi-step lookahead and planning.

This paper proposes a neural network architecture that utilize self-supervised methods in order to learn a latent transition model from observed samples. This work learns compressed models of the system dynamics by formulating the concept of Markov decision process (MDP) homomorphisms [82] as a variational inference problem. The method aggregates states that have similar rewards and transition dynamics. This approach learns a latent MDP model that can predict or generate the future rewards, identify termination or failure outcomes and can be used to simulate transitions in the latent space. Experiments performed on the *Cartpole* and *Acrobot* environments, show that the system is able to learn an efficient latent encoding and

model the dynamics of the latent representation with an equivalence to the underlying real world dynamics and is able to learn policies on these representations which can achieve close to the optimal performance.

6.2 Related Work

Most approaches for learning latent variable models for model-based RL involve the complex task of predicting the future observations, additionally conditioned on rewards and termination indicators, in order to infer the transition dynamics [76, 81]. However these approaches do not condition the representations on the predicted dynamics in latent space and thus fail to capture the transition dynamics of the observed system. The main distinction from such works is that the proposed approach learns to predict rewards and terminating states and captures the behavior of the observed system by modeling the transition dynamics in latent space.

There are approaches which model the transition dynamics directly without predicting the next observations. However, such systems require access to expert transitions or a computationally expensive nested gradient based planning module[78].

Another approach is to learn the dynamics in a latent representation using some form of value function decomposition. For example Value Prediction Networks (VPN)[83, 84] learn the models while minimizing the error in predicting the value function for a state. These approaches are similar to the proposed architecture. However, these methods learn models that depend upon the assumed hyperparameter defining the horizon γ, for solving a task. Thus, depending upon the value of γ these systems potentially learn different dynamics of the same observed model.

The proposed approach encodes the observed space irrespective of the choice of γ, while learning more compact representations. Using a variational inference based model, the method learns the dynamics of the abstract state space sufficient

for computing future rewards and identifying terminating states. Furthermore, the proposed network architecture learns the representations from randomly observed samples without requiring expert like trajectories, while still being able to utilize trajectories efficiently when available by augmenting an adversarial learning component with the inference model.

6.3 Markov Decision Processes

A *Markov Decision Process* is defined as a tuple $\langle S, A, \Psi, P, R, \beta \rangle$, where S is a finite set of states, A is a finite set of actions, $\Psi \subseteq S \times A$ is the set of admissible state-action pairs. $A_s = a|(s,a) \in \Psi \subseteq A$ defines the set of actions admissible in state s, assuming that $\forall s \in S$, A_s is non-empty. $P : \Psi \times S \mapsto [0,1]$ is the transition probability function with $P(s,a,s')$ being the probability of transition from state s to state s' under action a where $s, s' \in S$ and $a \in A_s$. $\beta : S \mapsto [0,1]$ is an attribute defining the special transition to an absorbing state. These states $\{s_\beta\} \in S$ form the set of special states with $\beta > 0$ which have a chance of termination of the current episode. $R : \Psi \mapsto \mathbb{R}$ is the expected reward function, with $R(s,a)$ being the expected reward for performing action a in state s.

6.3.1 MDP Homomorphisms

An *MDP homomorphism* [82, 85, 86] is the mapping from an MDP $M = \langle S, A, \Psi, P, R, \beta \rangle$ to an MDP $\widetilde{M} = \langle \widetilde{S}, \widetilde{A}, \widetilde{\Psi}, \widetilde{P}, \widetilde{R}, \widetilde{\beta} \rangle$ defined by a surjection, h, from Ψ to $\widetilde{\Psi}$, that is characterized by a tuple of surjections $\langle f, u_s | s \in S \rangle$, with $h((s,a)) = (f(s), u_s(a))$, where $f : S \mapsto \widetilde{S}$ and $u_s : A_s \mapsto A'_{f(s)}$ for $s \in S$, such that $\forall s, s' \in S, a \in A_s$:

$$P'(f(s), u_s(a), f(s')) = P(s, a, f^{-1}(f(s'))), \tag{6.1}$$

$$P'(\beta_{f(s)}|f(s)) = P(\beta_s|s), \tag{6.2}$$

$$R'(f(s), u_s(a)) = R(s, a) \tag{6.3}$$

An MDP homomorphism thus maps an MDP onto a different MDP which is potentially more abstract but preserves the transition and reward dynamics of the original system. Strict requirements of Equations (6.1) and (6.3) are often infeasible for equivalence relations in probabilistic systems. To address such issues, approximate homomorphisms are formulated, [82], which allow aggregating states that are not exactly equivalent. This allows formulating approximate constraints:

$$K_r = \max_{\substack{s\in S \\ a\in A_s}} |R(s, a) - \widetilde{R}(f(s), u_s(a))| \tag{6.4}$$

$$K_\beta = \max_{\substack{s\in S \\ a\in A_s}} |P(\beta'_s|s') - \widetilde{P}(\beta_{f(s')}|f(s))| \tag{6.5}$$

$$K_p = \max_{\substack{s,s'\in S \\ a\in A_s}} |P(s, a, f^{-1}(f(s'))) - \widetilde{P}(f(s), u_s(a), f(s'))| \tag{6.6}$$

Learning representations $f(s)$ and $u_s(a)$ while minimizing the divergence in Equations (6.4) and (6.6) encodes approximate equivalence relations between the observed transitions and latent transitions. This allows bounding the value function $V^*(s_k)$ with respect to the latent value function $\widetilde{V}^*(f(s_k))$ for the given task with respect to K_p and K_r.

$$|\widetilde{V}^*(f(s_k)) - V^*(s_k)| \leq \frac{2K_p}{K_r(1-\gamma)}$$

Previous works form the reduced homomorphic image of the observed Markov decision process by iteratively partitioning and fine tuning the space until convergence [87, 88, 89, 90, 91]. These approaches initially partition the space by identifying all termination states which are defined as states with a special transition, after which the episode ends. These states are defined as s_β which are goal or termination states.

