

Reinforcement Learning on Resource Bounded Systems

by

Jaden B. Travnik

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

© Jaden B. Travnik, 2018

Abstract

Recent advancements in reinforcement learning have made the field interesting to academia and industry alike. Many of these advancements depend on deep learning as a means to approximate a value function or a policy. This dependency usually relies on high performance hardware (e.g., a graphics processing unit, GPU) and applications of deep learning are often limited to domains where a substantial amount of time is allowed for a prediction to be made or an action to be chosen. Although these criteria cover many important use cases, the growing popularity of the “Internet of Things,” wearable electronics, and the advancement of myoelectric prosthetic limbs presents a rapidly growing real-time domain of resource bounded systems that are not properly suited for deep learning yet could still benefit from the application of reinforcement learning. Furthermore, many of these systems are limited by the physical space they must occupy. This restricts the size of the hardware and thereby the computational and memory resources that can be used. Despite these restrictions, demand for prompt actions from receptive systems continues to grow.

To address these problems, I first highlight the difficulties one is faced with when implementing reinforcement learning on a system which is deployed in an asynchronous environment and introduce a new metric of performance by measuring the time it takes for a system to react to an observed state of an asynchronous environment. Secondly, I develop a class of algorithms that addresses these issues by reordering the algorithmic components to minimize

reaction time. Thirdly, to minimize both the time and memory necessary to compute function approximation, I introduce a novel linear function approximation method, selective Kanerva coding (SKC), that allows a reinforcement learning agent to perform behaviors reactively in real-time while using less memory and computation time than the standard linear approach of tile coding. I also show that SKC is less sensitive to the curse of dimensionality than tile coding making SKC a significant step towards accurately representing high dimensional data on resource bounded systems. Moreover, I show that SKC can make the inclusion of more sensory modalities more feasible, which can increase prediction accuracy when those modes of sensation are relevant to the prediction. Finally, I present an exploration of the meta-parameters of SKC and evaluate the performance of two different variations of SKC against the original formulation.

These findings are imperative to the current state of the field of reinforcement learning as they form a challenging perspective that is contrary to the current direction of the field's focus on deep learning. I form this argument by emphasizing the impracticality of deep learning in domains of resource bounded systems deployed in real-time environments, establishing the limitations on available computation and memory of these systems, and address these issues by proposing new insights, algorithms, and representations.

Preface

A version of Chapter 3 has been submitted for publication as Jaden B. Travnik, Kory W. Mathewson, Richard S. Sutton, and Patrick M. Pilarski, “Reactive Reinforcement Learning in Asynchronous Environments”, in *Frontiers in Robotics and AI*.

A version of Chapter 4 has been accepted for publication and presentation as Jaden B. Travnik and Patrick M. Pilarski, “Representing High-Dimensional Data to Intelligent Prostheses and Other Wearable Assistive Robots: A First Comparison of Tile Coding and Selective Kanerva Coding”, for the *Proc. of the 2017 IEEE International Conference on Rehabilitation Robotics (ICORR)*. London, United Kingdom, 2017 where it was presented as a poster. An extended abstract of this paper was also presented as a poster and a 60 second spot-light talk at the Multidisciplinary Conference on Reinforcement Learning and Decision Making (RLDM), Ann Arbor, Michigan, June 11-14, 2017.

A version of Chapter 5 has been submitted for publication as Jaden B. Travnik, Dylan J. A. Brenneis, Michael R. Dawson, and Patrick M. Pilarski, “Grasping Predictions with Multimodal Sensors”, in *Frontiers in Neurorobotics*.

The experimental design, implementation, execution, and analysis for the bulk of manuscript composition was authored by myself. The supervisory author, Pilarski, contributed to the manuscript and provided feedback and insight throughout the process of the research detailed within.

This research received ethics approval for its protocol from the human research ethics board at the University of Alberta.

To my parents, Danijel and Christel

Thank you for the love and support and free food, for teaching me good work ethics, and for letting my imagination run wild.

To my brother, Tyson and sister-in-law Gabrielle

Thank you for looking out for me, for being role models, and for sharing your sense of humor.

To Sarah

Thank you for sharing your life with me, listening to my wacky thoughts, and for your unwavering love and support.

Acknowledgements

I wish to express my sincerest gratitude to Patrick M. Pilarski for his fantastic energy, optimism, and mentorship. To Patrick: Not only have you imparted to me an appreciation and understanding of good research but you also provided me with a great environment to work and think in as well as the time and encouragement necessary to explore my ideas and become a better scientist. The many opportunities and trips that I have gone on have given me a lot of research experience but also invaluable world knowledge and I could not have done any of it without your support and guidance. I couldn't have asked for a better supervisor.

I'd also like to thank Rich Sutton. Like many former, current, and likely future graduate students, his course was a major decisive factor in my choice to work on Reinforcement Learning. His critical feedback has made me analyze my thoughts thoroughly and has helped me become a better researcher. Many thanks to Martha White and Martin Müller for their thoughtful questions and suggestions as a part of my thesis committee.

I'm incredibly grateful to Dylan Brenneis, Kory Mathewson, Michael Rory Dawson, Alex Kearney, Gautham Vasani, Craig Sherstan, Adam Parker, and Vivek Veeriah for all their help throughout. We've had multiple discussions about AI, neuroscience, the function of happiness, water bears, flaws in time travel movies, video games, and we also started the Society of Those Who Have Generous Hoods And Sleeves. I am also hugely grateful to other colleagues in the BLINC lab (Nadia¹, Katherine, Tarvo, McNiel, Jon, Sai, Riley, Quinn, Oggy, James, and Jackie) and the RLAI lab for their sincere friendship, interest, and support.

¹Who is one of the best editors I have ever known.

Contents

1	Introduction	1
1.1	Addressing Computation Challenges of Reinforcement Learning on Resource Bounded Systems	2
1.2	Our Contributions	4
2	Background Material	5
2.1	Limited Computation on Untethered Systems	5
2.1.1	Internet of Things	5
2.1.2	Wearable Electronics	6
2.1.3	Myoelectric Prosthetic Limbs	6
2.2	Custom Hardware	8
2.2.1	The Bento Arm	8
2.2.2	Humanoid, Anthropometric, Naturally Dexterous, Intelligent (HANDi) Hand	9
2.2.3	Gesture Layout Observation Via Impedance (GLOVi) Glove	10
2.3	Reinforcement Learning	11
2.3.1	True Online Temporal-Difference Learning	13
2.3.2	General Value Functions	13
2.3.3	Continuous-Time and Real-Time Reinforcement Learning	14
2.3.4	Data Traces	16
2.4	Sparse Distributed Memory	17
3	Reaction Time is Important	18
3.1	Overview	18
3.1.1	Related Background	19
3.2	Temporal Delays in Asynchronous Environments	20
3.3	Reactive SARSA	21
3.4	Experiments	23
3.4.1	Experiment 1	25
3.4.2	Experiment 2	27
3.5	Discussion	28
3.6	Conclusions	30
4	Dealing with Increasing Dimensionality	32
4.1	Overview	32
4.1.1	Towards Computationally Efficient Representations	34
4.2	Representation using Linear Methods	35
4.2.1	Tile Coding	37
4.2.2	Kanerva Coding	38
4.2.3	Selective Kanerva Coding	39
4.3	Experiment	40

4.4	Results	42
4.5	Discussion	44
4.6	Conclusions	46
5	Selective Kanerva Coding in a Multimodal Domain	48
5.1	Overview	48
5.2	Experiments	50
5.2.1	Data Collection	50
5.3	Results	55
5.4	Discussion	58
5.5	Conclusions	60
6	Discussion and Exploration of Selective Kanerva Coding	61
6.1	Sensitivity Analysis of η	61
6.2	Multiple Feature Sets	63
6.2.1	Multiple Prototype Sets	64
6.2.2	Multiple η Values	67
6.3	Future Research Areas	69
6.3.1	Optimal Algorithm Ordering	69
6.3.2	Future Work with Selective Kanerva Coding	69
7	Conclusion	71
	References	74

List of Tables

4.1	The maximum number of features that can be calculated within 100 ms on a single-core 2.5 GHz processor.	42
-----	---	----

List of Figures

2.1	A labeled high-level depiction of a prosthetic arm.	7
2.2	A side view of the Bento Arm with a chopsticks gripper. . . .	8
2.3	A picture of the HANDi Hand	9
2.4	A labeled picture of an experimenter wearing the GLOVi Glove.	10
2.5	The interaction a reinforcement learning agent has with its environment adapted from Sutton and Barto, 1998.	12
2.6	An example of data traces.	17
3.1	The Hallway-World Task	21
3.2	Timestep comparison of standard and reactive reinforcement learning algorithms	23
3.3	The setup of the robot for Experiment 1 in Chapter 3.	24
3.4	Key result: a comparison of summed reward over the last 10 episodes of 30 trials across 5 different learning delay length lengths during robot arm motion.	26
3.5	The resulting of failed stops in Experiment 2 in Chapter 3. . .	28
3.6	Boxplot comparison of the distributions of events over all episodes between Reactive SARSA and the standard SARSA algorithm.	29
4.1	The setup of the Bento Arm in Chapter 4.	33
4.2	A depiction of Sparse Distributed Memory (SDM) prototypes being activated within a fixed radius of a specified address in the state space.	36
4.3	A depiction of tile coding	36
4.4	Comparison of the prediction error and the computation time fore tile coding and selective Kanerva coding.	43
5.1	The table associating GLOVi Glove hand gestures to shapes for Chapter 5.	51
5.2	A side view of the experiment setup showing how the shapes were oriented in the HANDi Hand’s grasping area.	52
5.3	A top view of the experiment setup showing how the shapes were rotated into the HANDi Hand’s grasping area.	52
5.4	A picture showing the HANDi Hand (left) holding a green cylinder and the experimenter wearing the GLOVi Glove (right) performing the appropriate hand gesture as indicated by the table in Figure 5.1.	52
5.5	The camera’s 480 x 640 RGB view of the green cylinder (top) against the black background with the Thumb of the HANDi Hand (left) in view.	52
5.6	A sample of the data stream from the Experiment in Chapter 5.	53
5.7	A plot of the predictions for one of the Green Cylinder trials in the testing phase.	56

5.8	A side-by-side boxplot of the distributions of the error between the true return and the predictions during the testing phase in Chapter 5.	57
5.9	The results of a Cohen's D test between the error of the different input settings for each prediction in Chapter 5.	58
6.1	The sensitivity in error for different values of η of varying sizes of prototype sets.	62
6.2	The distribution of the distance between a random point in $[0, 1]^n$ space, for $n \in \{1, 2, 4, 8\}$, to the nearest prototype as the size of the prototype set increases. 50 random seeds were used to initialize each prototype set for each dimension setting. 200 random points were then generated and the distance between them and the closest prototype was recorded.	63
6.3	A high-level example of 3 overlapping prototype subsets. . . .	64
6.4	Time comparison between single set and multiple subsets	65
6.5	The error sensitivity profiles of different prototype subsets across different values of η	66
6.6	A high-level example of 3 η s on the same prototype set.	67
6.7	A comparison of the error for the best η for normal 500, 4000, and 8000 prototype sets against the error for 4000 prototypes with two η s and 500 prototypes with 16 η s ranging between 0.0025 to 0.075.	68

If knowledge can create problems, it is not through ignorance that we can solve them.

– Issac Asimov, science fiction novelist & scholar (1920 - 1992)

This quote is often falsely attributed to Mark Twain.

– Randal Munroe, comic artist of xkcd (1984 - Present)

Chapter 1

Introduction

On-device learning for untethered systems is rapidly becoming a necessity for many applications of machine learning. Along with the complexities of machine learning, deployments of untethered systems bring along their own abundance of problems (Kober, Bagnell, and Peters 2013). The amount of on-board computation, memory, battery power, and the physical size and shape of the system are often restrictions imposed by the physical design and installation of the hardware. These challenges are not necessarily overcome by an increased budget for high-end components. For example, systems used in medical rehabilitation such as prosthetic devices are restricted to wearable sizes and must last long periods of time on a single battery. Even further restrictions are present in implanted devices like pacemakers or implantable cardioverter defibrillators, which must be small and last years on a single battery to minimize surgical procedures to replace the system. These physical restrictions, in turn, limit what on-board hardware can be installed. Despite the limitations, demand for intelligent devices grows as the benefit of intelligent devices becomes more apparent (Castellini et al. 2014; Kortuem et al. 2010; Scheme and Englehart 2011a; Google TensorFlow Lite¹; Apple Core ML²; SONY NNABLA³).

In addition to the demand of intelligent systems, low latency in response time is also a common request. Humans today often get frustrated when their

¹Google TensorFlow Lite - tensorflow.org/mobile/tflite/

²Apple Core ML - developer.apple.com/machine-learning/

³SONY NNABLA - nnabla.org/

mobile phones take seconds to perform a computation that once would have taken years (Ceaparu et al. 2004; Ramsay, Barbesi, and Preece 1998). A system that can react swiftly to new data is important to quell impatience but also is necessary to take advantage of the information and respond accordingly, such as selling when current stock prices are high or braking to avoid a newly detected pedestrian.

One method of decreasing the reaction time as well as the battery consumption on a resource-bounded system is to offload the computation to a more powerful remote system which can return results back to the resource-bounded system over a network. For many applications, this method is powerful in that it conserves time and battery life and is most beneficial when large amounts of computation are needed with relatively small amounts of communication (Chen et al. 2004). However, when the amount of communication becomes excessive, the toll on latency and battery life outweighs any benefit (Kumar, Liu, et al. 2013; Kumar and Lu 2010). For systems with multiple high-rate sensors, such as upper-limb prosthetic devices or self-driving cars where reactivity and reliability are priorities, offloading computation in this manner is not a viable option (Dawson et al. 2014; Johannes et al. 2011).

1.1 Addressing Computation Challenges of Reinforcement Learning on Resource Bounded Systems

This thesis explores how reinforcement learning can be applied on resource bounded systems. Specifically, we study how reinforcement learning with linear function approximation can be deployed on resource bounded systems and what limitations one should be aware of when doing so.

Linear function approximation is appealing on resource bounded systems because, along with theoretical convergence guarantees, linear function approximation can be applied on-line. This is crucial when considering optimal control policies and value functions in which estimates must be available after each interaction with the world (Sutton, Szepesvári, et al. 2012). Otherwise,

if learning updates were instead performed in mini-batches, the extra computation on the learning step would postpone actions and increase the reaction time of the system.

Although there has been no comprehensive study of reinforcement learning with linear approximation on resource bounded systems, other studies have addressed the issues that come with applying reinforcement learning to robotic domains, which also arise in resource bounded systems. Like other reinforcement learning applications, robotic systems often suffer from the curse of dimensionality, as they require increasing amounts of data and computation to cover the state-action space. This curse is confounded in robotic systems by the fact that collecting data on a robot is time consuming and expensive. Moreover, robotic systems themselves are often expensive and because they must interact with the physical world, safe exploration becomes key to avoid damage caused by the system (Hester, Quinlan, and Stone 2010; Kober, Bagnell, and Peters 2013).

Others have focused on maximizing the representative power of linear representations by gradually modifying a linear function approximation to have higher accuracy (Cheng and Meleis 2008; Curran et al. 2016; Li et al. 2017; Lin and Wright 2010; Ratitch, Mahadevan, and Precup 2004; Ratitch and Precup 2004; Whiteson, Taylor, and Stone 2007; Wu and Meleis 2009). However, these works often do not consider state spaces with more than four dimensions and never assess the time it takes to compute these representations on resource bounded systems as the tasks are performed in simulation.

More recent work shows promising results by approximating Gaussian Radial Basis Functions with Finite Support Basis Functions. A Gaussian Radial Basis Function (GRBF) is a function that computes the distance from an input point, x , to some fixed point called the 'center', c , using the formula $\phi(r) = e^{-(\epsilon r)^2}$, where $r = |||x - c|||$ and $\epsilon \in \mathbb{R}$. Typically, multiple GRBFs are combined in a weighted sum. Learning methods can be applied to these weights so that the result of the weighted sum of GRBFs approximates some target function. As function approximation using GRBFs requires the computation of multiple exponential functions, recent work has proposed using

approximations of exponential functions, called Finite Support Basis Functions, can reduce the computation time on resource bounded systems while not sacrificing performance (Lobos-Tsunekawa, Leottau, and Ruiz-del-Solar 2017).

1.2 Our Contributions

This thesis seeks to answer the following question: Can we develop a representation that allows a reinforcement learning agent to perform behaviors reactively in real-time with limited computational resources and training times? One of the main objectives of this thesis is to highlight the difficulties one is faced with when implementing reinforcement learning on a real-time system.

In chapter 3, we **define an asynchronous environment** to better model the control problem an agent faces when deployed in real-time domains. We then **present a class of reactive algorithms** to deal with these asynchronous environments by **minimizing the reaction time** of the algorithm while ensuring the same theoretical guarantees of previous control algorithms.

In Chapter 4, we **introduce a novel linear representation**, selective Kanerva coding, which **requires fewer features**, and less time than the popular linear representation method, tile coding, while having **better prediction performance**.

In Chapter 5, we show that although adding multiple modalities increases the dimensionality of the state space, it can be worth it for the increase in accuracy. We also show that by **using selective Kanerva coding** on a multimodal domain, we **can make adding multiple modality sensors more tractable** on resource bounded systems.

Chapter 6 **addresses possible issues** as well as **future areas of study** and includes an **exploration of the meta parameters and variations of selective Kanerva coding** which can help to decrease the necessary memory.

Lastly, Chapter 7 reaffirms the arguments made in the previous chapters, makes final remarks and offers a conclusive summary of this work.

Chapter 2

Background Material

2.1 Limited Computation on Untethered Systems

2.1.1 Internet of Things

The Internet of Things (IoT) connects electronic devices in much the same way that the Internet connects people around the globe. The main goal of connecting electronic devices in this way is so that they may communicate their own information to other devices so that the devices can work more synergistically (Lee, Crespi, et al. 2013; Xia et al. 2012). For example, a motion detector set up in a home as part of a security system could be connected to the lights in the house and turn them off when the occupants are not in the room thus saving energy. Another example could be a person's mobile phone sending a message to the house's furnace to heat up the home before the person arrives home from work. Use cases like these are common in modern life and become possible as more devices are connected to the IoT (Kortuem et al. 2010; Wortmann and Flüchter 2015; Yang 2014). It is often the case that appliances, such as the furnace in the previous example, are connected to the IoT through small cheap micro-controllers such as an Arduino or micro-computers similar to the Raspberry Pi or the Beagle Bone (Badamasi 2014; Coley 2013; Kopetz 2011; Richardson and Wallace 2012). Despite their lack of computation ability, these small devices are used to read sensors, receive messages, and carry out simple tasks thus providing the underlying mechanisms for the IoT.

2.1.2 Wearable Electronics

An interesting side product of the Internet of Things is wearable electronics, the idea that electronics can be worn to read and display information about the person wearing them or the environment around the person. An obvious example of wearable electronics are smart phones but growing interest in recording one's physical activity has motivated the development of wearable devices that measure data such as daily steps, heart rate, quality of sleep, steps climbed, and other personal metrics involved in fitness (Tao 2005). Other examples are becoming more common as well, including self-heated jackets, light up shoes, smart glasses, and smart ID tags. Unlike most other applications of the IoT, wearable electronics have to be in close proximity to a person moving around in the world. This means that they must be safe, durable, flexible, and portable. The challenge posed by wearable electronics led to innovations in the different materials and fibers to be used to make flexible wiring (Jost et al. 2013; Kou et al. 2014; Zeng et al. 2014), heaters (Hong et al. 2015), sensors (Lee, Kwon, et al. 2015), and even methods to self-power electronics using human body heat (Leonov and Vullers 2009).

2.1.3 Myoelectric Prosthetic Limbs

A myoelectric prosthetic arm is a battery-powered robotic arm attached to a person. Typically the actuating joints are controlled via electromyographic (EMG) signals, the electrical potential of active muscles. These signals are recorded by placing non-invasive electrodes on the user's skin over target muscle sites (see Figure 2.1). The recorded signals from each EMG channel are then filtered, amplified, and processed so that they can be used as control signals (Scheme and Englehart 2011b).

A common complaint with mobile electronics is limited battery power and myoelectric prosthetic arms are no exception (Pilarski and Hebert 2017). Although some mobile electronics have space for a larger battery, myoelectric prosthetic arms do not have this affordance so readily. Typically very little room for extra battery power is available within the housing of the arm and

often the available space is a complex shape. These restrictions challenge electrical engineers to design physically malleable batteries with unique geometries such that the volume within the arm can be optimally utilized.

Although myoelectric arms have become powerful devices, the gross mismatch between the functions available and the control signals offered by the person creates a very challenging problem. While applications of pattern recognition have had varying degrees of success to deal with this problem, the necessity for more intelligent prosthetic arms becomes more evident as the functionality of prosthetic arms continues to grow (Cook and Polgar 2015; Dawson et al. 2014; Johannes et al. 2011; Pilarski, Sutton, and Mathewson 2015; Scheme and Englehart 2011b). Although there are promising results of artificially intelligent arms working together with the person to overcome this challenge, the results have yet to be evaluated on the limited computation available on battery-powered prosthetic arms (Edwards 2016; Sherstan, Modayil, and Pilarski 2015).

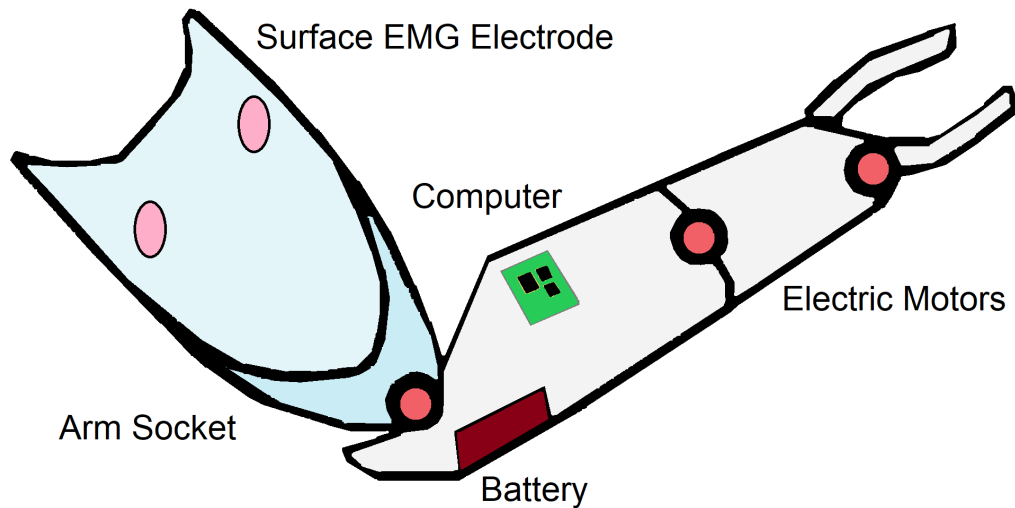


Figure 2.1: A labeled high-level depiction of a prosthetic arm.

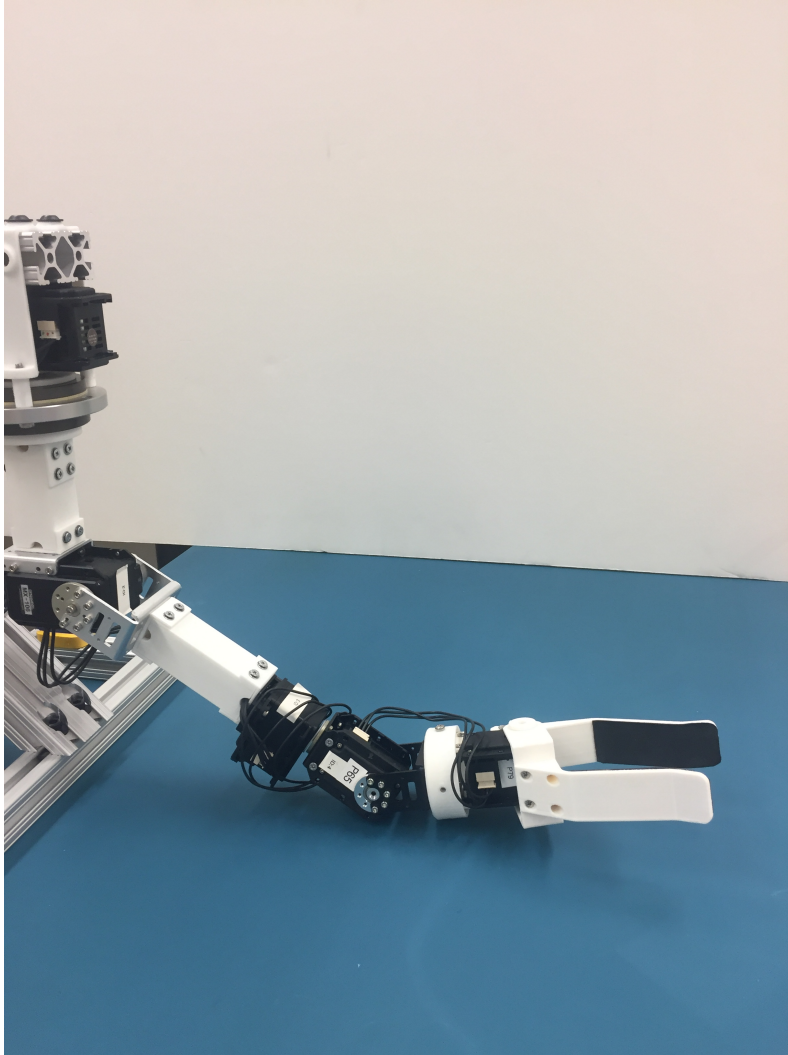


Figure 2.2: A side view of the Bento Arm with a chopsticks gripper.

2.2 Custom Hardware

2.2.1 The Bento Arm

The studies in Chapters 3 and 4 used an inexpensive robotic arm known as the Bento Arm (Dawson et al. 2014). This arm was designed to be worn as a myoelectric training device and research platform for machine learning techniques. The arm is comprised of five actuators from the Dynamixel line (Robotis, Inc.), one for each of the following functions: shoulder rotation, elbow flexion, wrist rotation and extension, and for the gripper (see Figure 2.2). For the experiment in Chapter 3, the gripper on the arm was replaced with

a custom hammer attachment to more effectively break eggs. Each actuator provided sensor readings of encoder position, angular velocity, temperature, and load which were read at a rate of 30Hz using a USB2Dynamixel converter provided by Robotis, Inc.

2.2.2 Humanoid, Anthropometric, Naturally Dexterous, Intelligent (HANDi) Hand

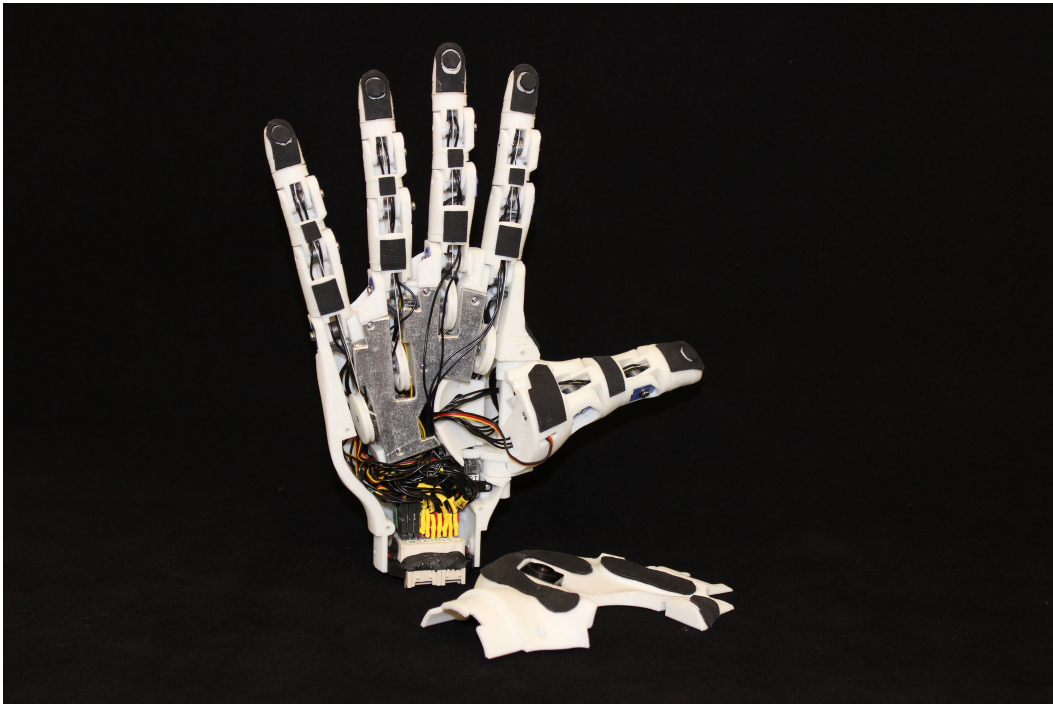


Figure 2.3: A picture of the HANDi Hand opened with the palm-plate open revealing the closing mechanisms for the fingers and the webcam embedded in the palm.

The experiments in Chapter 5 and 6 made use of an inexpensive sensorized hand known as the Humanoid, Anthropometric, Naturally Dexterous, Intelligent (HANDi) Hand, developed at the University of Alberta (Brenneis, Dawson, and Pilarski 2017)¹, which has one force sensitive resistor (FSR) embedded into each fingertip (see Figure 2.3). The HANDi Hand is also capable of position sensing by use of MuRata SV series rotary potentiometers (Murata Manufacturing Co. Ltd., Kyoto, JP) at 9 of the joints. These measured

¹HANDi Hand Repository - github.com/blincdev/HANDi-Hand

joints include all of the joints of the first digit (the thumb), the proximal and intermediate joints of digits 2 and 3, and the proximal joints of digits 4 and 5. A USB webcam embedded in the palm of the hand records visual data from the perspective of the hand. The hand is controlled using an Arduino Mega (Arduino LLC, Italy), which sends position signals to the servos of the hand and collects sensory signals from the sensors in the hand. The FSRs and potentiometers send analog signals ranging from 0 to 5 V to the Arduino; the Arduino’s analog-to-digital converter (ADC) converts this range to an integer in the range 0 - 1023. The USB webcam bypasses the Arduino and records directly to a computer, with an image size of 480 x 640 RGB at a rate of 15 Hz.

2.2.3 Gesture Layout Observation Via Impedance (GLOVi) Glove

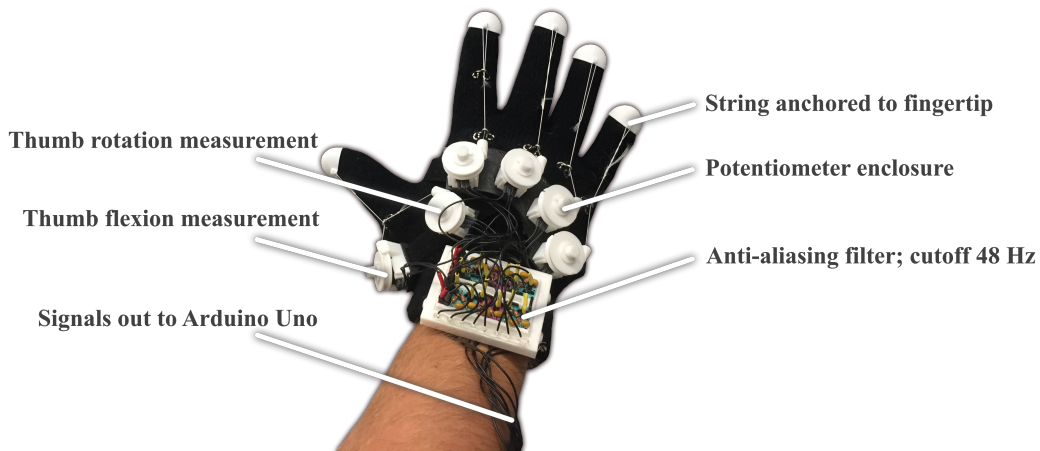


Figure 2.4: A labeled picture of an experimenter wearing the GLOVi Glove.

Also used in the experiment in Chapter 5 and 6 was a Gesture Layout Observation Via Impedance (GLOVi) Glove, an inexpensive data capture glove, again developed at the University of Alberta. This glove uses the same type of rotary potentiometers as the HANDi Hand to measure the flexion of each finger of the wearer of the glove, as shown in Figure 2.4. This is accomplished by means of a string tied to the tip of each finger, which rotates a spring-

loaded spool attached to the back of the glove, which in turn rotates the potentiometer. The potentiometer signals, after individually passing through an anti-aliasing filter with a cutoff frequency of 48 Hz, are fed directly to an Arduino Uno. These signals also range from 0 to 5 V, and are converted to an integer in the range 0 - 1023 using the Arduino's ADC. The potentiometers on the back of the glove are set up in such a manner so as to independently measure thumb flexion and rotation. In this way, six signals describe the level of flexion/extension of each of the fingers of the hand; finger ad/abduction is not measured. A more detailed description of the design and capabilities of the GLOVi Glove accompanies the open-source release of the design².

2.3 Reinforcement Learning

Reinforcement learning (RL) is a learning method that uses interaction with an environment to learn rewarding control policies. Often referred to as an agent, a system utilizing RL learns to achieve a goal through interaction with its environment. In the RL learning problem explained in Sutton and Barto, 1998 the agent interacts with its environment at each **timestep** t by taking an **action** A_t and the environment in turn yields a new situation referred to as the **state** S_{t+1} as well as a special scalar value called the **reward** r_{t+1} . The agent's goal is to maximize the sum of the future discounted rewards it receives known as the discounted **return** G_t by a discount factor $\gamma \in [0, 1)$. In many domains, the agent's actions have long term consequences thus, although the environment yields a reward on every timestep, a change in the reward as a consequence of an agent's action could be temporally delayed. This delay makes it difficult for an agent to discern what past action or series of actions caused this change. This temporal dependency is what makes RL different from other learning methods in that time matters.

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots = \sum_{i=0}^{\infty} \gamma^i * r_{t+i+1}$$

²GLOVi Glove Repository - github.com/blincdev/GLOVi-Glove

Although the agent cannot know the return ahead of time, it can be estimated by a **value function**. The **value** of a state estimates the return an agent can expect over the future, beginning in that state. These values can be learned through interaction with the environment and then used to take actions which lead to higher valued states. Further, one can learn the value of taking an action in a state, referred to as a *Q-value*. Q-learning is a popular algorithm that can learn these *Q-values* to give a good estimation of the value of an action in a state (Watkins 1989). *Q-values* are used to determine how an agent chooses its actions, referred to as a **policy**. For instance, taking the actions with the maximum *Q-value* is referred to as a greedy policy as it is exploiting the agent's previous experience to try and maximize the agent's expected future return. This exploitation means that some states and actions will not be seen by the agent if their estimated *Q-values* are low even if they would actually lead to higher returns. Thus, it is important to balance exploration and exploitation in an agent's policy.