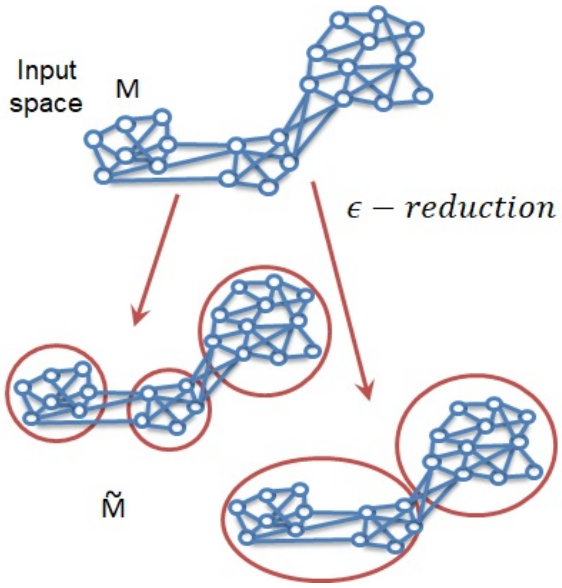
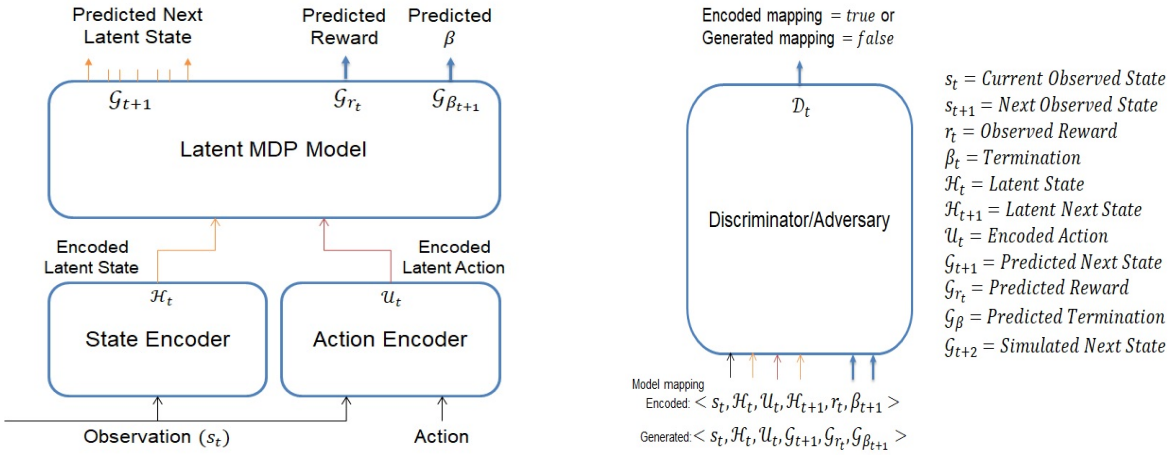


Figure 6.1: Mapping MDP M to reduced MDP model M', where blocks shown as circles denote the aggregated states

Following this, the resultant space is partitioned with respect to reward equivalence, where states aggregated together share a similar reward function(bounding the reward residual from Equation (6.4)). Finally, the set of partitions are fine tuned with respect to the corresponding observed state and partition space transition dynamics (bounding the transition dynamics residual in Equation (6.6). Figure 6.1 shows the process of performing ϵ-reduction and aggregating observed states to form the representation in partition space M'.

In contrast to this iterative construction approach that relies on pre-defined residual bounds, ϵ_r, ϵ_t, the proposed approach formulates the problem of ϵ-reduction into a variational inference problem, and applies an autoencoder architecture to directly learn the latent Markov decision process model from observed samples.



(a) Variational MDP Autoencoder Model (b) Discriminator Model

Figure 6.2: Proposed model architecture. (a) Set of encoding networks for state and action, generative model of MDP in latent space. (b) Discriminator network differentiates between the true and learned model mapping.

6.4 Latent Variable MDP Models

In order to learn a reduced Markov decision model, the proposed architecture uses an adversarial neural network architecture to jointly model the latent variable model in order to minimize the $\epsilon - loss$ as a combined metric from Equations (6.4), (6.5) and (6.6):

$$\begin{aligned} \epsilon - loss = |pr(h_{k+1}|s_k, a_k) - pr(h_{k+1}|h_k, u_k)| + \\ |pr(\beta_{k+1}|s_{k+1}) - pr(\beta_{k+1}|h_k, u_k)| + \\ |(r_k|s_k, a_k) - (r_k|h_k, u_k)| \end{aligned} \tag{6.7}$$

This architecture performs a gradient based approximate $\epsilon - reduction$ in order to learn reduced MDP models using observed trajectory segments from the true model. As shown in Figure 6.2, the latent MDP model predicts the next latent state $h_{k+1} \in \widetilde{S}$, reward $r_k \in R$ and the termination indicator $\beta_{k+1} \in \beta$, as a joint dynamics model of the data conditioned over the latent variables $h_k \in \widetilde{S}, u_k \in \widetilde{A}$.

The proposed architecture is inspired by the VAE/GAN [92] architecture, which is a combination of *variational autoencoders* (VAE)[93] and *Generative Adversarial Networks* (GAN)[94]. The model is comprised of a set of component neural networks as shown in Figure 6.2(a,b).In particular, the variational autoencoder portion consists of a set of encoder networks that encode a data sample x to a latent representation z, and a generator network that models the transitions in latent space. The discriminator portion consists of an adversarial network that discriminates between real and generated transition and reward samples in latent space.

6.4.1 Architecture

The method combines the advantage of GAN as a generative model and VAE as a method that produces an encoder of data into the latent space. *Variational autoencoders'* decoupled nature in terms of the latent space allows generating reconstructed data as well as simulating new data. Unlike traditional *variational autoencoder* architectures where the generator model is used to reconstruct the input data sample from the latent representation, the proposed architecture modifies the generator and discriminator networks in order to reconstruct the underlying transition dynamics of the system instead.

6.4.1.1 Discriminator

Let X be the set of true samples and X', X'_p be the set of fake and simulated samples, respectively. The GAN objective is to find the binary classifier that gives the best possible discrimination between true and generated or simulated data, optimizing with respect to the binary cross entropy, $\mathcal{L}_{GAN}$.

$$\mathcal{L}_{GAN} = log(Dis(X)) + log(1 - Dis(X')) + log(1 - Dis(X'_p)) \tag{6.8}$$

6.4.1.2 State & Action Encoder

As shown in 6.2(a), the state and action encoder networks $(\mathcal{H};\theta_{\mathcal{H}},\mathcal{U};\theta_{\mathcal{U}})$ parameterized $\theta_{\mathcal{H}},\theta_{\mathcal{U}}$ learn the mapping functions $\mathcal{H}: S \mapsto \widetilde{S}$ and $\mathcal{U}: S \times A \mapsto \widetilde{A}$ which encode data from observed space $\mathcal{M}$ to latent space $\widetilde{\mathcal{M}}$. Depending upon the type of data, different network architectures can be used for encoding the observed space. For example, convolutional neural networks (CNN) can be used to encode observed space represented by image data or recurrent neural networks (RNN) for partially observable systems. For an observed sample $\langle s_k, a_k, s_{k+1}, r_k, \beta_k \rangle$, s_k, s_{k+1} denote the current observed state and the next state by executing action a_k, while observing reward $r_k \in \mathcal{R}$. The termination flag β_k records whether the system terminated after timestep k. Let h_k, h_{k+1} denote the encoded representation of the current state s_k and the next state s_{k+1} respectively. The action encoder allows encoding the action space with respect to the current state, where u_k defines the encoded representation of action a_k when applied in state s_k. These set of encoders together, map the observed model onto a meaningful latent space.

6.4.1.3 Generator

The generator network as shown in Figure 6.2(a), learns a latent MDP model of the observed system and models the function $\mathcal{G};\theta_{\mathcal{G}}: \widetilde{S} \times \widetilde{A} \mapsto (\widetilde{S}, \widetilde{R}, \widetilde{\beta})$ parameterized by $\theta_{\mathcal{G}}$. This function predicts the next latent state $g_{k+1}|(h_k, \{u_k\})$, reward g_{r_k} and terminal, goal or failure conditions g_{β_k}, from the input latent state and action. Here, $g_{k+1}|(h_k, \{u_k\})$, g_{r_k}, g_{β_k} denote the predicted dynamics of the system at timestep $k+1$, with respect to the encoded state and action h_k and u_k for a sample in some timestep k.