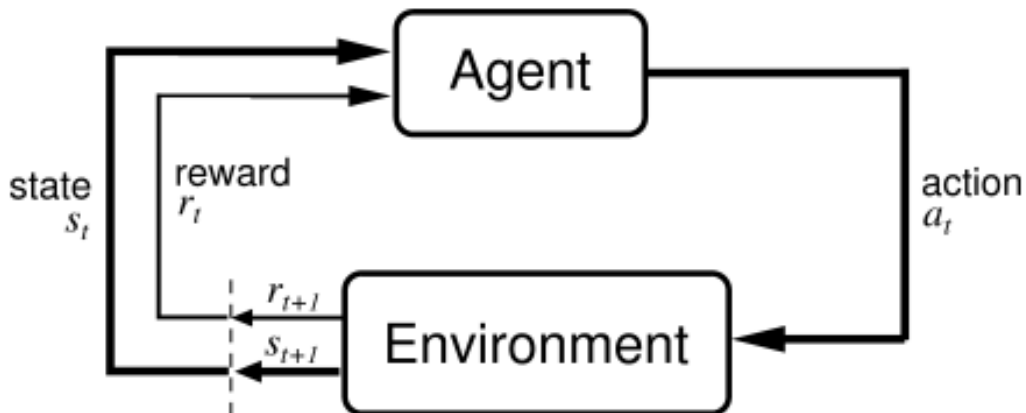


Figure 2.5: The interaction a reinforcement learning agent has with its environment adapted from Sutton and Barto, 1998.

2.3.1 True Online Temporal-Difference Learning

A central part of RL is the temporal-difference learning method known as TD(λ)³. With its low computational cost and good performance, it is frequently used when learning temporally extended predictions in the RL setting. However, TD(λ) is known to not maintain an exact equivalence with ideal mathematical outcomes for learned predictions—termed the *forward view*. Recently, Van Seijen et al. 2016 proposed two small changes to the update rule of TD(λ), allowing true online temporal-difference learning methods to be constructed that do have algorithmic equivalence with the forward view. These true online methods have been used to approximate value functions of sensorimotor interactions with superior performance over regular TD(λ) and SARSA(λ) (Van Seijen et al. 2016). Algorithm 1 shows the true online algorithm as implemented in Chapter 3.

Algorithm 1 True Online TD(λ)

```
Initialize  $\theta$  arbitrarily
loop {over episodes}
  Initialize  $e = \mathbf{0}$ 
  Initialize  $S$ 
   $\hat{v}_S \leftarrow \theta^\top \phi(S')$ 
  Repeat (for each step of episode):
    generate reward  $R$  and next state  $S'$  for  $S$ 
     $\hat{v}_{S'} \leftarrow \theta^\top \phi(S')$ 
     $\delta \leftarrow R + \gamma \hat{v}_{S'} - \hat{v}_S$ 
     $e \leftarrow \gamma \lambda e + \alpha [1 - \gamma \lambda e^\top \phi(S)] \phi(S)$ 
     $\theta \leftarrow \theta + \delta e + \alpha [\hat{v}_S - \theta^\top \phi(S)] \phi(S)$ 
     $\hat{v}_S \leftarrow \hat{v}_{S'}$ 
     $S \leftarrow S'$ 
  until  $S$  is terminal
end loop
```

2.3.2 General Value Functions

A value function in reinforcement learning is a mapping from the state at time t , S_t , of an agent to a prediction of an input signal given a target policy π and a

³ λ refers to the eligibility trace, a way of assigning credit for good and bad outcomes to states seen in the past (Sutton and Barto 1998).

termination function, $\gamma(S_t)$ (Sutton and Barto 1998; White 2015). Although value functions are commonly used in control problems to find the optimal policy by predicting the future return of a state given the current policy, they have also been used to construct more general knowledge from sensorimotor interaction such as bump sensors on mobile robots (Sutton, Modayil, et al. 2011) or servo encoder sensors on robotic arms (Edwards 2016; Travník and Pilarski 2017). These *general value functions* (GVFs) provide the semantics to ask questions about data experienced by a system (Sutton, Modayil, et al. 2011; White 2015). When a GVF is asking a question about the current behavior of an agent, it is referred to as an on-policy GVF. If the termination function γ is set to a constant value between 0 and 1, it represents making a prediction about the expected discounted return over a certain timescale. One can approximate this timescale by estimating the number of timesteps within the timescale as given by the equation $timesteps = \frac{1}{1-\gamma}$. Thus with a constant $\gamma = 0.9$, an on-policy GVF will make a prediction about the expected discounted return over approximately the next 10 timesteps. Similarly, with $\gamma = 0.999$ the prediction concerns the expected return over approximately the next 1000 timesteps.

2.3.3 Continuous-Time and Real-Time Reinforcement Learning

As described in the previous subsections, RL is classically defined as an agent in state S_t , taking an action, A_t , observing reward and state, r_{t+1} and S_{t+1} , respectively, and performing a learning update. The $t + 1$ subscript, that denotes a timestep jump, refers to the iteration of the learning loop. In stationary problems, such as checkers, the duration of the timestep is not a factor in learning nor part of the problem and is readily ignored. In other problems, like the cart-pole task, the timestep must be carefully chosen to have the desired outcome (see Sutton & Barto, 1998, p. 59 for a brief description). Large timesteps may not give the agent a proper view of the world unless special features are chosen to represent changes in the world’s state over time. Choosing timesteps to be smaller may also impact learning. For example, in domains

with continuous states and actions, it is often the case that performing one non-optimal action in a long sequence of optimal actions will have little effect on the total reward. In such cases, if using Q-learning, the Q -values between actions in the same state will be relatively close. If the Q-function is stored with a function approximation method with some error, the Q -values will tend to be sensitive to that error. This sensitivity may cause a non-optimal action to have a learned Q -value that is higher than the optimal action leading to a non-optimal policy. As the length of a timestep approaches 0, the penalty for taking one wrong action in a sequence decreases and the approximated Q-function becomes more sensitive to noise or function approximation error yielding a highly erroneous policy (Baird and Klopff 1993).

Beyond the issues associated with problem domains where a constant timestep length can be chosen, there are problem domains that have timesteps with random lengths. Rather than the typical time series that classic RL problems have, problem domains like traffic control, customer behaviour, and traces of events in computer systems have event-driven time series. In these domains, it is difficult to settle on a static timestep in which to apply RL. Unlike in classic RL problems, where the agent observes one state within one timestep, in event-driven time series problems there is some probability that more than one observation of state occurs within one timestep. Although there are several solutions to this problem, including combining observations, ignoring some, or assigning observations to “empty” timesteps, none of them are convincingly justified. It is obvious that a reduction of timestep length reduces the probability of co-occurrence of observations. However, small timesteps cannot represent long delays appropriately because if the length of time between observations varies greatly, it leads to long chains of silence which tend to deteriorate model quality.

It is important to note that even though learning algorithms on digital computers cannot be in continuous time, the policies they produce are in continuous time if they are within a continuous time system such as a robot. For example, if an agent deployed on a robot initiates the action of *forward*, the robot will begin moving forward and will continue to do so until the agent’s

next timestep where the agent can initiate another action. If the robot instead moved a preset distance on each *forward* action and stopped before the agent’s next action took place, the pause in movement is inherent to *forward* action and thus a result of the agent’s policy. In either case, an agent is constantly interacting with the world through its most recently calculated policy. It is therefore importunate that RL systems have short timesteps when deployed in fast dynamic systems, and even more so when reaction time is prioritized. The longer the duration of a timestep, the rarer the opportunity an agent has to change its behaviour.

2.3.4 Data Traces

A common problem in reinforcement learning is state conflation, when an agent cannot determine the difference between two different states. This tends to happen if the agent has no information about how long a signal has had its current value. For example, given a repeating binary signal which is 1 for 500 steps and 0 for 1500 steps (a total period of 2000 steps) the agent would not be able to predict the signal changes unless it had information about how long the signal had been in its current state. A common solution to this type of problem is to include a summarization of the history of the signal, known as a trace, in the agent’s representation (Edwards 2016; Edwards, Hebert, and Pilarski 2016; Edwards, Kearney, et al. 2013; Pilarski, Dawson, Degris, Carey, and Sutton 2012; Pilarski, Dawson, Degris, Fahimi, et al. 2011). Given a signal S_i and a decay parameter $\lambda_{decay} \in (0, 1)$, the value of the trace T at time t is $T_{t+1} = T_t * \lambda_{decay} + S_{i_{t+1}}$. The closer the value of λ_{decay} is to 1, the longer the history will be, with the length of the history in timesteps given by the formula $timesteps = \frac{1}{1-\lambda_{decay}}$ as seen in Figure 2.6. It is common to normalize the value of the trace before using it in a representation ($NormalizedT_{t+1} = T_{t+1}(1 - \lambda_{decay})$).

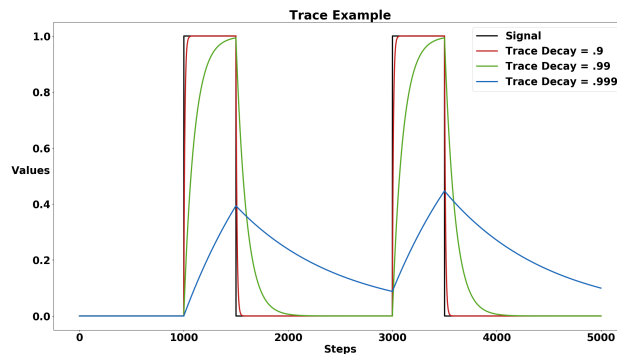


Figure 2.6: An example of traces for a signal (black) using 3 different values of a decay parameter, λ_{decay} .

2.4 Sparse Distributed Memory

Sparse Distributed Memory (SDM) is a mathematical model of human long-term memory (Kanerva 1988). It models how the distances between concepts in a human mind are similar to the distances between points of a high-dimensional space where high-dimensional is at least in the hundreds. In the original formulation presented by Pentti Kanerva, a high-dimensional binary space could be represented with a set of fixed randomly placed points in the same space known as *prototypes* (Kanerva 1988). Each prototype can be thought of as a memory element where its address is its position in the space, and its value is data that can be read from and written to. The main parameter of the representation is a radius. A prototype is said to be activated by a given address if the hamming distance between the prototype's position and the specified address is smaller than the radius.

In order to write to SDM, given an address and a value, each prototype that is activated by the specified address is updated to an average of its existing value and the specified value (see Figure 4.2). Reading from SDM is performed using the same radius method where the read value is then the average of the values from all activated prototypes. This read value is then a close approximation of data written previously to addresses near the specified address.

Chapter 3

Reaction Time is Important¹

3.1 Overview

Reinforcement learning (RL) algorithms for solving optimal control problems are comprised of four distinct components: acting, observing, choosing an action, and learning. This ordering of components forms a protocol which is used in a variety of applications. Many of these applications can be described as synchronous environments where the state of the environment remains in the same state until the agent acts at which point the environment immediately returns its new state. In these synchronous environments, such as Backgammon (Tesauro 1995) or classic control problems, it is not necessary to know the computation time to perform any of the protocol’s components. For this reason, most reinforcement learning software libraries, such as RL-Glue, BURLAP² or OpenAI gym³, have functions which accept the agent’s action, and return the new state and reward immediately (Tanner and White 2009). These functions remain convenient for simulated environments where the dynamics of the environment can be computed easily (Sutton and Barto 1998). However, unlike synchronous environments, asynchronous environments do not wait for an agent to select an action before they change state. The computation of RL protocol components (acting, observing, choosing an action, learning) takes

¹A version of Chapter 3 has been submitted for publication as Jaden B. Travnik, Kory W. Mathewson, Richard S. Sutton, and Patrick M. Pilarski, “Reactive Reinforcement Learning in Asynchronous Environments”, for *Frontiers in Robotics and AI*.

²<http://burlap.cs.brown.edu/>

³<https://gym.openai.com/>

time and an asynchronous environment will continually change state during this time (Caarls and Schuitema 2016; Degris and Modayil 2012; Hester, Quinlan, and Stone 2010). This can negatively affect the performance of the agent. If the agent’s reaction time is too long, its chosen action may become inappropriate in the now changed environment. Alternatively, the environment may have moved into an undesirable terminal state.

In this Chapter, we explore a very simple alternative arrangement of the reinforcement learning protocol components. We first investigate a way to reorder SARSA control algorithms so that they are able to react to the most recent observation before learning about the previous timestep; we then discuss convergence guarantees of these reordered approaches when viewed in discrete time (following (Singh et al. 2000)). Then, we examine an asynchronous continuous-time robot task where the reaction times of agents affect the overall task performance—in this case, breaking or not breaking an egg with a fast-moving robotic arm. Finally, we present a discussion on the implementation of reactive algorithms and their application in related settings.

3.1.1 Related Background

The focus of most contemporary RL research is on action selection, representation of state, and the learning update itself; the performance impact of reaction time is considered less frequently, but is no less important of a concern (Barto, Bradtke, and Singh 1995). Several groups have discussed the importance of minimizing reaction time (Caarls and Schuitema 2016; Degris and Modayil 2012; Hester, Quinlan, and Stone 2010). Hester et al. noted that existing model-based reinforcement learning methods may take too much time between successive actions and presented a parallel architecture that outperformed traditional methods (Hester, Quinlan, and Stone 2010). Caarls and Schuitema extended this parallel architecture to the online learning of a system’s dynamics (Caarls and Schuitema 2016). Their learned model allowed for the generation of simulated experience which could be combined with real experience in batch updates. While parallelization methods may improve performance, they are computationally demanding. We propose an alternative

approach when system resources are constrained.

3.2 Temporal Delays in Asynchronous Environments

Temporal-difference (TD) control algorithms like SARSA and Q-Learning were introduced with synchronous discrete-time environments in mind; these environments are characterized by remaining stationary during the planning and learning of the agent (Sutton and Barto 1998; Watkins and Dayan 1992). In synchronous environments, the time to perform the individual components of the SARSA algorithm protocol has no impact on task performance. Specifically, the time it takes to react to a new game state in chess without imposed time limits has no influence over the end of the game. In asynchronous environments, however, the time it takes for the agent to react to new observations can drastically influence its performance on the task. Such as, in the formulation of a cart-pole, the agent applied its actions *left* and *right* at discrete time intervals (Barto, Sutton, and Anderson 1983). These time intervals were set small enough so that the pole would not fall further than the agent would be able to recover.

As a concrete example, imagine an asynchronous environment called Hallway-World with a left turn leading to the terminal state, as shown in Fig. 3.1. The agent starts an episode near the bottom of a hallway and has two actions: *move left* and *move up* which move the agent in a direction and continue to move the agent in that direction with constant velocity until interrupted with the other action, hitting a wall, or arriving in the terminal state. If the agent hits a wall it receives a reward of -1 and comes to a stop. When the agent reaches the terminal state it will receive a negative reward directly proportional to the duration of the episode. In this way, the agent is motivated to get to the terminal state as quickly as possible without touching the walls. The only observation that the agent can make is to determine if there is a wall on its left.

The optimal policy in Hallway-World is for the agent to *move upward*

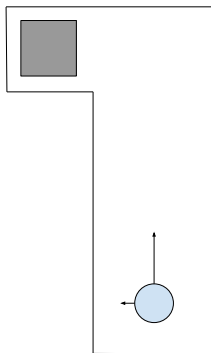


Figure 3.1: The Hallway-World task with the agent (the blue circle) starting near the bottom of the hallway. The gray square denotes the terminal state. The arrows denote the 2 actions which move the agent leftwards or upwards and continue moving the agent in that direction until interrupted.

and observe the wall continually until an opening in the wall is observed then immediately move leftwards towards the terminal state. SARSA (Alg. 2) is unable to learn this optimal policy because it is restricted by the delay between observing the opening in the wall and moving towards the terminal state. Even in the best case scenario, that is, if the SARSA agent observed the opening in the wall just as it appeared, it would not be able to act on this observation until it had spent time learning about the previous action and observation. Assuming that each component of the algorithm (acting, observing, choosing an action, and learning) takes some constant amount of time t_c , if a SARSA agent observes an opening in the wall, it must choose to move left and learn about the previous state-actions before taking the action, this would add $2t_c$ onto episode time, thereby affecting the total reward and task performance. Thus, overall performance in Hallway-World decreases with the time the agent spends selecting an action and learning, irrespective of how these components are performed.

3.3 Reactive SARSA

To minimize the time between observing a state and acting upon it, we propose a modification to conventional TD-control algorithms: *take actions immediately after choosing them given the most recent observation.* Reactive SARSA

is one example of this modification (Alg. 3); in each iteration of the learning loop, the agent observes a reward and new state, chooses an action from a policy based on the new state, immediately takes that action, then performs the learning update based on the previous action.

Algorithm 2 SARSA: An on-policy TD control algorithm

Initialize $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Repeat (for each episode):
 Initialize S
 Choose A from S using policy derived from Q (e.g., ϵ -greedy)
 Repeat (for each step of episode):
 Take action A
 Observe R, S'
 Choose A' from S' using policy derived from Q (e.g., ϵ -greedy)
 $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
 $S \leftarrow S', A \leftarrow A'$

Algorithm 3 Reactive SARSA: A *reactionary* on-policy TD algorithm

Initialize $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Repeat (for each episode):
 Initialize S
 Choose A from S using policy derived from Q (e.g., ϵ -greedy)
 Take action A
 Repeat (for each step of episode):
 Observe R, S'
 Choose A' from S' using policy derived from Q (e.g., ϵ -greedy)
 Take action A'
 $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
 $S \leftarrow S', A \leftarrow A'$

The slight reordering of RL algorithm protocol components does not affect convergence in discrete time. Here, we provide a basic theoretical sketch that in discrete-time synchronous tasks, Reactive SARSA learns the same optimal policy as SARSA, in the same manner. This equivalence is trivially evident by observing that in both algorithms the first 2 actions are selected using the initial policy. In each subsequent step t , actions are chosen using the policy learned on the last step, and the policy updates happen with identical experiences, as is illustrated in Fig. 2.

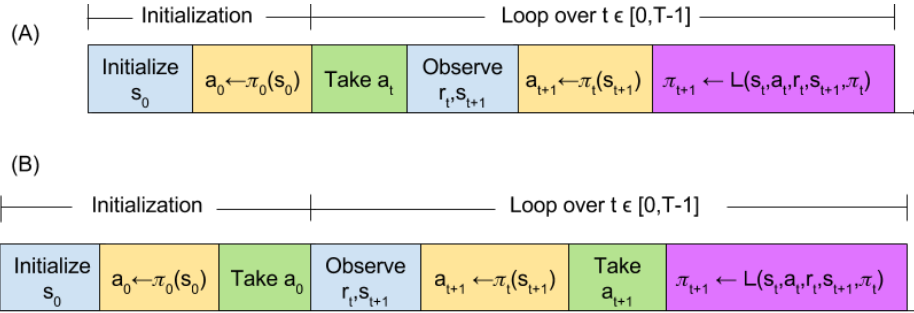


Figure 3.2: Timestep comparison of (A) standard and (B) reactive reinforcement learning algorithms. The function L refers to a learning function which updates the policy π . The learning function L is not limited to the SARSA learning update and encompasses any learning update such as Q-learning.

If we redefine Hallway-World as a synchronous environment where the agent moves a constant distance for each action instead of continually moving, the same policies and performance would be expected between both algorithms and this is what we found in practice. The difference between reactive and non-reactive algorithms is the order of the RL components (acting, observing, choosing an action, and learning).

3.4 Experiments

To explore the differences between the SARSA and Reactive SARSA learning algorithms in asynchronous environments, we designed a reaction-time-dependent task similar to Hallway-World as described above and illustrated in Figure 3.1. The task was performed using one joint of a robotic arm (see Figure 3.3). We conducted two experiments with the same episodic stopping task. The arm started at one extreme of the joint rotation range and was then rotated quickly towards the other end of its range. The agent must stop the rotation as soon as possible following an indication to stop that is observed by a state change from “Normal” to “Emergency”.

The agent had two actions: *stop* and *move*. If the agent chose to *stop* while in the “Normal” state, the agent would receive a reward of -1, remain in the “Normal” state, and the arm would continue rotating. If the agent chose



Figure 3.3: Experimental setup, showing the robot arm in motion for the first experiment (left) and the robot arm poised to impact an egg during the second experiment (right).

to *move* while in the “Normal” state, the agent would receive a reward of 0, remain in the “Normal” state, and continue rotating. Once the “Emergency” state had been observed, the reward for either action would be a negative reward proportional to the amount of time (in μs) spent in the “Emergency” state. When the agent chose to *stop* in the “Emergency” state, it transitioned to the terminal state, thereby ending the episode. This reward definition was chosen as a convenient means of valuing reaction time; the distance traveled during the reaction time is a valid alternative.

If complete information about the stopping task was available, optimal performance could be obtained through direct engineering of a control system designed to stop the arm as soon as the state changes. However, the agent did not know which state was the “Emergency” state and used its experience to learn what to do in any given state. By excluding the complex state information of many real-world robotic tasks and using a simple stopping task, we elucidate the effects of reaction time on overall performance, and the differences between conventional and reactive TD-control algorithms can be investigated.

3.4.1 Experiment 1

To explore the effect of the reactive algorithms on reaction time and task performance, the Bento Arm was programmed to move at a constant velocity along a simple trajectory (see Figure 3.3, left) (Dawson et al. 2014). The experiment involved 30 trials, each of which was comprised of 20 episodes with the agent starting in a “Normal” state and switching to the “Emergency” state after some uniform random amount of time. The standard and Reactive SARSA agents were compared with greedy policies, $\gamma = 0.9$, $\lambda = 0.9$, and $\alpha = 0.1$.

In any learning system, there may be time delays in the learning step. One example comes from the idea of predictive knowledge representations. Here knowledge is represented and learned as a collection of predictions about an agent’s observed experience. Such knowledge may be updated and computed during each cycle. One approach to building this knowledge is the Horde architecture; Horde introduces the idea of *demons* (also referred to as *general value functions*), which learn predictions about the environment and can build on each other to achieve a scalable method of knowledge learning (Sutton, Modayil, et al. 2011). A Horde architecture with 2576 demons (predicting the position, velocity, temperature, load and other measures) was experimentally validated on the robotic arm. On the experimental setting tested, this setup resulted in an average computation time of one demon’s prediction to be $\sim 3.33\mu s$. The more predictions one wants to make, the longer the duration of learning, thus the reaction time increases. Specifically, the time delays of 50ms, 100ms, 250ms, and 500ms on the experimental hardware are equivalent to Horde architectures of approximately 15000, 45000, 75000, and 150000 demons, respectively. It is clear that more predictions increase the reaction-time, thus adding time delays in the following experiments appropriately simulates the addition of more predictions. To simulate the performance of these additional predictions and modulate in a controlled fashion the effect of longer learning steps, time delays were added to the learning update step.

Figure 3.4 shows how the the duration of learning influenced the task per-

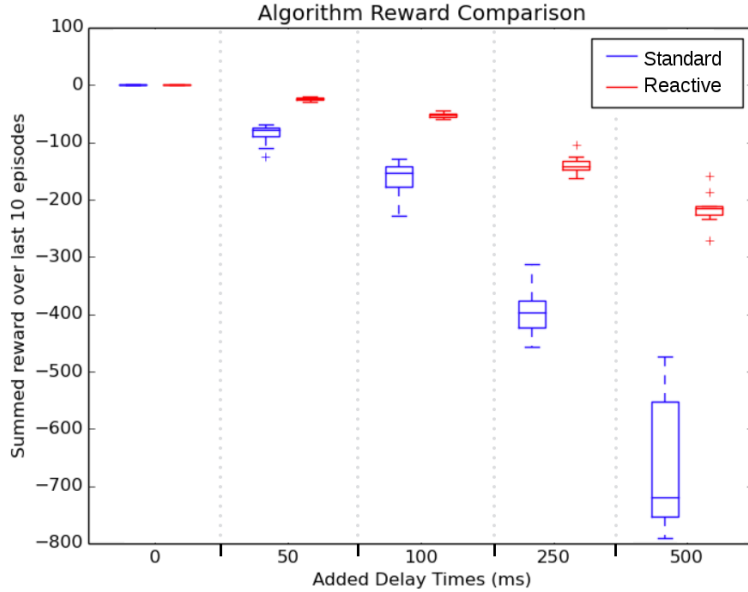


Figure 3.4: Key result: a comparison of summed reward over the last 10 episodes of 30 trials across 5 different learning delay length lengths during robot arm motion. Reactive SARSA had a significantly reduced reaction time when compared to the standard SARSA algorithm for all delay lengths.

formance. The figure shows the average cumulative episodic return for the last 10 episodes, once both agents had learned policies. As the delay increased, both algorithms suffered performance decreases, but the standard SARSA algorithm performed worse with larger variability. While Reactive SARSA was affected by increasing time delays, the impact was less severe. Specifically, the median reaction time of Reactive SARSA was approximately half of the added learning delay. This effect is most likely because the transition from “Normal” to the “Emergency” state occurred at a uniformly random selected time. Since the majority of the duration of a timestep was comprised of learning, the random timing lead the state change to occur, on average, halfway through the learning step. This also accounts for the increasing reaction time in the standard SARSA implementation, as the agent must wait a full additional timestep before reacting to the “Emergency” state.

3.4.2 Experiment 2

The second experiment considered a human-robot interaction task which demanded cooperation between a human and a robotic arm to not crush an egg (see Figure 3.3, right). The robot arm was positioned above a target, which in this task was an egg, and would move at a constant velocity toward the target. The human was told to press a button to stop the arm before crushing the egg, and to try to stop it as close to the egg as possible without touching it. The learning task for the RL agent was to learn to stop as soon as the participant pressed a button. For the first 10 episodes of a trial, the participant trained using a hard-wired stopping algorithm which automatically stopped the arm when the participant pressed a button. We refer to this algorithm as the control condition. For the remaining 40 episodes of the trial, agents that had previously learned using the SARSA and Reactive SARSA algorithms were used. Each algorithm was used for 20 episodes, and the algorithm used was randomly chosen on each episode. The state changed from “Normal” to “Emergency” when the participant pressed a button. All three algorithmic conditions: 1) control, 2) SARSA, and 3) Reactive SARSA, included a constant 50ms delay to simulate a longer learning step (e.g., the time it would take to update the predictions for 15000 demons). The algorithm used on a trial was hidden from the participant. Four individuals participated in the experiment, providing a total of 80 episodes of each algorithm. All participants provided informed consent as per the University’s Ethics Review Board and could voluntarily end the experiment at any time if they wished.

Figure 3.5 shows the total of all failed stops (“broken eggs”) for each algorithmic condition as summed across all four participants. Reactive SARSA had fewer failed stops than the standard SARSA agent which performed significantly worse with more than four times as many failed stops.

In addition to comparing the number of failed stops, and thus crushed eggs, of each algorithmic condition, the time between the state change from the button push of the participant and the reaction time of the agent was recorded and is presented in Figure 3.6. The effects of longer learning on reaction time

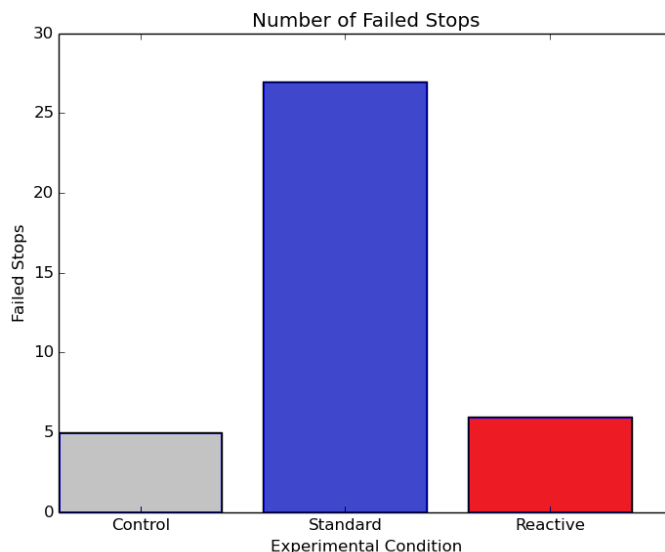


Figure 3.5: The total number of failed stops for each algorithm during the robot’s acceleration toward a breakable object (Experiment 2), summed over all four participants. For all subjects, Reactive SARSA had far fewer failed stops than the standard SARSA.

are evident in this figure, as the standard SARSA agent’s *stop* action is trailing behind the button press by approximately the length of the learning update and delay; this is contrasted by the tight overlap of the stimulus and action for the Reactive SARSA agent.

3.5 Discussion

Our results indicate that rearranging the fundamental components of existing TD-control algorithms (act, observe, choose action, learn) has a beneficial effect on performance in asynchronous environments where task performance is reaction-time dependent. A reactive agent can perform better in these environments as it can act immediately following observations. This effect becomes especially prominent as the duration of learning operations increases. Although the current experimental design added a simulated delay to the learning update step, our results indicate that as the time between observing and acting grows, performance in these environments deteriorates, regardless

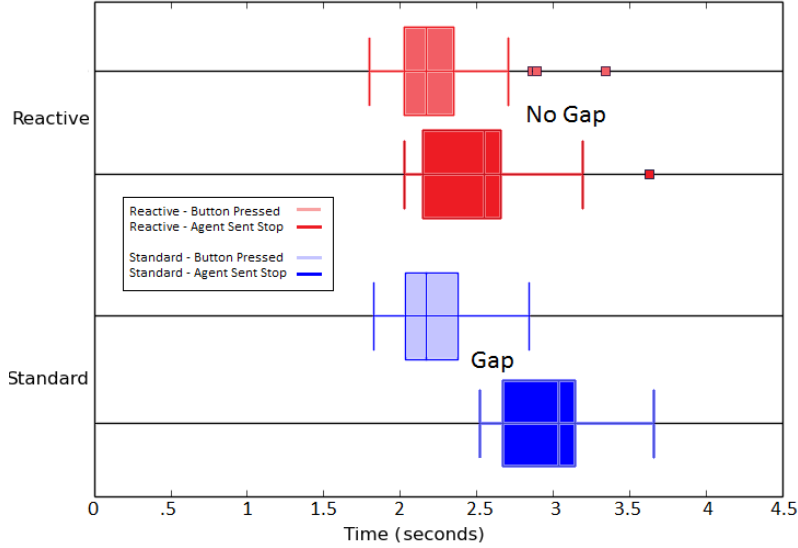


Figure 3.6: Boxplot comparison of the distributions of events over all episodes between Reactive SARSA and the standard SARSA algorithm. Zero on the x-axis is the moment the arm begins moving. The overlap of the button press and reactive agent’s action indicates that the reactive agent has negligible delay in its reaction to the participant’s input (seen in the overlap between light red and dark red, top). The standard agent’s ability to act is delayed by the length of learning (visible gap between light blue and dark blue, bottom).

of the source of these delays. As standard online RL algorithms perform learning and state representation construction between observing and acting, additional computation time is necessary (e.g., tile-coding (Sutton and Barto 1998), deep neural networks (Silver et al. 2016), etc.). In asynchronous environments, as these steps become longer, the order of algorithmic components (acting, observing, choosing an action, and learning) becomes more critical. As we have shown in Figure 3.4, performance in asynchronous environments that favor short reaction-times has an inverse relationship to the total length of time between observations and acting.

One alternate means of addressing delay-induced performance concerns may be to create a dedicated thread for each of the RL algorithm components (Caarls and Schuitema 2016). We believe this is a promising area for continued research. However, as multi-threading is difficult on single-core machines such as micro-controllers, reactive algorithms as suggested in this work may have

great utility when applied to embedded learning systems or smaller single-thread computers. While the order of algorithmic components might seem at first like a minor implementation detail, it may prove critical when applied to these systems. Reactive SARSA, and similar reactive algorithms, do not require multiple threads. Even in the case of multi-threaded systems, it may still prove fruitful to re-order the fundamental components of RL learning for improved performance. For example, if a state requires minimal reaction time, one may forgo a learning update and queue it for a later time when fast reactions are not required. We believe allowing an RL agent to learn an optimal ordering of its learning protocol or to interrupt learning components for more pressing computations are interesting subjects of future work.

3.6 Conclusions

RL algorithms are built on four main components: acting, observing, choosing an action, and learning. The execution of any of these components takes time, and while this may not affect synchronous discrete-time environments, it is a critical consideration for asynchronous environments, especially when task performance is tied to the reaction time of the agent. If learning is done correctly, an agent should never have to wait to take an action after receiving up-to-date observations. In this chapter we present a novel reordering of the conventional RL algorithm which allows for faster reaction times. We present a simple proof for the algorithmic equivalence in synchronous discrete-time and show improved performance in an asynchronous continuous-time stopping task which is directly linked to agent reaction time. These results indicate that 1) reaction time is an important consideration in asynchronous environments, 2) the choice of when in a loop the RL agent should act affects an agent's reaction time, 3) reordering of the components of the algorithm will not affect an agent's performance in synchronous discrete-time environments, 4) reactive algorithms reduce the reaction time, and thus improve performance, potentially also decreasing the time it takes for an agent to learn an optimal policy. This work, therefore, has wide potential application in real-world settings where

decision making systems must swiftly respond to new stimuli.

Chapter 4

Dealing with Increasing Dimensionality¹

4.1 Overview

To respond appropriately to the intentions and needs of their human users, prosthetic limbs and other assistive rehabilitation technologies rely on their sensors. In the specific case of clinically prescribed upper-limb prosthetic devices, the electromechanical sensor information available to a device’s control system is typically limited to mechanical toggles or a small number of myoelectric (EMG) signals recorded from the tissue of the user’s residual limb (Castellini et al. 2014; Pilarski and Hebert 2017). These sensors provide enough information to design a prosthetic solution wherein a patient may be able to directly control one or two prosthetic actuators (Parker, Englehart, and Hudgins 2006). The use of machine learning approaches such as pattern recognition allow available sensors to be further leveraged to increase the number of functions controllable by a user (Scheme and Englehart 2011a). There is now convincing evidence that machine learning control approaches such as pattern recognition can enable patients with amputations to sequentially

¹A version of Chapter 4 has been published and presented as Jaden B. Travnik and Patrick M. Pilarski, “Representing High-Dimensional Data to Intelligent Prostheses and Other Wearable Assistive Robots: A First Comparison of Tile Coding and Selective Kanerva Coding”, for the *Proc. of the 2017 IEEE International Conference on Rehabilitation Robotics (ICORR)*. London, United Kingdom, 2017 where it was presented as a poster. An extended abstract of this paper was also presented as a poster and a 60 second spot-light talk at the Multidisciplinary Conference on Reinforcement Learning and Decision Making (RLDM), Ann Arbor, Michigan, June 11-14, 2017.

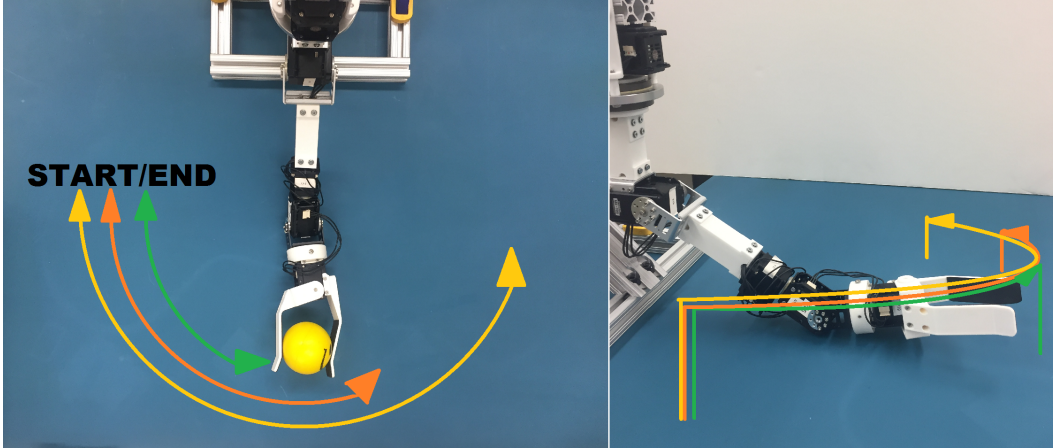


Figure 4.1: The upper-limb research prosthesis used in this study (the Bento Arm (Dawson et al. 2014)). The Bento Arm generates a continuous stream of position, velocity, load, voltage, and temperature sensor signals for each of its five actuators during its ongoing operation.

control a device with a robotic elbow, wrist, and hand with multiple discrete grasp patterns—far more degrees of control than were previously possible with conventional control solutions (Castellini et al. 2014; Scheme and Englehart 2011a). This increase in function can be attributed both to an increase in the number of sensors deployed within a prosthetic socket, and the efficient use and synthesis of the information provided by these sensors. The combination of sensorimotor signals into a useful summary of a system’s state, termed a *state representation*, has become increasingly important to the performance of prosthetic devices, especially those that leverage pattern recognition, regression, or real-time machine learning (Castellini et al. 2014).