The generated latent state g_{k+1} is used as a prediction of the next latent state and shares the same space as that of h_k. This is used to enforce the *MDP homomorphic* constraints on the encoder network such that the encoded representations agree with the generated predictions and vice versa. Here the generator function $\mathcal{G}$ and encoder functions $(\mathcal{H},\mathcal{U})$ are constrained by each other in a *self-supervised* fashion. This constraint forces the encoding to be consistent and be able to predict the respective reward and identify absorbing states. Forcing the generator to predict the reward g_{r_k} and terminations flag g_{β_k} allows the latent representation to be task-specific and keeps the adversarial system from collapsing to the trivial solution where all observed states are mapped to a single latent state.

Such a generator model is an important tool that is useful for offline simulations, where the predicted next state is used as the input state for predictions. Staring from a given state encoding and applying a sequence of actions, virtual rollouts allow predicting the forward dynamics of the system. Here, $g_{i+N}|(h_i, \{u_i, u_{i+1}, ..., u_{i+N}\})$ denotes the predicted state in latent space at step $i+N$ while applying the sequence of actions $u_i, ..., u_{i+N}$ from the initial state h_i and simulating onwards from the i^{th} timestep for N steps.

6.4.2 Supervised Learning

The system forms a latent representation in a *self-supervised* method by conditioning the latent dynamics model on the encoded state h_k and action u_k where the next observed state s_{k+1}, reward r_k and termination condition β_{k+1} depend upon the current state s_k. Equation (6.7) can then be rewritten as the $\mathbb{KL}$-divergence between the the true model posterior $p_\phi(h_k, u_k|h_{k+1}, r_k, \beta_{k+1})$ and the approximated posterior using the encoder networks $q_{\theta_\mathcal{H},\theta_\mathcal{U}}(h_h, u_k|s_k, a_k)$. This formulation models the true posterior using an approximation, while observing the latent encoding of the next

observed state h_{k+1}, observed reward r_k and whether the episode terminated at this state β_{k+1} as evidence in order to update the latent variable.

$$\begin{aligned} &\epsilon{-}loss = \\ &\quad \mathbb{KL}(q_{\theta_{\mathcal{H}},\theta_{\mathcal{U}}}(h_h, u_k|s_k, a_k)||p_\phi(h_k, u_k|h_{k+1}, r_k, \beta_{k+1})) \end{aligned} \tag{6.9}$$

This formulates the transition dynamics as a latent variable model by updating the uncertainty given new evidence of the encoded next state, and the reward and termination conditions. However, this formulation is intractable and requires integrating over the evidence across all possible latent representations of a state and the encoding of the corresponding transition to the next observed states conditioned on the executed action. In order to address this, [93] approximate the variational inference by optimizing the Evidence Lower BOund ($ELBO$) loss function instead, which is a differentiable approximation bounding the log likelihood of the evidence. Maximizing the $ELBO$ loss is equivalent to minimizing the $\mathbb{KL}$-divergence. Using this approximation the network parameters $\theta_{\mathcal{H}}, \theta_{\mathcal{U}}, \theta_{\mathcal{G}}$ can now be updated by defining the $\mathcal{L}_{supervised}$ loss.

$$\begin{aligned} &\mathcal{L}_{supervised} = \\ &\big(\mathbb{E}q_{\theta_{\mathcal{H}},\theta_{\mathcal{U}}}(h_h, u_k|s_k, a_k) \\ &\qquad [log\ \mathcal{G}_{\theta_{\mathcal{G}}}(g_{k+1}, g_{r_k}, r_{\beta_{k+1}}|h_k, u_k)]\big) - \\ &\qquad \Big(\mathbb{KL}(\mathcal{H}_{\theta_{\mathcal{H}}}(h_k|s_k)||p(h)) + \\ &\qquad\quad \mathbb{KL}(\mathcal{G}_{\theta_{\mathcal{G}}}(g_{k+1}|h_k, u_k)||p(h)) + \\ &\qquad\quad \mathbb{KL}(\mathcal{U}_{\theta_{\mathcal{U}}}(u_k|s_k, a_k)||p(u))\Big) \end{aligned} \tag{6.10}$$

This equation can generally be viewed as having two component terms, where, the first term is the reconstruction loss while the remaining elements regularize the network parameters $\theta_{\mathcal{H}}, \theta_{\mathcal{U}}, \theta_{\mathcal{G}}$ with respect to a prior on the latent variables. The $\mathcal{L}_{supervised}$ loss formulates the lower bound of the proposed $\epsilon{-}loss$, i.e. on the divergence between

the latent model dynamics with respect to the true model behavior. Furthermore, minimizing the $\epsilon - loss$ also bounds the loss when subsequentlylearning the value function from the encoded space with respect to ϵ and γ.

Reparameterization Trick

Training this architecture requires backpropagation with respect to the random variables h_k, u_k, which is not possible. In order to address this issue a reparameterization trick is applied [93] that involves injecting noise into the approximated latent model and allowing backpropoagation through the networks and bypass the sampling process. This reformulation allows the model to depend on a deterministic encoding with a stochastic noise $\mathcal{N}$ as an auxillary variable . This approach can be generalized to a variety of latent priors, which meet certain conditions. Latent models with Gaussian priors have been used for encoding images, using noise from a normal distribution with unit variance, $\mathcal{N}(0, \mathcal{I})$. Other forms of reparameterizations, include learning binary representations using stochastic *sigmoid* representations.

The proposed approach learns discrerte representations for the latent model using a *softmax* layer and applies the *Gumbel-softmax* trick [95, 96] to efficiently sample from a softmax distribution. This classifies the observed space into discrete blocks and is similar to predicting actions in traditional reinforcement learning algorithms[35].

This network performs a gradient descent based approximate ϵ-reduction, with a valid choice of the representation prior, training the variational autoencoder in a *self-supervised* fashion.

6.4.3 Adversarial Learning

The system uses an adversarial component $\mathcal{D}$, in order to derive gradients for virtual rollouts. This discriminator network parameterized by $\theta_{\mathcal{D}}$, learns the similarity function between model mappings $\mathcal{D} : M \times \widetilde{M} \mapsto [0, 1]$.

6.4.3.1 Samples

Given a set of observed samples $o_k \in \mathcal{O}$, where $o_k = \langle s_k, a_k, s_{k+1}, r_k, \beta_k \rangle$. The encoder maps the observed space onto a latent variable model while the generator tries to model the transition dynamics of the encoded representations in a self-supervised fashion. The proposed architecture derives its losses by discriminating between true, fake and simulated samples.