As sensors have played a critical role in increasing the capabilities of clinically deployed prostheses, pre-clinical research prostheses have also continued to evolve in terms of their sensorimotor space to try and meet the function, form, and feedback needs of users with amputations (c.f., Antfolk et al. 2013; Atkins, Heard, and Donovan 1996; Castellini et al. 2014; Pilarski and Hebert 2017). As one representative example, Fougner et al. have shown that the addition of sensors to help resolve residual limb position (e.g., accelerometers or inertial measurement units) can dramatically increase the performance of myoelectric pattern recognition as a subject with an amputation moves their

limb through a range of common positions (Fougner et al. 2011). Further, even without the addition of new sensors to a prosthetic socket, the modern actuators and sensors within multi-joint prosthetic limbs can now generate a wealth of data of different frequencies, ranges, and modalities. If used carefully, these signals present a valuable window into the intent of a human user and their prosthesis’ interactions with a changing, daily-life environment.

4.1.1 Towards Computationally Efficient Representations

A prosthesis must be able to approximate and react to its sensor inputs within a short window of time in order to remain effective (Farrell and Weir 2007). If a representation can be formed for a control system in an efficient way, even in the face of high-dimensional sensor data, it can readily be *computed, stored, and used in real time* on the computationally limited hardware embedded within wearable prosthetic devices.

Previous work by our group has explored in detail the use of real-time machine learning methods for prosthetic control—we have to date performed a wide range of prosthetic prediction and control studies wherein subjects with and without amputations used different techniques from the field of reinforcement learning (RL) to operate a robotic arm (Edwards 2016; Edwards, Dawson, et al. 2016; Edwards, Hebert, and Pilarski 2016; Pilarski, Dawson, Degrís, Carey, Chan, et al. 2013; Pilarski, Dawson, Degrís, Fahimi, et al. 2011; Pilarski, Dick, and Sutton 2013; Sherstan, Modayil, and Pilarski 2015). In all of these studies we exclusively relied on the linear representation method known as tile-coding (Sutton and Barto 1998) to provide our RL prediction and control systems with a view into the available space of prosthesis- and user-derived sensorimotor signals. These linear representations were chosen because they are highly efficient in both computation and data usage for a fixed (small) number of input signals. However, as prosthetic devices improve and the number of sensors available to the system increases, it remains unclear how standard linear representations like tile-coding are affected, and if more scalable representations exist which would be more aptly suited for use in prosthetic limbs and other wearable assistive rehabilitation technologies.

Finding a good representation of input signals in complex, high-dimensional, and continuous environments is a difficult problem faced not just in the adaptive control of prosthetic limbs, but in a wide range of domains where designers seek to deploy machine learning for real-world control tasks; notable examples range from speech recognition to self-driving vehicles. However, unlike enterprise-scale applications with access to vast computational resources, the prosthetic setting requires that all computation be limited to run on small, power efficient processors that can be installed in a self-contained way within the chassis of a prosthetic device. The representation of high-dimensional continuous state information therefore needs to be done in a computationally efficient fashion. Furthermore, and specific to robotic applications, it is often impossible to exactly represent the state of a system to a control process; some approximation of the continuous state of the system must be used. One method of approximating continuous signals and reducing high dimensional data is principal component analysis (PCA) which compresses high dimensional data to a much smaller set of salient features. However, recent studies have found that the relevant variables PCA produces are not usable by a machine learner that requires representations of the interactions between sensorimotor data to perform well (Legenstein, Wilbert, and Wiskott 2010).

In this chapter, we therefore contribute a first study on the effects of increasing the dimensionality of prosthetic sensory data in terms of computation time and prediction accuracy for linear tile-coding representations (the dominant linear function approximation approach used in RL), and propose a novel approach, *Selective Kanerva Coding*. Modified from *Kanerva Coding*, *Selective Kanerva Coding* promises to scale accurately and efficiently as the number of sensory dimensions increases on an assistive device (Kanerva 1988; Ratitch and Precup 2004).

4.2 Representation using Linear Methods

Approximating a value function using linear methods is considered a standard approach because of the ease of computing the gradient of the value function

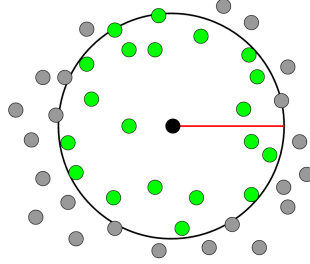


Figure 4.2: A depiction of Sparse Distributed Memory (SDM) prototypes being activated within a fixed radius of a specified address in the state space.

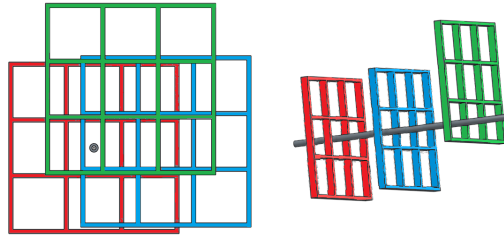


Figure 4.3: 2D and 3D views of three overlapping tilings covering a 2D state space with three tiles per dimension. The number of features for this example is 27, where only three features are activated as given by the grey dot and line inside their tiles.

with respect to the learned parameters or weights (e.g., the weight vector θ in Alg. 1) (Sutton and Barto 1998). This leads to a particularly simple form of the gradient-descent update with the guarantee that any convergence to a local optimum is a convergence to a global optimum. On top of their theoretical results, linear methods can excel in terms of computation and data efficiency but this depends critically on how the dimensions of states are represented in terms of the representation’s feature vector, ϕ . For instance, in continuous environments, it is natural for a single feature, $\phi(i)$, to represent a selected range of continuous values such as “joint A’s angle is between 150 and 180 degrees.” For complex tasks using linear methods, it is necessary to use features that represent combinations of state dimensions because linear methods, by their nature, are unable to represent interactions between their features.

4.2.1 Tile Coding

Tile coding (TC) is a linear representation that is often chosen in RL for its efficiency in online learning. Typically, tile coding splits up a d -dimensional state space into m overlapping partitions called tilings where each tiling is split into a set of n^d tiles, n tiles along each dimension. Each tiling has an offset from each other (see Figure 4.3), which leads to a better generalization (Sutton and Barto 1998). Each tile represents a binary (0 or 1) feature that is activated when the state lies within the tile. Finer granularity can then be achieved by increasing the number of tiles and decreasing the size of a tile in each tiling. Unfortunately, this granularity has a trade-off with generalization because states are less likely to activate the same tile.

A binary feature vector ϕ , of length mn^d , can then be constructed by concatenating all n^d features for all m tilings. Since only m features are activated, tile coding has an advantage when choosing a constant step-size parameter, α , as the number of active features is independent of the state. For example, a reasonable step-size parameter might be $\alpha = 0.1m$.

In order to capture the interaction between dimensions, tilings must span the state space so that each tile represents a small d -dimensional hyper-cube. The number of features grow exponentially with the number of dimensions. For the small number of dimensions found in common control tasks, tile coding provides an easily implementable representation with clear benefits. However, as the dimension of the task grows, so does the memory cost.

A common method of reducing the memory requirements is hashing, a pseudo-random compression of a large tiling into a reduced set of tiles of a given memory size, however, hashing still has some drawbacks. It is not clear how to choose a good memory size and some hand-tuning must be administered in practice. Additionally, the hashing function is a pseudo-random process which may have collisions leading to one feature being activated by two or more distinctly different sensory observations. Collisions may be accidentally beneficial if the collisions occur between sensory observations that

have similarities² and extend the generalization capabilities of the representation. However, the odds are usually in favor of a collision having negative affects. These tile collisions become more common as the memory size shrinks relative to the total number of tiles. Although it is possible to use chaining³ to make “safe” hashing implementations of tile coding that take preventative measures when a collision is detected, these checks take additional time and require still more time to handle a collision when one has occurred.

4.2.2 Kanerva Coding

Kanerva coding is the application of sparse distributed memory as a representation in reinforcement learning (Algorithm 4). Unlike tile coding, the number of features in Kanerva coding does not grow with the number of dimensions. This can lead to dramatic savings in necessary memory resources. Although a common issue when using Kanerva coding in RL is choosing an appropriate number of prototypes and activation radius, in contrast to other approximators, e.g., neural networks, the structural parameters of Kanerva coding can be easily changed without retraining the learned model. In fact, several researchers have shown improved performance by moving the prototypes around the representation space or by growing the representation using experience from the learner (Cheng and Meleis 2008; Ratitch and Precup 2004). These methods are effective but add extra memory and computational complexity to keep and examine different prototype statistics. Other work has shown that Kanerva coding may successfully be applied to continuous domains (Ratitch, Mahadevan, and Precup 2004). However, the computational cost of high-dimensional continuous states has not yet been explored. This is especially important in settings with limited computation and where the representation must be computed within a small amount of time.

²This is known as Locally Sensitive Hashing.

³A hashtable with chaining stores an array at each hash index which stores all of the collisions for the index.

Algorithm 4 Kanerva Coding

```
Initialize all  $K$  prototypes in  $P$  randomly
Choose an activation radius  $r$  and distance function  $d$  (e.g., Euclidean distance)
Given state  $S$ 
  For  $i = 0$  to  $K-1$ 
     $\phi_i \leftarrow 0$ 
    If  $d(P_i, S) < r$ 
       $\phi_i \leftarrow 1$ 
  return  $\phi$ 
```

4.2.3 Selective Kanerva Coding

We propose a method of finding nearby prototypes with minimal computation. The idea is to remove the activation radius and simply find the c closest prototypes to the state of the system using Hoare’s quickselect which can be used to find the c shortest distances in an array of distances (Hoare 1961). One way of choosing a good c is to choose a small ratio, η such that $c = \lfloor K\eta \rfloor$. Not only does this method, which we refer to here as *Selective Kanerva Coding* (SKC), still have the same $O(K)$ complexity as computing how many prototypes are within an activation radius, but it shares with tile coding the guarantee of the same number of activated features along with the associated benefits like selecting learning step sizes. Utilized alongside True Online TD(λ), SKC promises to be an efficient, reliable representation for computing GVFs.

A simple extension of selective Kanerva coding would be returning real-valued features. In this setting, after applying quickselect to find the closest c prototypes, the value of the closest prototypes’ features would be based on the prototypes’ proximity to the state relative to the other close prototypes while still ensuring that the total activation of the features remains equal to c . Further investigation into this real valued selective Kanerva coding is needed, as it has great potential utility for assistive technologies.

Algorithm 5 Selective Kanerva Coding

Uses *quickselect*(D, c) which finds the c smallest indicies in array D of length K in $O(K)$ complexity
Initialize all K prototypes in P randomly
Choose an η such that $c = \lfloor K\eta \rfloor \ll K$ and distance function d (e.g., Euclidean distance)
Given state S
 Initialize $D = \mathbf{0}$
 For $i = 0$ to $K - 1$
 $\phi_i \leftarrow 0$
 $D_i \leftarrow d(P_i, S)$
 $I \leftarrow \text{quickselect}(D, c)$
 For $i = 0$ to $c - 1$
 $index \leftarrow I_i$
 $\phi_{index} \leftarrow 1$
 return ϕ

4.3 Experiment

A robotic arm designed to be worn as a research prosthesis (the Bento Arm of Dawson et al., (Dawson et al. 2014), Figure 4.1) was used to generate data for a prediction problem in order to compare the time and prediction performance of selective Kanerva coding and tile coding. This robot arm has shoulder and elbow joints, wrist rotation and flexion, as well as a gripper for a total of 5 degrees of freedom. Each joint contained sensors for position, velocity, load, and temperature which leads to 20 real valued sensor signals from all of the servos (as shown using different coloured traces in Figure 4.1).

The arm was controlled in a sorting exercise where three different objects were picked up from the same location, carried, and then dropped at different locations assigned by their weight. A single trial consisted of the arm beginning at one end of its shoulder rotation, closing its gripper around an object, lifting its elbow while rotating its shoulder towards the drop off location at which point the elbow would descend, the gripper would release the object, and then the arm would return to the initial position. 30 trials were performed for each of the three objects, giving a total of 90 trials. Data from all servos was collected during each of these trials. Since each trial began and ended in the

same location, the order in which each trial was presented could be and was randomly shuffled. Five long streams of continuous sensor information were created using this method, each containing over 16 minutes of sensor readings.

The goal of the prediction task was then to predict what the expected cumulative sum of the shoulder angle sensor would be over the next 10 timesteps (~ 0.3 seconds into the future). Predictions were made through the use of an on-policy general value function learned via true online temporal-difference learning (Algorithm 1). In order to predict 10 timesteps into the future, $\gamma = 0.9$ was used in specifying the GVF. Learning rate and the eligibility trace parameters were empirically set to $\alpha = 0.001$ and $\lambda = 0.99$, respectively.

To examine the effect of the number of sensor dimensions on prediction performance and computation time, three different sensor combinations were used—representations were constructed with 2, 8, and 12 sensory input signals. In the 2-dimensional setting, only the elbow angle and load were given as part of the representation, whereas in the 8-dimensional setting, the angle, load, temperature, and velocity of both the elbow and the wrist rotator were given. Finally, the angle, load, temperature, and velocity of the elbow, the wrist rotator, and the wrist flexor were given in the 12-dimensional task.

Exactly 248 different combinations of tiles and tilings were used to generate a wide range of tile coding configurations and thus a wide range of features for each dimensional setting, ranging from 128 to 331776 features. One existing method of offsetting each tiling is to deterministically shift each tiling using a deterministic knight’s-move pattern⁴. Although this knight’s pattern is deterministic, by using the five different long streams of data generated by shuffling the trials, a representative distribution of online learning performance was calculated.

For selective Kanerva coding, the number of prototypes ranged from 1000 to 20000 where each prototype was a point in the 2, 8, or 12 dimensional space. Euclidean distance was used to measure how close each prototype was to the

⁴Similar to that in chess, the knight’s pattern used here shifted by an increasing amount for each new tiling, as per Miller and Glanz 1996. Each tiling produced by a shift is then included in the representation.

observed state after the state was normalized using the ranges of the sensors. A constant ratio $\eta = 0.025$ of features (rounded down) were active at any time. For example, in the 1000 prototype case, the features of the 25 closest prototypes to the observed state were activated. In the 8000 prototype case, the features of the closest 200 prototypes to the observed state were activated. To ensure that the performance was not dependent on the random distribution of the prototypes, five different seeds were used to randomly distribute the prototypes.

Thus each of the 248 configurations of tile coding and all five seeds of selective Kanerva coding were used for the 2, 8, and 12 dimensional settings using all five of the long streams of robot generated data. For each run, the prediction performance and the computation time per timestep was recorded. To reveal the relation between the number of features that could be calculated and the calculation time per timestep, the number of features that could be calculated within 100 ms was also recorded.

4.4 Results

Figure 4.4 compares the variation in both prediction error and computation time for tile coding and SKC as their number of features increases, up to a maximum computation time per step cut-off of 100ms (selected in this study as the upper limit for real-time prosthesis control, c.f., Farrell and Weir 2007). After removing outliers from the tile coding data which had error orders of magnitude worse than the trend seen in Figure 4.4⁵, the data indicates that both tile coding and SKC have improved performance as the number of features

⁵This was due to configurations of tile coding that had very few features.

Table 4.1: The maximum number of features that can be calculated within 100 ms on a single-core 2.5 GHz processor.

Dimensions	Tile Coding		Selective Kanerva Coding	
	Mean	Std	Mean	Std
2	247439	302	19333	236
8	244496	534	12960	198
12	245724	620	10883	165

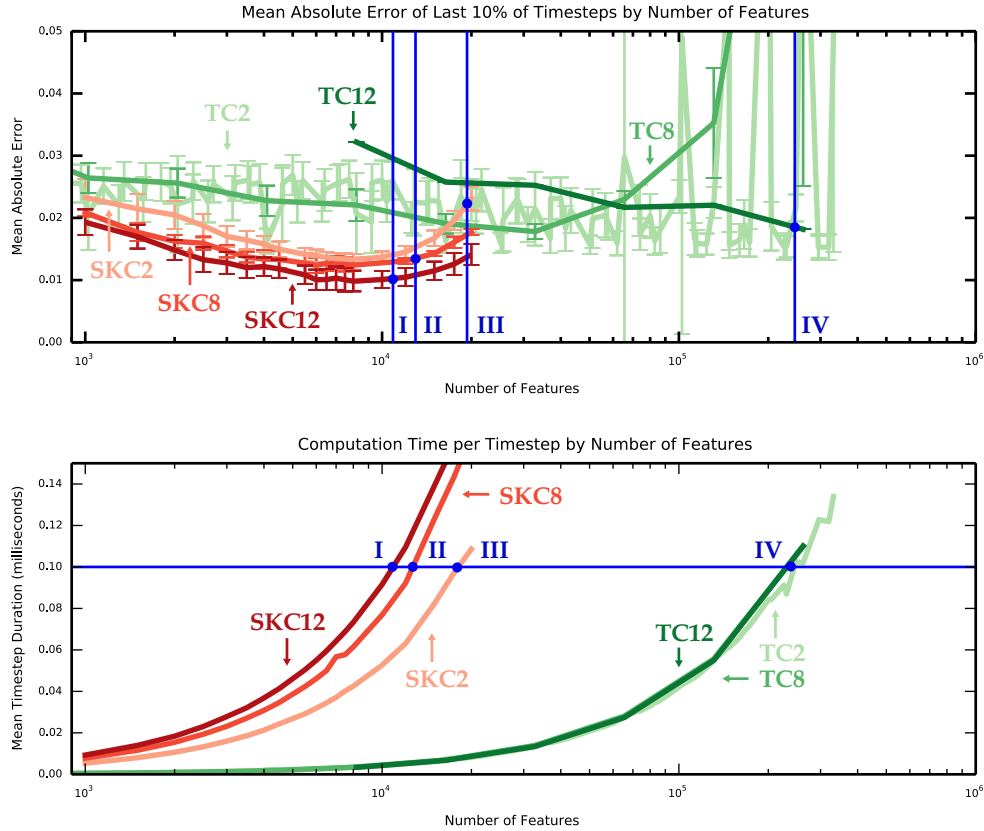


Figure 4.4: Key result: comparison of prediction error and computation time for tile coding and SKC as their number of features increases, including (top) the mean absolute prediction error on the last 10% of the combined data for SKC and tile coding on 2, 8, and 12 dimensions; and (bottom) the mean computation time of the combined data for SKC and tile coding on 2, 8, and 12 dimensions. The maximum features computable in a 100ms timestep is shown in both plots by blue lines and the points I, II, II, and IV for SKC12, SKC8, SKC2, and TC2/8/12, respectively

increases up until an optimum point after which the error increased or the representation took too long to compute. The maximum number of features calculated within 100 ms for the 12, 8, and 2 dimensional settings for SKC are represented as points I, II, and III, respectively. The maximum number of features that can be calculated within 100 ms using tile coding had little variation on the log scale presented in Fig. 4.4, and is represented by a single point IV. The exact numbers these labeled points represent is shown in Table 4.1.