6.4.3.1.1 True Samples The state and action encoders $\mathcal{H}, \mathcal{U}$ are used to create the set of samples, $x_k \in X$, which are assumed *true* by the discriminator $\mathcal{D}$

$$x_k = \langle s_k, h_k, u_k, h_{k+1}, r_k, \beta_k \rangle \tag{6.11}$$

Here $h_k = \mathcal{H}(s_k)$, $h_{k+1} = \mathcal{H}(s_{k+1})$ denote the latent representations of the observed states $s_k, s_{k+1} \in o_k$. The action encoder $\mathcal{U}$ is used to map the observed action to latent action space $\widetilde{A}$ with respect to the current state s_k, where $u_k = \mathcal{U}(s_k, a_k)$.

6.4.3.1.2 Fake Samples The generative model $\mathcal{G}$ is used to create the set of samples $x'_k \in X'$, which are assumed *false* by the discriminator $\mathcal{D}$

$$x'_k = \langle s_k, h_k, u_k, g_{k+1} | (h_k, u_k), g_{r_k}, g_{\beta_k} \rangle \tag{6.12}$$

Here $g_{k+1}, g_{r_k}, g_{\beta_k} = \mathcal{G}(h_k, u_k)$ is the predicted next latent state as modeled by the generator $\mathcal{G}$. The generator network also models the expected reward g_{r_k} and the chance of termination g_{β_k} upon simulating action u_k at state h_k.

The goal of the autoencoder architecture with encoder networks ($\mathcal{H}$,$\mathcal{U}$) and the generator network $\mathcal{G}$ is to learn a predictable model which is equivalent to the observed mapping such that the discriminator $\mathcal{D}$ is unable to distinguish between them. Comparing the tuple of fake samples X' to that of true samples X, for a given observed state $s_k \in \mathcal{S}$ and action encoding $u_k \in \mathcal{U}$, the variables $g_{k+1}, g_{r_k}, g_{\beta_{k+1}}$ which represent the predicted dynamics in latent space should have the same distribution equivalent to the observed dynamics of the encoded space. Thus the gradients from the discriminator network $\mathcal{D}$ is used to further constrain the autoencoder architecture with respect to the learned distribution by imposing $g_{k+1} \approx h_{k+1}, g_{r_k} \approx r_k, g_{\beta_{k+1}} \approx \beta_{k+1}$. This enforces the conditions proposed in the MDP homomorphism framework in addition to the supervised training. Furthermore, in order to discriminate between the encoded and generated model mapping, the discriminator network $\mathcal{D}$ learns a similarity function between observed states s_k and the corresponding step in the mapped model which is beneficial for planning tasks on the latent space.

6.4.3.1.3 Simulated Samples The adversarial component allows computing gradients for virtual rollouts or lookaheads. Utilizing single step segments from the observation model M with samples of the form $o_k = \langle s_k, a_k, r_k, s_{k+1}, \beta_{k+1} \rangle$, one step lookaheads can be generated using the latent MDP model. These samples are generated by using the predicted next latent state g_{k+1} as input to the latent MDP network $\mathcal{G}$, and predict the latent state g_{k+2} following state at timestep $k+1$. The generative model $\mathcal{G}$ is used to create the set of samples $x'_{p_k} \in X'_p$, which are assumed *false* by the discriminator $\mathcal{D}$. Here u_p is a latent action $u_p \sim \widetilde{A}$ randomly sampled from the set of

possible latent actions, or the encoding of a random action $a_p \sim A$, $u_p = \mathcal{U}(s_{k+1}, a_p)$ with respect to the observed state.

$$
\begin{aligned}
& x'_{p_{k+1}} = \\
& \langle s_{k+1}, g_{k+1} | (h_k, u_k), u_p, \\
& \qquad g_{k+2} | (h_k, \{u_k, u_p\}), g_{r_{k+1}}, g_{\beta_{k+1}} \rangle
\end{aligned}
\tag{6.13}
$$

Here $g_{k+2}, g_{r_{k+1}}, g_{\beta_{k+1}} = \mathcal{G}(g_{k+1}, u_p)$

6.4.3.1.4 Latent Overshooting with Trajectories Although Experiments were performed by training the architecture using $Markov$ samples, without exploiting expert trajectories. However, trajectories can be utilized efficiently when available by assuming the Markov property and comprehensively generating samples for each step performed in multi step lookaheads. These samples can be used to minimize compounding errors over longer horizons. For a given trajectory of length $N + 1$, where $\forall t \in \{0, ..., N + 1\}$

$$
\begin{aligned}
& X'_{traj} = \\
& \langle s_{t+N}, g_{t+N} | (h_t, \{u_t, .., u_{t+N-1}\}), u_{t+N}, \\
& \qquad g_{t+N+1} | (h_t, \{u_t, .., u_{t+N}\}), g_{r_{t+N}}, g_{\beta_{t+N}} \rangle
\end{aligned}
$$

6.4.4 Cost Functions

6.4.4.1 Adversarial Loss

The adversary is a classification network using samples $M \times \widetilde{M} \rightarrow [0, 1]$ and needs to distinguish between the true samples from the generated ones. The loss is

defined as $\mathcal{L}_{GAN}$, using the hyperparameter λ in order to normalize the losses with respect to the true samples, and the fake or simulated samples.

$$\begin{aligned}\mathcal{L}_{\mathcal{GAN}} = log(Dis(X))+ \\ \lambda(log(1 - Dis(X')) + log(1 - Dis(X'_p)))\end{aligned} \tag{6.14}$$

This function is used to update the network parameters of the discriminator, $\theta_{\mathcal{D}} \overset{+}{\leftarrow} -\nabla_{\theta_{\mathcal{D}}}(\mathcal{L}_{GAN})$, parameterized by $\theta_{\mathcal{D}}$.

The $\mathcal{L}_{GAN}$ loss is used to train the state, action encoders and the generative model. In addition to this, the divergence in the representations learned for true and fake samples by the discriminator is used to provide information about the reconstruction loss for the autoencoder networks. Here $\mathcal{L}_{llike}$ is defined as the loss in representations with respect to some layer $l \in \mathcal{D}$

$$\mathcal{L}_{llike} = -\mathbb{E}_{q(h_k,u_k|s_k,a_k)}[log\ p(\mathcal{D}_l(h_k, u_k)] \tag{6.15}$$

6.4.4.2 Supervised Loss

The combined losses from the adversarial network and self-supevised losses are used to update the autoencoder networks, the state action encoders ($\mathcal{H}$,$\mathcal{U}$) and the generative latent model $\mathcal{G}$.

$$\mathcal{L}_{vae} = \mathcal{L}_{supervised} + \mathcal{L}_{llike} - \mathcal{L}_{GAN} \tag{6.16}$$

Algorithm 3 shows the process of training the complete architecture using samples of observed data $\mathcal{O}$. Using a random batch of samples, the algorithm first computes the latent encodings of the observed state, next state, and action. The generator module is then used to simulate the dynamics using the generative network. These samples and their respective representations in latent space are used to generate true, fake and simulated samples as in Equations (6.11), (6.12), and (6.13). Finally the net-

Algorithm 3 Training the MDP autoencoder

Data: Dataset $o_k \in \mathcal{O}$, where $o_k = \langle s_k, a_k, r_k, s_{k+1}, \beta_{k+1} \rangle$

Result: Reduced Markov model $\widetilde{M}$

Initialize network parameters $\theta_{\mathcal{H}}, \theta_{\mathcal{U}}, \theta_{\mathcal{G}}, \theta_{\mathcal{D}}$

while *not done* **do**

$\{o_k\} \leftarrow$ Random minibatch from dataset $\mathcal{O}$,

$\quad$ *where* $\{o_k\} = \{\langle s_k, a_k, r_k, s_{k+1}, \beta_{k+1} \rangle\}$

$\{h_k\}, \{h_{k+1}\} \leftarrow \mathcal{H}(\{s_k\}), \mathcal{H}(\{s_{k+1}\})$

$\{u_k\} \leftarrow \mathcal{U}(\{s_k, a_k\})$

$\{g_{k+1}, g_{r_k}, g_{\beta_{k+1}}\} \leftarrow \mathcal{G}(\{h_k, u_k\})$

$X, X', X'_p \leftarrow$ Generate the set of true, fake and simulated samples for adversarial learning using eqns (6.11, 6.12, & 6.13)

Compute losses $\mathcal{L}_{vae}, \mathcal{L}_{GAN}$ using eqns (6.14, 6.16) Perform updates using the gradient w.r.t. the network parameters $\theta_{\mathcal{H}}, \theta_{\mathcal{U}}, \theta_{\mathcal{G}}, \theta_{\mathcal{D}}$

$\theta_{\mathcal{H}} \overset{+}{\leftarrow} -\nabla_{\theta_{\mathcal{H}}}(\mathcal{L}_{vae})$

$\theta_{\mathcal{U}} \overset{+}{\leftarrow} -\nabla_{\theta_{\mathcal{U}}}(\mathcal{L}_{vae})$

$\theta_{\mathcal{G}} \overset{+}{\leftarrow} -\nabla_{\theta_{\mathcal{G}}}(\mathcal{L}_{vae})$

$\theta_{\mathcal{D}} \overset{+}{\leftarrow} -\nabla_{\theta_{\mathcal{D}}}(\mathcal{L}_{GAN})$

end

$\widetilde{M} \leftarrow \mathcal{G}$

work parameters $\theta_{\mathcal{H}}, \theta_{\mathcal{U}}, \theta_{\mathcal{G}}, \theta_{\mathcal{D}}$, are updated as $\theta_{\mathcal{H}} \overset{+}{\leftarrow} -\nabla_{\theta_{\mathcal{H}}}(\mathcal{L}_{vae})$, $\theta_{\mathcal{U}} \overset{+}{\leftarrow} -\nabla_{\theta_{\mathcal{U}}}(\mathcal{L}_{vae})$, $\theta_{\mathcal{G}} \overset{+}{\leftarrow} -\nabla_{\theta_{\mathcal{G}}}(\mathcal{L}_{vae})$ with respect to the gradient of the loss functions $\mathcal{L}_{vae}, \mathcal{L}_{GAN}$.

6.5 Experiments

This section evaluates the proposed approach to learn latent variable dynamics models and evaluates the performance from the policy learned with the encoded observations. The algorithms is evaluated on the *CartPole* and *Acrobot* problems from OpenAI gym [97].

6.5.0.1 CartPole-v0

This environment emulates the classic cart-pole system defined in [25]. The state space of this environment is defined as a tuple with four values defining the current *cart position, cart velocity, Pole angle* and *pole velocity at the tip*. The system consists of two discrete actions, *push cart left* and *push cart right*. Reward is 1 for every step taken, including the termination step. The system starts in a random state with the pole close to the upright position. An episode terminates when the *pole angle* is more than 12 *degrees*, or *cart position* is more than 2.4 where the center of the cart reaches the edge of the allowed space. The task is to balance a pole with one end attached to a cart. The task is considered solved when the average reward is greater than or equal to 195.0 over 100 consecutive trials.

6.5.0.2 Acrobot-v1

The acrobot system in comprised of two joints and two links, where the joint between the two links is actuated. The state space of this environment consists of the $sin()$ and $cos()$ of the two rotational joint angles θ_1, θ_2 and their respective angle velocities $\dot{\theta}_1$, $\dot{\theta}_2$. Thus the observed state is defined as a tuple with six values [$cos(\theta_1)$, $sin(\theta_1)$, $cos(\theta_2)$, $sin(\theta_2)$, $\dot{\theta}_1$, $\dot{\theta}_2$]. The action space allows applying a $+1$, 0 or -1 torque on the joint between the two pendulum links. Reward is -1 for every step

taken, except the termination step. The task is considered solved when the average reward is greater than -60 over 100 consecutive trials.

6.5.1 Training Procedure

Experiments were performed by training the architecture offline using one step samples from the environments. For simplicity latent actions $u_k \in \mathcal{U}$ is mapped to actions in the observed MDP M, $u_k = a_k$ where $a_k \in \mathcal{A}$.

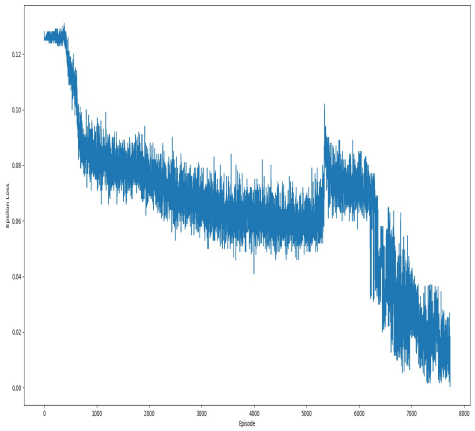
6.5.1.1 Network Architectures

Experiments in both cases were performed with the proposed set of encoders $(\mathcal{H};\theta_{\mathcal{H}},\mathcal{U};\theta_{\mathcal{U}})$, generator network $(\mathcal{G};\theta_{\mathcal{G}})$, and the discriminator network $(\mathcal{D};\theta_{\mathcal{H}})$. These networks have a similar architecture, with *two* hidden layers and 150 hidden nodes in each layer. A softmax layer is used to learn a categorical encoding of the observed states, while sigmoid activations were used for the reward and termination outputs. Figure 6.3(b,c) show the scores achieved by the policy trained over the encoded representations. The latent policy is trained and evaluated from scratch, every 5000 steps of training. The policy network is designed with *two* hidden layers and 50 nodes in each layer.

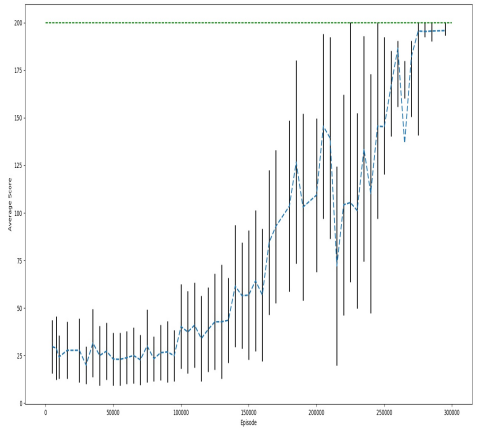
6.5.1.2 Performance

$\epsilon - loss$ is a measure of divergence between the true dynamics of the observed space and the learned dynamics model. Figure 6.3(a), shows the $\epsilon - loss$ being minimized for the CartPole problem, with 20 discrete latent states.