The 100 ms processing limit had the effect that the 8 and 12 dimensional settings of tile coding did not improve the prediction error beyond what the 2 dimensional setting was able to achieve, despite the possibility of improved prediction performance if more computation was allowed. TC was quite unlike SKC, which not only had similar performance trends across different numbers of dimensions but utilized the additional dimensions of the state space to improve performance using the same number of prototypes and thus features. The best performance in terms of error across all conditions was found to be SKC at 8000 prototypes, with 12D SKC at 8000 features demonstrating the overall best performance on the entire experiment. After 8000 features, large numbers of additional features proved to be detrimental to SKC’s performance which resulted in the error having a convex form with a minimum at 8000.

Additionally, the timestep duration increased for both tile coding and SKC as the number of features increased. Not only was tile coding significantly more time efficient than SKC, but because there were tile coding configurations with the same number of features across different numbers of dimensions and each tile coding feature is calculated at the same speed, these different configurations had the same computation time per timestep and thus visually overlapped as seen in Figure 4.4 (bottom). As Euclidean distance was used to compute the distance between prototypes in SKC, the computation time per timestep increased with additional dimensions. As more dimensions were added, this extra computation decreased the number of features that could be calculated within 100 ms, as seen in Table 4.1.

Although SKC required significantly more time to calculate activated features, the extra time taken proved to have a stronger influence on prediction accuracy up until the optimal.

4.5 Discussion

Our results indicate that SKC is a representation that should be explored further within the context of prosthetic control, assistive or rehabilitation robotics, and other domains where high-dimensional continuous signals must

be efficiently represented in real-time to an adaptive or non-adaptive control process.

The five different random distributions of prototypes created for SKC did not lead to significant inter-distribution variations in performance (i.e., SKC had a consistently small standard error across prototypes distributions). This is an improvement over standard Kanerva coding, where the distribution of prototypes is known to play a significant role in the performance of the predictor. By only activating the features whose prototypes are the closest relative to other prototypes instead of within a radius, SKC invokes a limit on the distance between the furthest activated prototype and the state by the nature of distances in higher dimensional spaces. As the number of dimensions of a state space grows, the distances between random points in the space grows. In standard Kanerva coding, this growth along with the distribution of prototypes themselves increases the difficulty of appropriately setting the activation radius as it must grow as well. SKC, on the other hand, uses a notion similar to K-Nearest-Neighbors and locates the closest prototypes given that the majority of the prototypes (assuming $\eta < 0.5$) are further away. Following from the increasing distances between these prototypes, the set of features activated by SKC is appropriately flexible to changes in scale and dimension.

The error with respect to the number of features in SKC follows a convex curve which indicates that there is an optimal number of prototypes, given an η . This decrease in performance as excessive prototypes are added to the representation requires further investigation. It is most likely caused by the limited number of training examples but may also be caused by over-fitting. Following from previous work where the addition and deletion of prototypes has been explored, one could extend the present work by applying gradient descent methods to learn the optimal number of prototypes K , activation ratio η , and effective distribution of prototypes (e.g., via gradient derivations similar to those of Sutton, Maei, et al. 2009). This would be an important result, as it could lead to a representation with fewer prototypes, and thus features, that still accurately represents a high dimensional state space. With fewer features, the representation can be constructed faster and the extra time can be devoted

to making more predictions or to engaging more computation-heavy methods such as planning.

As the computation time increases with additional dimensions and features, there must exist an upper bound of how many dimensions and features can be represented on a single-core processor within a specified amount of time. The results indicate that SKC might be a representation that could provide accurate predictions until this upper bound is reached. Although this study explored the effect of higher dimensions while constraining computation time within 100ms, further studies are needed to consider the utility of different methods given variable time constraints and even higher dimensions.

Finally, it is natural to expect that to achieve the best performance on a prosthetic prediction or control task, the optimal number of or distribution of features in a SKC (or other) representation may be specific to each individual. That is, the best representation for a given prosthesis-user partnership may depend on the unique characteristics of an individual user's signals, behaviours, and the capabilities and operation of their prosthetic device. The best way to interpret signals for use in a machine learning or conventional control system may also change with time as the user's interactions with a device shift through experience and training. Exploring adaptive extensions to SKC and comparing them with other representation learning approaches to determine their viability on resource-constrained prosthetic control systems is therefore an important topic for future study.

4.6 Conclusions

As the number of sensors available on prosthetic devices grows with their capabilities, an appropriate synthesis of sensorimotor signals into a useful representation becomes vital to the performance of these devices' machine learning control systems. If such a representation can remain computationally efficient, it can readily be used on the computationally limited systems residing within wearable prosthetic technology. The study in this chapter explored how increasing the number of input signals affected performance and per step com-

putation time of a true-online reinforcement learning system using both tile coding and a new, modified version of Kanerva coding that we term *selective Kanerva coding*.

The presented results reaffirm previous findings about tile coding’s increasing computational requirements on high dimensional data. Our results further show that selective Kanerva coding can be readily applied to upper-limb robotic prediction tasks. We note that selective Kanerva coding takes more time to compute a representation than tile coding, but also show that not only are there significant gains in prediction performance with additional features but that there is an optimal number of prototypes for a fixed activation ratio, η and limited training data. These findings suggest that selective Kanerva coding merits further study, and as such, this work contributes a significant step towards accurately representing the high-dimensional data of assistive technologies to a machine learning control system such as a reinforcement learning agent.

Chapter 5

Selective Kanerva Coding in a Multimodal Domain¹

5.1 Overview

Intelligent robotic limbs represent the fusion of advanced robotics and machine intelligence, and are beginning to make their way out of the pages of science fiction into real-world applications. Previous work shows that intelligent arms have become useful for many tasks such as Amazon’s Picking Challenge (Hernandez et al. 2016) which requires an arm to pick up and place several objects in the shortest amount of time. Another application where intelligent robotic arms are becoming increasingly useful is as prosthetic devices where the robotic arm cooperates with a human user to allow synergistic movements (Sherstan, Modayil, and Pilarski 2015; Pilarski, Edwards, and Chan 2015; Pilarski, Sutton, and Mathewson 2015; DeGol et al. 2016).

As robotic limbs interact more and more with their environment, the ambition to build accurate knowledge of a complex and changing world becomes increasingly necessary. To address this problem, architectures have been proposed which show improved results on the 3-dimensional *Labyrinth* task by having multiple predictions of subtasks (Jaderberg et al. 2016), while other architectures show the expressive power of multiple predictions as knowledge (Sutton, Modayil, et al. 2011; Littman and Sutton 2002).

¹A version of Chapter 5 has been submitted for publication as Jaden B. Travnik, Dylan J. A. Brenneis, Michael R. Dawson, and Patrick M. Pilarski, “Grasping Predictions with Multimodal Sensors”, for *Frontiers in Neurorobotics*.

Vision systems combined with distance or inertial measurement sensors have previously been used to provide grasp prediction in powered hand prostheses (Došen et al. 2010; Ghazaei et al. 2017; Markovic et al. 2015) and robotic arms (Lenz, Lee, and Saxena 2015). The methods employed to make the grasp predictions include heuristics (Došen et al. 2010; Markovic et al. 2015) and deep learning (Ghazaei et al. 2017; Lenz, Lee, and Saxena 2015) where the sensors are placed near the back of the grasping hand (Došen et al. 2010; Ghazaei et al. 2017) or affixed to the head of the robot or user (Lenz, Lee, and Saxena 2015; Markovic et al. 2015). Using multimodal sensors with surface electromyography (sEMG) has also been identified as a promising approach for improving the control of myoelectric prostheses when mapping muscle contractions to joint movements on the prosthesis (Jiang et al. 2012). Several studies have found that combining inertial measurement data (accelerometer, magnetometer, and gyro) with sEMG can improve classifier accuracy when using pattern recognition classifiers such as linear discriminant analysis (Krasoulis et al. 2017; Kyranou et al. 2016; Markovic et al. 2015; Radmand, Scheme, and Englehart 2014). The main contributing signals were reported to be accelerometer, magnetometer, and sEMG while the gyro data was found to not contribute significantly (Markovic et al. 2015). Scheme et al. found that accelerometer data could sometimes degrade classification when trying to solve the limb position problem (Radmand, Scheme, and Englehart 2014). Their proposed solution was to train the classifier while moving the prosthesis dynamically through a range of static movements in which case the training time is reduced and performance is improved by using both the accelerometry and sEMG data instead of the sEMG alone.

It can be argued that making multiple accurate predictions requires multiple types of sensors, because the type of sensory information presented to a learning agent drastically affects the accuracy of the predictions it is able to make with the data. Take for example, a robotic arm tasked with predicting whether it will contact a red object. A robotic arm capable of sensing color would be expected to make more accurate predictions at this task than an arm without this sensory information. A challenge then lies in determining which

modes of sensory information will be the most important to a learner for any particular task: a process, referred to as feature selection, used in many machine learning applications (Guyon and Elisseeff 2003). The more relevant the subset of inputs are to the prediction to be made, the higher the prediction accuracy. In the arm example, if sensors relaying color information were available, they would be most relevant to predictions involving color. In this way, the type of sensors used on a robotic arm dictate what predictions can be made accurately irrespective of the representation used. If specific types of sensors are required to make accurate predictions which pertain to specific types of sensory information, it is a natural extension to see that multiple modes of sensors will be required to make predictions which pertain to multiple sensory modalities.

In this chapter we examine the effects of using different sets of multi-modal sensory inputs in a prediction task using real-time data on a robotic hand. We first document the custom robotic hand and data glove used for data collection in the experiment. We then introduce the design of experiment and provide a description of the data collected. Then, we detail the implementation of the representation and learning algorithms used, along with the 3 modality settings for comparison. After discussing the results, we present insight on the effects of using multi-modal data sensors in predictions of robotic arm data. Finally, we conclude that intelligent robotic arms of the future require multi-modal sensors.

5.2 Experiments

5.2.1 Data Collection

Four shapes differing in color and shape were 3D printed and a mapping between shape and hand gestures was chosen so that there was no association between any color and finger nor any shape and finger. As follows, we hypothesized that the most accurate prediction would have to have information about both modalities.

The HANDi Hand was set up such that the camera faced a solid black





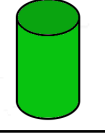
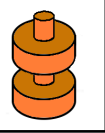
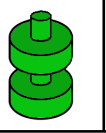
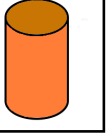
			
			

Figure 5.1: The table associating GLOVi Glove hand gestures to shapes. There are 4 shapes in all: green and orange cylinders, and green and orange steps. If the shape is the Orange Step, then the experimenter wearing the GLOVi Glove, puts their ring and pinky fingers down.

background, next to a pipet stand used to present objects to the hand in a repeatable orientation (see Figures 5.2 and 5.3). The objects varied in shape and color to include a green cylinder, an orange cylinder, a green stepped cylinder, and an orange stepped cylinder. These colors were chosen as it was expected that averaged RGB values collected by a camera would show noticeable differences between the colors. Similarly, the shapes were chosen as such because it was expected that the HANDi Hand would naturally grasp them differently, resulting in a noticeable difference in finger position. Each trial began with a recorded button press, followed by the introduction of the shape to the hand’s grasping area. Three seconds following the button press, the hand closed around the object in a column grasp, and held the object for three seconds. During this grasp phase, the experimenter wearing the data glove would perform a hand gesture particular to the object being held by the HANDi Hand, as outlined in Figure 5.1. The presentation of these hand gestures never deviated from the presentations in the table. Natural small deviations in movement of the experimenter’s hand (i.e. small inconsistencies in finger position or timing) were allowed, as they are representative of typical human interactions. Upon release of the grasp, the object was removed from the grasping area of the hand, and the experimenter returned the GLOVi Glove to a neutral position with all fingers extended. Three seconds following the release of the object marked the end of each trial, making each trial a total

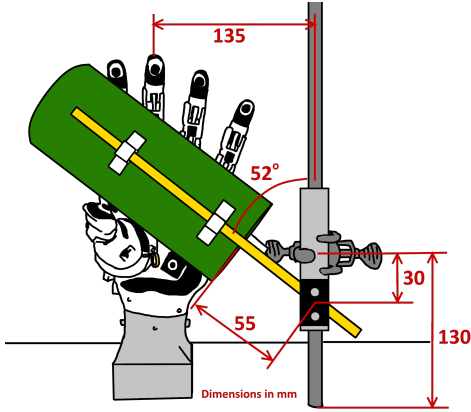


Figure 5.2: A side view of the experiment setup showing how the shapes were oriented in the HANDi Hand's grasping area.

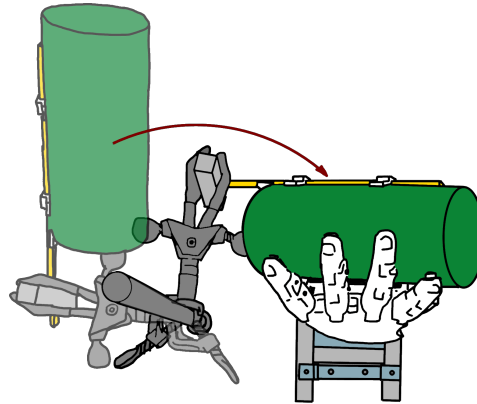


Figure 5.3: A top view of the experiment setup showing how the shapes were rotated into the HANDi Hand's grasping area.

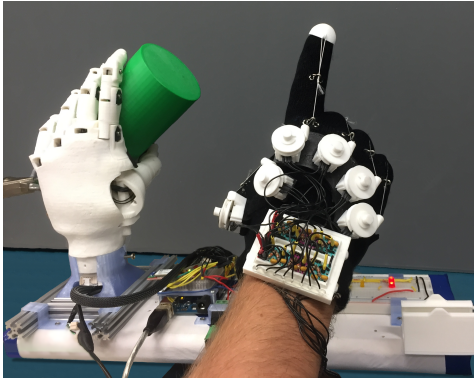


Figure 5.4: A picture showing the HANDi Hand (left) holding a green cylinder and the experimenter wearing the GLOVi Glove (right) performing the appropriate hand gesture as indicated by the table in Figure 5.1.



Figure 5.5: The camera's 480 x 640 RGB view of the green cylinder (top) against the black background with the Thumb of the HANDi Hand (left) in view.

of nine seconds long.

For each object, the number of trials recorded is as follows: green cylinder: $n = 31$; orange stepped cylinder: $n = 33$; green stepped cylinder: $n = 33$; orange cylinder: $n = 34$. Position and force data from the HANDi Hand was collected using an Arduino Mega at a rate of approximately 70 Hz; potentiometer data from the GLOVi Glove was collected on an Arduino Uno at a rate of approximately 750 Hz; camera data was collected via USB to the

computer at a rate of 15 Hz with an image size of 480x640 RGB. A timestamp was also recorded for each reading so that the data could be synchronized offline. As the highest data collection frequency was from the GLOVi Glove, the data from the camera and HANDi Hand was upsampled using the most recent timestamp. This resulted in a total of over 850,000 data points for all sensors. As each trial started and ended in relatively the same position with the GLOVi Glove and the HANDi Hand opened and an empty camera view, the order of the trials was randomly shuffled such that the trials of four shapes were randomly distributed into a long stream of sensory data. The sensory values were then normalized between 0 and 1 using the sensor’s minimum and maximum over the entire data stream.