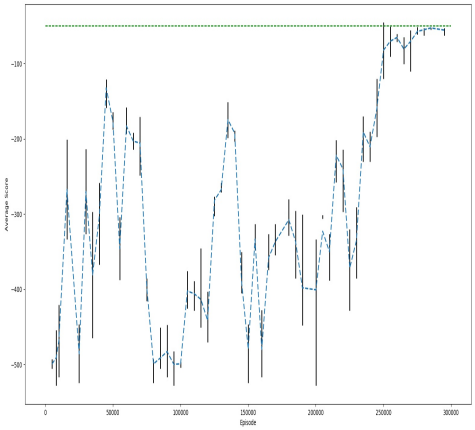
Figure 6.3(b) shows the scores achieved by the policy learned over the learned latent model for the CartPole problem, while figure 6.3(c) shows the scores for the latent policy for the Acrobot environment. These results show that the learned latent



(a) $\epsilon - loss$ for CartPole

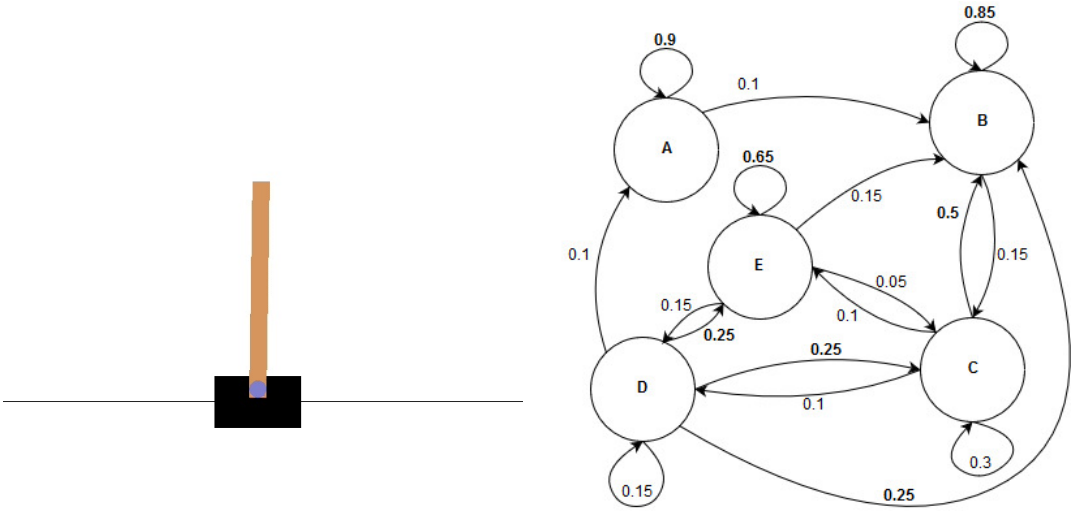


(b) Latent CartPole policy scores



(c) Latent Acrobot policy scores

Figure 6.3: Rewards achieved by policy defined over latent space. (a) $\epsilon - loss$ for reducing CartPole problem to 40 discrete latent states. (b) Rewards over latent policy for solving the CartPole environment using 40 discrete latent states. (c) Rewards over latent policy for solving the Acrobot environment using 40 discrete latent states



(a) Cartpole Frame

(b) Transition dynamics of the learned latent space

Figure 6.4: Mapping continuous observed space to discrete graphical models. (a) Sample frame for the Cartpole environment, where the state space is defined with $four$ continuous variables and two discrete actions. (b) Transition dynamics of the learned latent space with $five$ discrete blocks and two actions. Transitions for a single action is shown.

model is able to encode the underlying dynamics of the system and permits to learn policies with respect to the encoded space that can achieve scores close to the optimal policy conditioned on the observed space.

To investigate the effect of the number of blocks in the latent representation on the quality of the latent model, Table 6.1 shows the performance of policies learned on representations of different complexity, illustrating that increasing the number of blocks results in better performance for both problems.

Name	5 blocks	10 blocks	20 blocks	40 blocks	Optimal
CartPole-v0	$+140 \pm 55$	$+180 \pm 20$	$\mathbf{+195 \pm 5}$	$\mathbf{+195 \pm 5}$	$\mathbf{+195 \pm 5}$
Acrobot-v1	-350 ± 100	-150 ± 50	-70 ± 10	$\mathbf{-55 \pm 5}$	$\mathbf{-55 \pm 5}$

Table 6.1: Performance for different block numbers in the latent MDP model

To further illustrate the nature of the latent MDP model, the resulting model for the CartPole system with 5 blocks was explicitly extracted and represented graphically. Figure 6.4(a) shows a state of the cart pole environment, while Figure 6.4(b) shows a graphical MDP representation of the corresponding transition dynamics of the learned latent model. Here the cartpole environment is encoded into $five$ discrete *blocks*. The results in the previous table show that even on this simple representation the system is able to learn a relatively good policy with an average utility of 140.

6.6 Conclusions

The framework proposed here automatically learns and optimizes MDP homomorphisms by introducing a novel architecture which is able to simultaneously learn the dynamics of abstract, latent states that can make predictions of future rewards and termination or failure states rather than future observations. Our empirical evaluations shows that the proposed approach learns the underlying dynamics of the system and the policy over the learned latent space is capable of achieving scores similar or close to the maximum achievable value.

CHAPTER 7

Conclusions

The presented work shows various methods for applying algorithms from the framework of reinforcement learning to traditional machine learning problems such as classification, clustering, and general representation learning. Experiments show that such algorithms are useful for extracting domain dependent abstractions which can be applied to solve new tasks more efficiently. Furthermore, reinforcement learning algorithms are suitable for problems with sparse rewards which allow deriving approximate gradients for complex non-differentiable objectives.

7.1 Contributions

The presented work uses reinforcement learning algorithms in thecontext of various machine learning problems. Chapter 3 applies reinforcement learning to extract information for designing neural network architectures for classification problems, even in the context of streaming data or concept drift. The design of a neural network is defined by *three* MDP systems which are used to incrementally build ensemble neural network classifiers. Chapter 4 applies reinforcement learning to learn a generic policy to build feature mappings for constrained clustering tasks. For a given dataset, this system learns a dynamic feature mapping process that automatically morphs the observed space to a new feature space in order to satisfy the provided constraints.

Chapter 5 and 6 apply reinforcement learning in order to guide representations learned by neural networks. Chapter 5 utilizes reinforcement learning to provide gradients for non-differentiable objectives. This allows learning complex feature rep-

resentations which can encode hierarchical relationships within the same layer of representations. Finally, Chapter 6 applies reinforcement learning in order to provide bounds on the forward dynamics of a learned representation. This allows learning a powerful representation which forms a predictive MDP model which is behaviorally grounded to the observed model.

These algorithms show the power of using reinforcement learning in the context of building and tuning neural network and deep learning architectures in situations where task objectives are not easily representable or where completely labeled data is hard to obtain.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[2] W. N. Street and Y. Kim, "A streaming ensemble algorithm (sea) for large-scale classification," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2001, pp. 377–382.

[3] A. Tsymbal, "The problem of concept drift: definitions and related work," *Computer Science Department, Trinity College Dublin*, vol. 106, no. 2, 2004.

[4] K. L. Wagstaff, S. Basu, and I. Davidson, "When is constrained clustering beneficial, and why?" *Ionosphere*, vol. 58, no. 60.1, pp. 62–3, 2006.

[5] J. Hernández-González, I. Inza, and J. A. Lozano, "Weak supervision and other non-standard classification problems: a taxonomy," *Pattern Recognition Letters*, vol. 69, pp. 49–55, 2016.

[6] M. Belkin, I. Matveeva, and P. Niyogi, "Regularization and semi-supervised learning on large graphs," in *COLT*, vol. 3120. Springer, 2004, pp. 624–638.

[7] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the royal statistical society. Series B (methodological)*, pp. 1–38, 1977.

[8] S. S. Rangapuram and M. Hein, "Constrained 1-spectral clustering," *arXiv preprint arXiv:1505.06485*, 2015.

[9] S. Basu, M. Bilenko, and R. J. Mooney, "A probabilistic framework for semi-supervised clustering," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004, pp. 59–68.

[10] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[11] Z.-H. Zhou, J. Wu, and W. Tang, "Ensembling neural networks: many could be better than all," *Artificial intelligence*, vol. 137, no. 1, pp. 239–263, 2002.

[12] J. Platt *et al.*, "Sequential minimal optimization: A fast algorithm for training support vector machines," 1998.

[13] A. D. Szlam, K. Gregor, and Y. L. Cun, "Structured sparse coding via lateral inhibition," in *Advances in Neural Information Processing Systems*, 2011, pp. 1116–1124.

[14] A. Mahajan and T. Tulabandhula, "Symmetry learning for function approximation in reinforcement learning," *arXiv preprint arXiv:1706.02999*, 2017.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[16] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[17] T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. Černockỳ, "Strategies for training large scale neural network language models," in *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*. IEEE, 2011, pp. 196–201.

[18] J. Ma, R. P. Sheridan, A. Liaw, G. E. Dahl, and V. Svetnik, "Deep neural nets as a method for quantitative structure–activity relationships," *Journal of chemical information and modeling*, vol. 55, no. 2, pp. 263–274, 2015.

[19] M. Helmstaedter, K. L. Briggman, S. C. Turaga, V. Jain, H. S. Seung, and W. Denk, "Connectomic reconstruction of the inner plexiform layer in the mouse retina," *Nature*, vol. 500, no. 7461, p. 168, 2013.

[20] H. Y. Xiong, B. Alipanahi, L. J. Lee, H. Bretschneider, D. Merico, R. K. Yuen, Y. Hua, S. Gueroussov, H. S. Najafabadi, T. R. Hughes, *et al.*, "The human splicing code reveals new insights into the genetic determinants of disease," *Science*, vol. 347, no. 6218, p. 1254806, 2015.

[21] M. K. Leung, H. Y. Xiong, L. J. Lee, and B. J. Frey, "Deep learning of the tissue-regulated splicing code," *Bioinformatics*, vol. 30, no. 12, pp. i121–i129, 2014.

[22] A. G. Barto, R. S. Sutton, and C. J. Watkins, "Sequential decision problems and neural networks," in *Advances in neural information processing systems*, 1990, pp. 686–693.

[23] C. Guestrin, D. Koller, R. Parr, and S. Venkataraman, "Efficient solution algorithms for factored mdps," *Journal of Artificial Intelligence Research*, vol. 19, pp. 399–468, 2003.

[24] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, pp. 237–285, 1996.

[25] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[26] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[27] R. S. Sutton, S. D. Whitehead, *et al.*, "Online learning with random representations," in *Proceedings of the Tenth International Conference on Machine Learning*, 1993, pp. 314–321.

[28] G. A. Rummery and M. Niranjan, "On-line q-learning using connectionist systems," 1994.

[29] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," *Advances in neural information processing systems*, pp. 1038–1044, 1996.

[30] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*, 2000, pp. 1008–1014.

[31] A. Likas, "A reinforcement learning approach to online clustering," *Neural computation*, vol. 11, no. 8, pp. 1915–1932, 1999.

[32] W. T. Uther and M. M. Veloso, "Tree based discretization for continuous state space reinforcement learning," in *Aaai/iaai*, 1998, pp. 769–774.

[33] B. Ravindran, "Smdp homomorphisms: An algebraic approach to abstraction in semi markov decision processes," 2003.

[34] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[35] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[36] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[37] V. Mnih, N. Heess, A. Graves, *et al.*, "Recurrent models of visual attention," in *Advances in neural information processing systems*, 2014, pp. 2204–2212.

[38] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[39] Y. Freund, R. E. Schapire, *et al.*, "Experiments with a new boosting algorithm," in *ICML*, vol. 96, 1996, pp. 148–156.

[40] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[41] Y. Zhang, S. Burer, and W. N. Street, "Ensemble pruning via semi-definite programming," *The Journal of Machine Learning Research*, vol. 7, pp. 1315–1338, 2006.

[42] S. Bose and M. Huber, "Incremental learning of neural network classifiers using reinforcement learning," in *Systems, Man, and Cybernetics (SMC), 2016 IEEE International Conference on*. IEEE, 2016, pp. 002 097–002 103.

[43] J. D. Basilico, M. A. Munson, T. G. Kolda, K. R. Dixon, and W. P. Kegelmeyer, "Comet: A recipe for learning and using large ensembles on massive data," in *Data mining (ICDM), 2011 IEEE 11th international conference on*. IEEE, 2011, pp. 41–50.

[44] P. K. Chan and S. J. Stolfo, "On the accuracy of meta-learning for scalable data mining," *Journal of Intelligent Information Systems*, vol. 8, no. 1, pp. 5–28, 1997.

[45] P. M. Granitto, P. F. Verdes, and H. A. Ceccatto, "Neural network ensembles: evaluation of aggregation algorithms," *Artificial Intelligence*, vol. 163, no. 2, pp. 139–162, 2005.

[46] M. R. Azimi-Sadjadi, S. Sheedvash, and F. O. Trujillo, "Recursive dynamic node creation in multilayer neural networks," *Neural Networks, IEEE Transactions on*, vol. 4, no. 2, pp. 242–256, 1993.

[47] R. Rifkin and A. Klautau, "In defense of one-vs-all classification," *The Journal of Machine Learning Research*, vol. 5, pp. 101–141, 2004.

[48] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1, pp. 181–211, 1999.

[49] G. Martinez-Muoz, D. Hernández-Lobato, and A. Suarez, "An analysis of ensemble pruning techniques based on ordered aggregation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, no. 2, pp. 245–259, 2009.