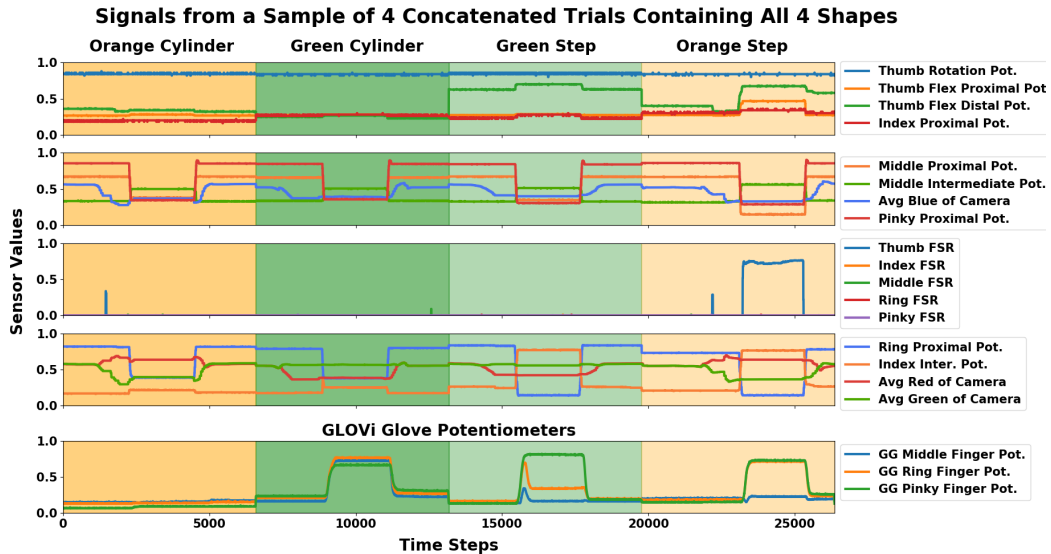


Figure 5.6: A sample of the data stream created by concatenating the different trials. The top 4 rows show different groups normalized sensor values from the HANDi Hand. The 4th row shows the subset of signals used in the “All” input setting. The 5th row shows normalized values of the Middle, Ring, and Pinky Potentiometers of the GLOVi Glove. The colored sections indicate which shape is being presented. Small jumps can be seen in the signals when trials meet. An exception is the Thumb Flex Distal Potentiometer which had large variations in its value as the thumb of the HANDi Hand drifted over the course of the experiment.

When the HANDi Hand closed around a step shape, the index and ring fingers closed further than they would have if the shape had been a cylinder.

This can be seen from Figure 5.6 in the 4th row. Further, the potentiometers of all other joints on the HANDi Hand either did not move (e.g., Thumb Rotation), had movement but were indiscernible across the different shapes (e.g., Middle Intermediate), or had a noticeable drift during the experiment which produced large jumps in value after the trials were concatenated together. As these large jumps were indicative of what shape was to be presented, the Thumb Flex Distal Potentiometer was not considered as an input signal for the experiment. Additionally, only the force sensitive resistor in the thumb of the HANDi Hand was activated and although it activated with different profiles between all shapes it was determined that this difference was a result of the drift in the HANDi Hand thumb flex joint which occurred during data collection so it was also not included as an input to the predictions. Likewise, since adding a non-relevant sensor would hamper the accuracy of a prediction and would make a prediction harder to interpret (Guyon and Elisseeff 2003; James et al. 2013), such signals were also not used as inputs to the predictions.

In this way, a manual feature selection ensured that the potentiometer of the intermediate joint of the index finger and the proximal potentiometer of the ring finger from the HANDi Hand together contained enough information to discern between cylindrical shapes and stepped shapes. In a similar manner, the averages of the red and green channels from the camera, unlike the average of the blue channel, contained enough information to discern between the green and orange shapes. In this way, a representative subset of 4 sensors was isolated as it provided the information necessary to discern each shape and thus predict each finger for the GLOVi Glove. For each of the 4 sensor values, a trace with $\lambda_{decay} = 0.999$ was calculated and normalized as well. Three different input space settings were constructed using this subset of sensors and associated traces. A “Color Only” setting was created using the averages of the red and green channels from the camera and their normalized traces thus making a 4 dimensional input space. A “Position Only” setting was created using the potentiometers of the HANDi Hand fingers from the subset along with their normalized traces making another 4 dimensional input space. Finally, an “All” input space was constructed by using all 4 sensory signals from the subset and

their associated normalized traces making an 8 dimensional input space.

Each of the 3 different input space settings was represented using selective Kanerva coding with 8000 features and an $\eta = .025$ with input dimensions respective to the input space setting (Travnik 2017). Three on-policy general value functions, one for each of the moving finger’s potentiometers on the GLOVi Glove (middle, ring, and pinky finger), were implemented with learning parameters $\gamma = 0.999$ and $\lambda = 0.9$ and the learning target of their respective GLOVi Glove potentiometer. Each of these general value functions was used to learn a prediction of their potentiometer with a step size of $\alpha = 0.01$ for 80% of the data then was tested on the last 20% of the data using a step size which did not modify the learned weights ($\alpha = 0$). For each of the 177419 timesteps of the testing phase, the true return of the potentiometer was calculated with $\gamma = .999$ and the squared error between the prediction and true return was recorded.

5.3 Results

After collecting the data, the predictions of each of the three general value functions on the last 20% of data, where there was no learning ($\alpha = 0$), were plotted along side the potentiometer signal and its true return. The predictions, potentiometer values, and the true return, for one trial of the Green Cylinder are plotted in Figure 5.7. As the Green Cylinder shape required the experimenter wearing the GLOVi Glove to put down their middle, ring, and pinky fingers, the values for the potentiometers of middle, ring, and pinky have a distinct square pulse as the potentiometer turned with the spooling mechanism, and the associated true return has a curve indicative of this pulse. The accuracy of the predictions (blue) for these square pulses on this trial vary across the different input space settings (columns), with the “All” setting having a better approximation of the true return than the other 2 settings. The “Position Only” predictions of the ring, and pinky fingers have a large offset before the onset of the experimenter’s finger movements but were able to have a fairly close approximation of the drop off of the return during the

step function. The “Color Only” predictions did not have a large offset but were not able to accurately predict any of the potentiometer values. The “All” input setting was able to have both the lower initial offset similar to the “Color Only” input setting and an accurate approximation of the of the return during the rest of the trial.

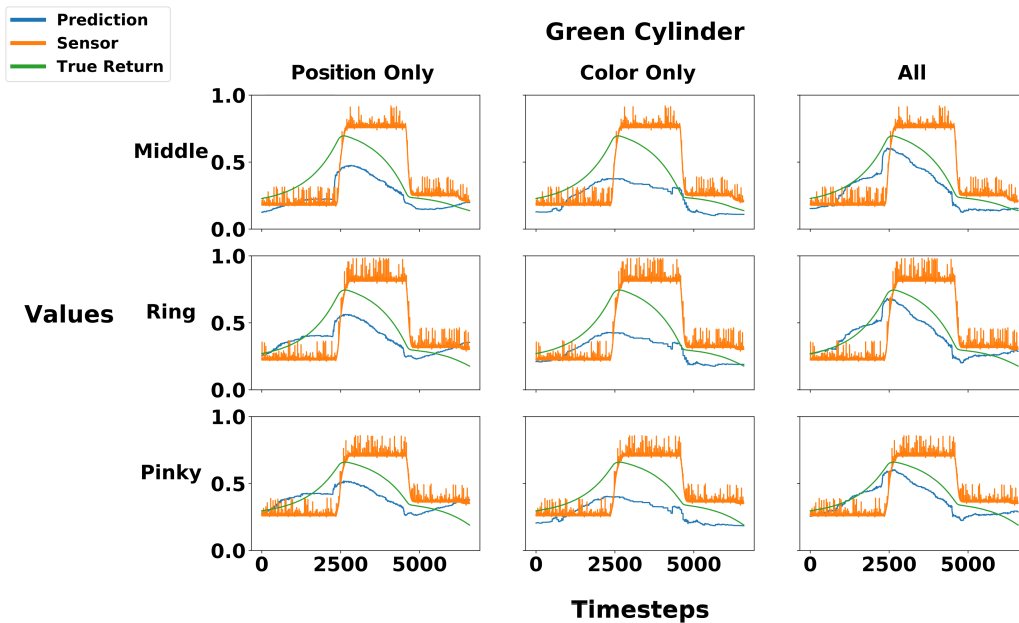


Figure 5.7: A plot of the predictions for one of the Green Cylinder trials in the testing phase. For each of the three generate value functions, the prediction of the potentiometer for each of the input space settings (blue) is plotted against the value of the potentiometer sensor (orange) and the sensor’s true return (green). Following the hand gestures from 5.1, the experimenter wearing the GLOVi Glove put down their middle, ring, and pinky fingers for the Green Cylinder shape as can be seen from the sensor (orange) lines.

To compare the rest of the trials, the distributions of the root squared error between the true return and the predictions after learning were compared for each prediction and input setting (see Figure 5.8). Accross all input settings, the distribution of error for the middle finger has lower variance than those of the ring finger and pinky. The distributions in error for the “All” input space setting have a smaller variance and a lower mean and median for the middle and pinky potentiometers compared to the other two input space settings. The median for the ring finger in the “All” input setting is almost identical with

the median for the ring finger in the “Position Only” input setting but the variance for the “All” input setting on the ring finger is slightly smaller.

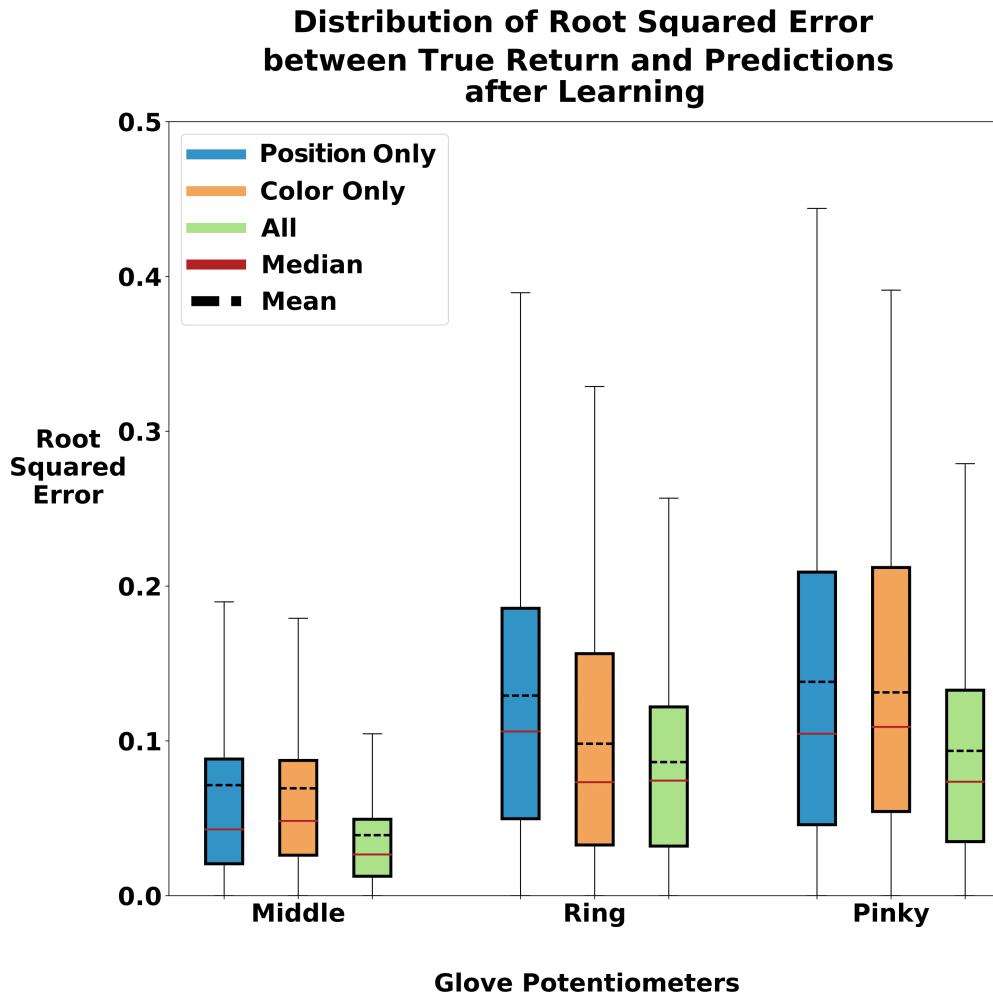


Figure 5.8: A side-by-side boxplot of the distributions of the root squared error between the true return and the predictions of the general value functions during the testing phase. Each distribution is comprised of 177419 data points. Outliers outside of 2 standard deviations are removed for clarity.

Although a two-tailed two-sample t-test proved statistical difference (p -value ≈ 0) between the distributions of error of the “All” input space setting and the other two input space settings, a Cohen’s D test was performed to find the effective difference between the distributions of error (Cohen 1977; Lin, Lucas Jr, and Shmueli 2013). A small to medium effective difference was observed between the “All” input space setting and the other 2 settings for

all of the fingers with the exception of the ring finger on the “Position Only” setting.

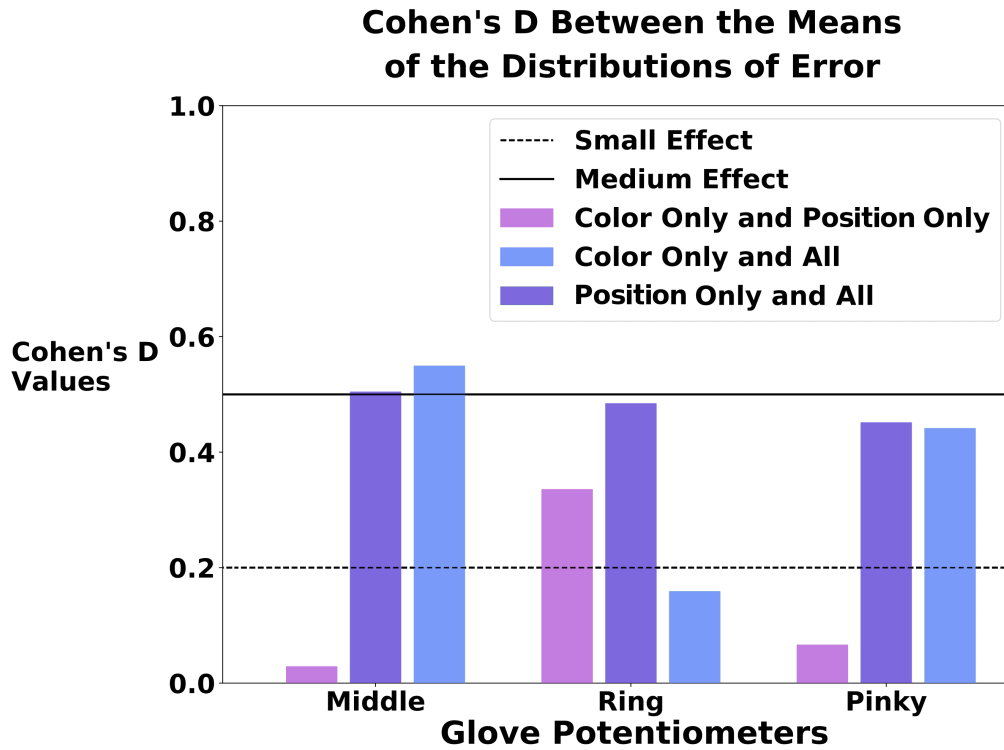


Figure 5.9: The results of a Cohen’s D test between the distributions of error of the different input settings for each prediction.

5.4 Discussion

As can be seen in Figure 5.7, the predictions more closely approximate the true return when the data from all sensory modalities is given to the learner. This same result is supported by the analysis of error distributions given in the boxplots of Figure 5.8. Perhaps the most interesting feature of Figure 5.7 is the bottom central plot; that of the pinky finger prediction as made using only the color data. For this particular prediction one might expect reasonably good accuracy, since if the learner knows that the color of the object is green, regardless of the shape, the pinky finger of the GLOVi Glove should be predicted to move down, giving the square pulse. The prediction

that is made here does not fully reflect the true return unless the data from the HANDi Hand fingers is also included, as seen in the “All” column. Thus, it appears that even when it is expected that a particular subset of sensors might be enough to give a reasonable prediction, additional data from different sensory modalities can be used to increase prediction accuracy.

The results of a Cohen’s D test show that the middle, ring, and pinky predictions made in the “All” input setting not only have a small to medium effective difference between the distributions of error of the “Color Only” and “Position Only” input spaces (see Figure 5.9), but with the addition of sensors with different modalities, the root mean squared error of the predictions decreased along with the median for most of the predictions. Additionally, the variances of the distributions of error for the middle, ring, and pinky predictions were narrower than those of the other two input settings (see Figure 5.8). The higher error seen in the “Color Only” input setting predictions is evident given the large offsets between the predictions and the baseline positions of the potentiometer sensors.

The performance seen in the “Position Only” input setting predictions suffered in part due to the lack of information about the incoming object which was available in the “Color Only” input setting. As the “Position Only” input setting did not provide enough information to tell the difference between the green and orange objects, each prediction made using that input setting was instead offset based on the frequency of the associated finger moving². The “All” input setting did not have this drawback and had an accurate approximation of the return during the entirety of the trial.

In the “Color Only” input setting, the ring and pinky predictions tended to increase just before the hand gesture in the GLOVi Glove was made. This was surprising, as this jump occurred just as the HANDi Hand closed. Analyzing the data after the fact, the jump in prediction occurs because the HANDi Hand thumb is in the frame of the camera, so when the HANDi Hand closed around a shape, the averages of the colors also changed. In fact, for the Step

²The Pinky was moved in 75% of the trials, the Ring in 50% of the trials, and the Middle in only 25% of the trials.

shapes, the thumb rotated into the groove of the step and further out of the frame of the camera than it did for the cylindrical shapes. This made a small but noticeable difference in the average of the colors such that the predictions in the “Color Only” input setting could discern when the hand was closed.

5.5 Conclusions

Using a custom robotic hand and inexpensive data capture glove, comparisons were made between the prediction accuracies of different learning scenarios with particular subsets of sensory modalities. The results from this study show that the inclusion of relevant multimodal sensors to an input space of a machine learner can increase the prediction accuracy over the prediction accuracy of machine learners that only include sensors of a single sensory modality. In cases where it might be expected that reasonably accurate predictions could be made using a single sensory modality, the introduction of alternate modalities still increased the prediction accuracy. This finding seems to suggest that machine learning systems that have access to a variety of different types of sensory information might be able to generate more accurate predictions than those with more limited sensory inputs. Indeed, this makes intuitive sense: we as human beings integrate information from a wide variety of sensory inputs to make predictions about the world we interact with, and when deprived of our senses (for example, by being blindfolded), we tend to have more trouble.