[50] E. Bauer and R. Kohavi, "An empirical comparison of voting classification algorithms: Bagging, boosting, and variants," *Machine learning*, vol. 36, no. 1-2, pp. 105–139, 1999.

[51] D. Mena-Torres and J. S. Aguilar-Ruiz, "A similarity-based approach for data stream classification," *Expert Systems with Applications*, vol. 41, no. 9, pp. 4224–4234, 2014.

[52] N. C. Oza, "Online bagging and boosting," in *Systems, man and cybernetics, 2005 IEEE international conference on*, vol. 3. IEEE, 2005, pp. 2340–2345.

[53] S. Hashemi, Y. Yang, Z. Mirzamomen, and M. Kangavari, "Adapted one-versus-all decision trees for data stream classification," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 21, no. 5, pp. 624–637, 2009.

[54] R. Xu, D. Wunsch, *et al.*, "Survey of clustering algorithms," *Neural Networks, IEEE Transactions on*, vol. 16, no. 3, pp. 645–678, 2005.

[55] R. K. Ando and T. Zhang, "A framework for learning predictive structures from multiple tasks and unlabeled data," *The Journal of Machine Learning Research*, vol. 6, pp. 1817–1853, 2005.

[56] S. Bose and M. Huber, "Semi-unsupervised clustering using reinforcement learning." in *FLAIRS Conference*, 2016, pp. 150–153.

[57] H. Zeng and Y.-m. Cheung, "Semi-supervised maximum margin clustering with pairwise constraints," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 24, no. 5, pp. 926–939, 2012.

[58] N. Grira, M. Crucianu, and N. Boujemaa, "Unsupervised and semi-supervised clustering: a brief survey," *A review of machine learning techniques for processing multimedia content, Report of the MUSCLE European Network of Excellence (FP6)*, 2004.

[59] W. Zhao, Q. He, H. Ma, and Z. Shi, "Effective semi-supervised document clustering via active learning with instance-level constraints," *Knowledge and information systems*, vol. 30, no. 3, pp. 569–587, 2012.

[60] J. Zhang, M. Marszałek, S. Lazebnik, and C. Schmid, "Local features and kernels for classification of texture and object categories: A comprehensive study," *International journal of computer vision*, vol. 73, no. 2, pp. 213–238, 2007.

[61] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[62] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. 2579-2605, p. 85, 2008.

[63] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016.

[64] N. Z. Shor, *Minimization methods for non-differentiable functions*. Springer Science & Business Media, 2012, vol. 3.

[65] F. R. Bach, G. R. Lanckriet, and M. I. Jordan, "Multiple kernel learning, conic duality, and the smo algorithm," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 6.

[66] A. Ng, "Sparse autoencoder," *CS294A Lecture notes*, vol. 72, pp. 1–19, 2011.

[67] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[68] R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour, *et al.*, "Policy gradient methods for reinforcement learning with function approximation." in *NIPS*, vol. 99, 1999, pp. 1057–1063.

[69] G. Lever, "Deterministic policy gradient algorithms," 2014.

[70] S. Kakade, "A natural policy gradient." in *NIPS*, vol. 14, 2001, pp. 1531–1538.

[71] M. N. Katehakis and A. F. Veinott Jr, "The multi-armed bandit problem: decomposition and computation," *Mathematics of Operations Research*, vol. 12, no. 2, pp. 262–268, 1987.

[72] J. Langford and T. Zhang, "The epoch-greedy algorithm for multi-armed bandits with side information," in *Advances in neural information processing systems*, 2008, pp. 817–824.

[73] R. Hafner and M. Riedmiller, "Reinforcement learning in feedback control," *Machine learning*, vol. 84, no. 1-2, pp. 137–169, 2011.

[74] J. Mutch and D. G. Lowe, "Object class recognition and localization using sparse features with limited receptive fields," *International Journal of Computer Vision*, vol. 80, no. 1, pp. 45–57, 2008.

[75] W. E. Vinje and J. L. Gallant, "Sparse coding and decorrelation in primary visual cortex during natural vision," *Science*, vol. 287, no. 5456, pp. 1273–1276, 2000.

[76] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, *et al.*, "Model-based reinforcement learning for atari," *arXiv preprint arXiv:1903.00374*, 2019.

[77] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller, "Embed to control: A locally linear latent dynamics model for control from raw images," in *Advances in neural information processing systems*, 2015, pp. 2746–2754.

[78] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learning latent dynamics for planning from pixels," *arXiv preprint arXiv:1811.04551*, 2018.

[79] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh, "Action-conditional video prediction using deep networks in atari games," in *Advances in neural information processing systems*, 2015, pp. 2863–2871.

[80] J. Schmidhuber, "Formal theory of creativity, fun, and intrinsic motivation (1990–2010)," *IEEE Transactions on Autonomous Mental Development*, vol. 2, no. 3, pp. 230–247, 2010.

[81] N. Kalchbrenner, A. van den Oord, K. Simonyan, I. Danihelka, O. Vinyals, A. Graves, and K. Kavukcuoglu, "Video pixel networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1771–1779.

[82] B. Ravindran and A. G. Barto, "Approximate homomorphisms: A framework for non-exact minimization in markov decision processes," 2004.

[83] J. Oh, S. Singh, and H. Lee, "Value prediction network," in *Advances in Neural Information Processing Systems*, 2017, pp. 6118–6128.

[84] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, "Value iteration networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 2154–2162.

[85] J. Taylor, D. Precup, and P. Panagaden, "Bounding performance loss in approximate mdp homomorphisms," in *Advances in Neural Information Processing Systems*, 2009, pp. 1649–1656.

[86] T. Dean, R. Givan, and S. Leach, "Model reduction techniques for computing approximately optimal solutions for markov decision processes," in *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1997, pp. 124–131.

[87] S. Rajendran and M. Huber, "Learning to generalize and reuse skills using approximate partial policy homomorphisms," in *2009 IEEE International Conference on Systems, Man and Cybernetics*. IEEE, 2009, pp. 2239–2244.

[88] M. Asadi and M. Huber, "A dynamic hierarchical task transfer in multiple robot explorations," in *Proceedings on the International Conference on Artificial Intelligence (ICAI)*. The Steering Committee of The World Congress in Computer Science, Computer ..., 2015, p. 22.

[89] A. P. Wolfe and A. G. Barto, "Decision tree methods for finding reusable mdp homomorphisms," in *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, vol. 21, no. 1. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006, p. 530.

[90] V. Soni and S. Singh, "Using homomorphisms to transfer options across continuous reinforcement learning domains," in *AAAI*, vol. 6, 2006, pp. 494–499.

[91] G. Comanici, D. Precup, and P. Panangaden, "Basis refinement strategies for linear value function approximation in mdps," in *Advances in Neural Information Processing Systems*, 2015, pp. 2899–2907.

[92] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther, "Autoencoding beyond pixels using a learned similarity metric," *arXiv preprint arXiv:1512.09300*, 2015.

[93] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

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

[95] E. J. Gumbel, *Statistical theory of extreme values and some practical applications: a series of lectures*. US Government Printing Office, 1954, vol. 33.

[96] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[97] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.