The future of artificial intelligence systems will depend on the generation of accurate predictions in order to build up expansive knowledge of the complex world that an agent acts in. The findings of this study suggest that these accurate predictions rely on diversity in the sensory input space, hinting that future robotic applications will benefit by the inclusion of multimodal sensory inputs. In the specific case of machine learning as applied to prosthetic limbs, this means that robotic arms should be made to include many different kinds of sensors.

Chapter 6

Discussion and Exploration of Selective Kanerva Coding

The experiments in Chapters 4 and 5 show how selective Kanerva coding (SKC) can be used as an effective linear function approximator with large state dimensions and across different sensor modalities. In this chapter we explore the sensitivity of SKC’s meta-parameter η on prediction accuracy, evaluate possible variations of SKC, and discuss future research opportunities.

The following experiments reuse the HANDi Hand and GLOVi Glove data from Chapter 5. Specifically, they use the same subset of 4 signals from the HANDi Hand data: the potentiometer of the intermediate joint of the index finger, the proximal potentiometer of the ring finger, the averages of the red and green channels from the camera. Each following experiment also uses the ring finger from the GLOVi Glove as the target signal to be predicted by on-policy GVs. The learning parameters of $\alpha = 0.01$, $\gamma = 0.999$, and $\lambda = 0.9$ were also kept the same as well as a data trace of $\lambda_{decay} = 0.999$.

6.1 Sensitivity Analysis of η

To analyze the sensitivity of SKC’s prediction accuracy to changes in η , an encompassing sweep across varying numbers of prototypes and varying values of η was performed. To account for variations in the spacial distribution of prototypes, 40 iterations, each with a different random seed were used. Using the same strategy as in Chapter 5, GVs were implemented and trained for

80% of the data and then had their predictions compared against the true return for the last 20%. The mean squared error between the prediction and the true return for each GVF was calculated and recorded. The resulting profiles for the prototype set size and the η value used can be seen in Figure 6.1.

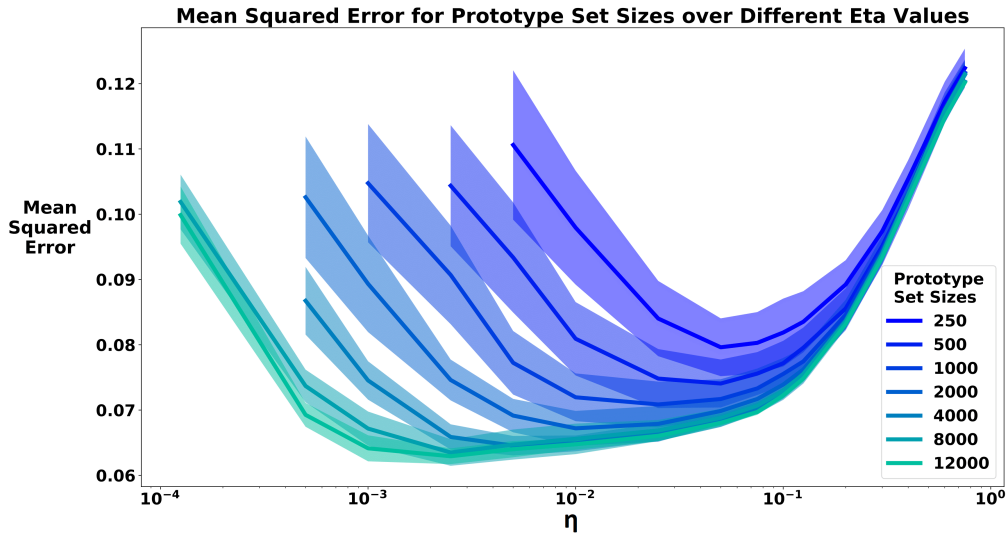


Figure 6.1: The sensitivity in error for different values of η of varying sizes of prototype sets. The profiles for the smaller prototype set sizes are staggered as an $\eta < \frac{1}{K}$, where K is the prototype set size, results in 0 active prototypes so they can not be evaluated at these values. The smaller prototype sets (e.g., 250 to 4000) show an increase in variability for small η s.

The plot in Figure 6.1 captures a few important qualities about SKC. Firstly, it reaffirms that additional prototypes can decrease the prediction error with diminishing returns, at least until a point as shown in Chapter 4. Secondly, it shows that the variation in different prototype distributions affects the variability of prediction error more strongly for small prototype distributions. This variability is easily attributed to the larger variability in the average distance between prototypes as when smaller prototype sets are randomly initialized (see Figure 6.2). Finally, together with the results from Chapter 4 Figure 4.4, Figure 6.1 shows that for a given GVF which requires a prediction within a limited amount of time, if a GVF can be predicted using selective Kanerva coding, there exists a global minimum selective Kanerva

coding representation with a specific prototype set size and η which gives the least prediction error while remaining within the limited time constraint. For instance, although the setting with 12,000 prototypes performed the best out of all of the other settings, it also took longer to compute one step of the learning algorithm than 0.0013 seconds, the maximum amount of time allowed for the algorithm to stay up to speed with the GLOVi Glove signals which were read at a rate of 750 Hz. Although reducing the rate of sensor readings would allow for more time to compute more prototypes, the diminishing returns of additional prototypes show that it is not worth the effort.

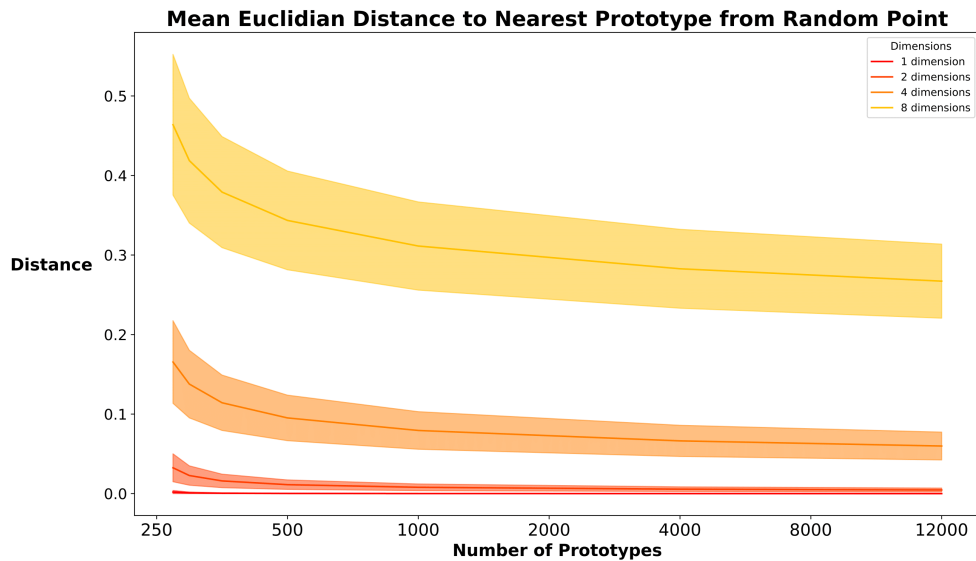


Figure 6.2: The distribution of the distance between a random point in $[0, 1]^n$ space, for $n \in \{1, 2, 4, 8\}$, to the nearest prototype as the size of the prototype set increases. 50 random seeds were used to initialize each prototype set for each dimension setting. 200 random points were then generated and the distance between them and the closest prototype was recorded.

6.2 Multiple Feature Sets

Implementations of tile coding often employ a simple technique to increase the granularity of their representations. The idea is to have multiple overlapping tilings which may even have different resolutions. It is reasonable to ask if

this technique can translate to SKC and if it would improve performance. Specifically, we would like to have improved prediction performance without extra memory or computation requirements, namely no additional features or prototypes. In this section we describe two possible scenarios of how one could implement multiple overlapping prototype sets and evaluate their performance against vanilla SKC.

6.2.1 Multiple Prototype Sets

A very simple change to SKC is to have multiple overlapping prototype sets as seen in Figure 6.3. Essentially, one generates two or more sets of prototypes across the same state space and uses each independently to create a feature vector. Then all of the feature vectors are concatenated together to form a longer feature vector whose length is equal to the total number of prototypes. This final feature vector is given to the learning agent.

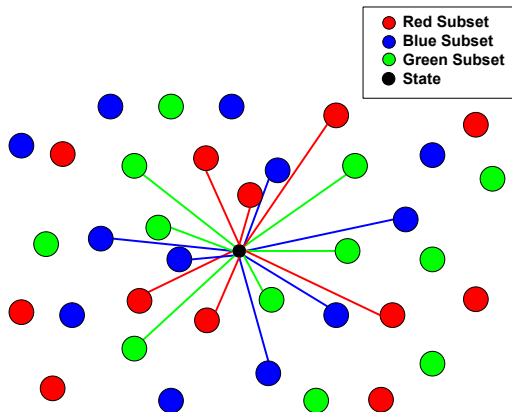


Figure 6.3: A high-level example of 3 overlapping prototype subsets using $\eta = .5$. The lines between the state (black) and the prototypes (red, green, and blue) indicate that the feature associated with the prototype is activated.

For example, the SKC with 8000 prototypes from section 6.1 could be broken up into multiple smaller SKCs; 2 subsets of 4000, 4 subsets of 2000, or even 16 subsets of 500. The subsets do not have to be all the same size, however, so a subset of 1500 together with a subset of 6500 is reasonable. Although the total number of prototypes or features does not change, and the

computational complexity for each subset remains the same as the original SKC algorithm given the same total number of prototypes, the total time to complete training, and thus the length of a single timestep, varies with the size of a subset and number of subsets as seen in Figure 6.4. Together with the error plots in Figure 6.5, this suggests although a representation comprised of a few small subsets (e.g., 4 sets of 2000 prototypes) has worse performance than a single set of 8000, it can be computed in roughly half of the time.

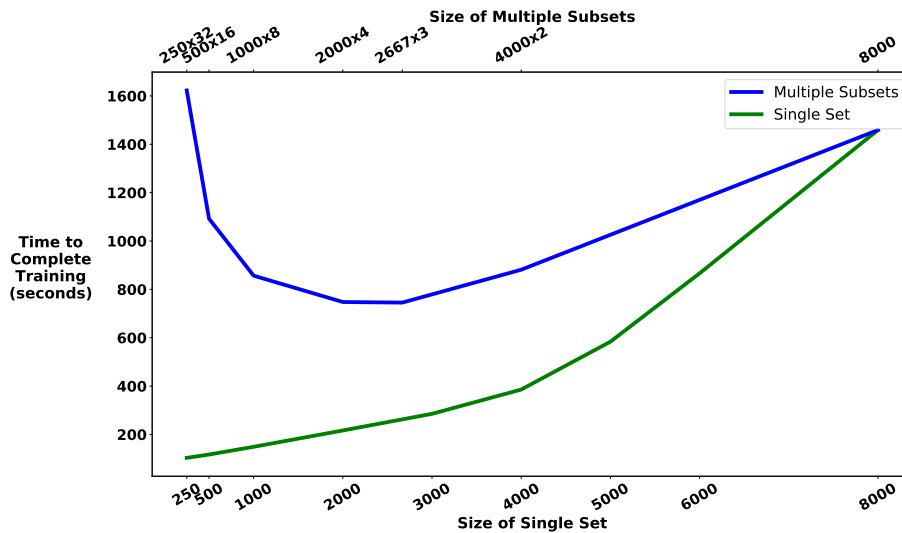


Figure 6.4: A time comparison between single prototype set representations (lower x-axis, green), and multiple prototype subset representations (upper x-axis, blue) using $\eta = 0.05$. Similar to the curves in Figure 4.4, the green curve indicates that the time to compute a single set representation increases with its size. The bowl shape in blue indicates that although the time to compute the one subset representation grows with its size, the total time to compute all of the subsets depends on the number and size of the subsets.

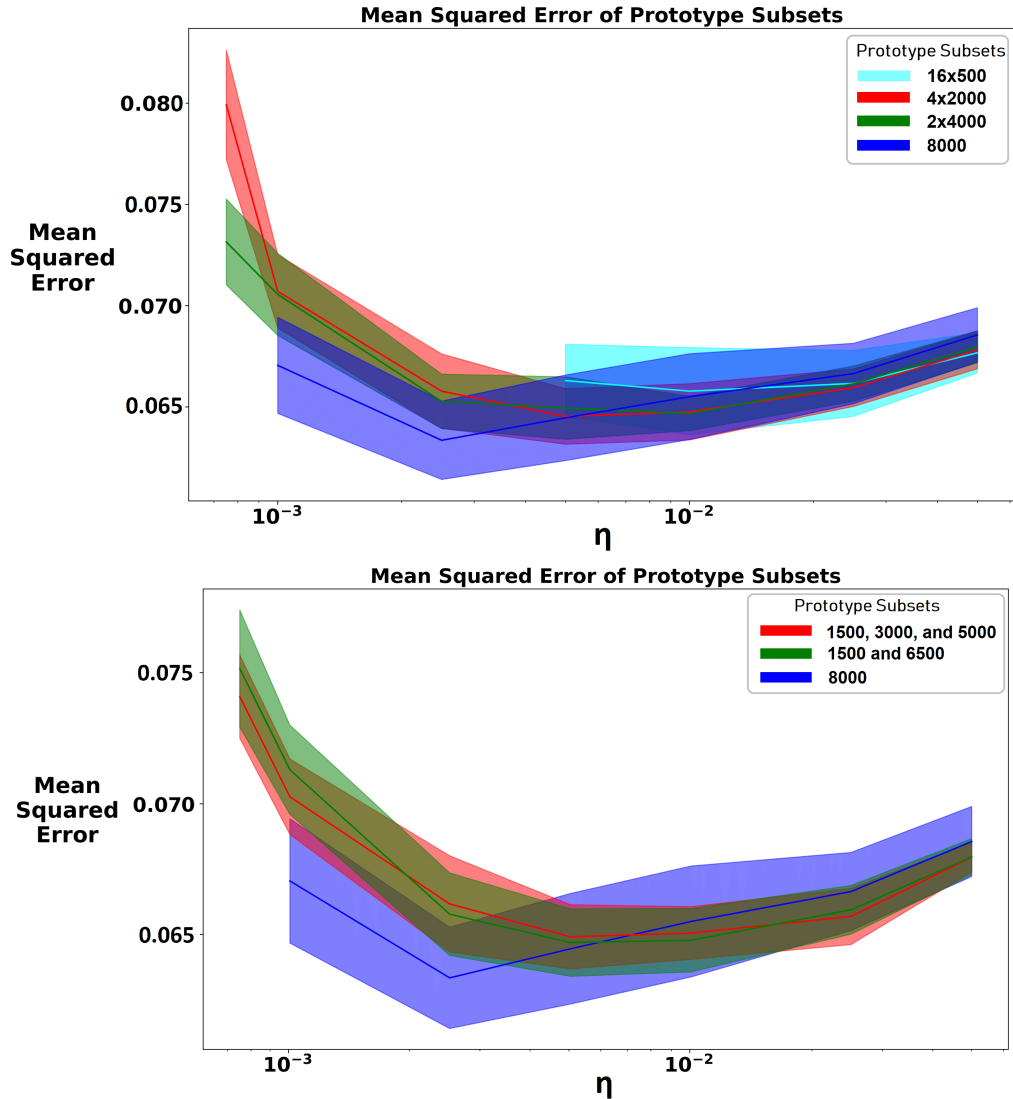


Figure 6.5: The error sensitivity profiles of different prototype subsets across different values of η . The smaller a prototype subset (e.g., 1500), the more variability in prediction accuracy for small η s.

The results in Figure 6.5 compare examples of multiple prototype sets, as listed above, on the same prediction problem of predicting the ring finger signal of the GLOVi Glove. These results do not indicate any advantage to using multiple prototype sets although they do verify that the fewer prototypes that are in a subset, the more sensitive the predictions are to the variability in distance as described by Figure 6.2.

6.2.2 Multiple η Values

An alternative to having multiple prototype subsets is to instead have one set of prototypes and have multiple different values of η as seen in Figure 6.6. In this variation, features for each η_i are calculated and concatenated together. In this variation, every prototype is represented by N features where N is the total number of η s used. If a prototype happens to be the closest to the input, then all N of its features will be on as it will satisfy the activations of every η . Each of these features will then be used with a different weight during the weight update step.

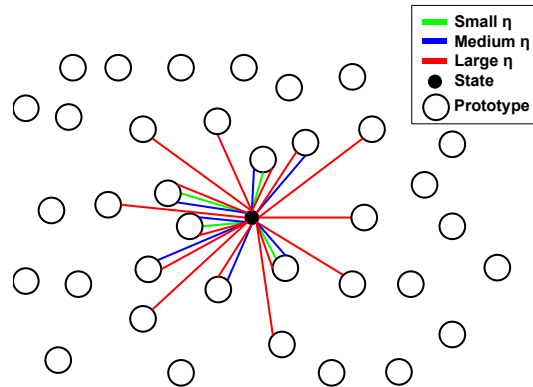


Figure 6.6: A high-level example of 3 η s on the same prototype set. The different lines (red, green, and blue) between the state (black) and the prototypes (white) indicate that a feature associated with the prototype and that η is activated.

This variation may be faster than the either of “normal” or the multi-prototype set settings, given a feature vector length. There are two reasons. First, since there are fewer prototypes to store in memory, there are few distances to calculate. Second, by starting with the largest η in the set, the resulting partition can be used as the starting set for the second largest η as every prototype not in this partition is too far away to be considered by the smaller η . For example, in one experimental run with 4000 prototypes, we computed the partitions for $\eta = 0.1, 0.05$, in order.

By reducing the number of prototypes necessary in memory, there is more room for extra weights to be learned. For instance, comparing once again to

the same baseline of 8000 prototypes and one η , one could imagine having a set of 4000 prototypes with two η s, meaning that there are half as many prototypes in memory but the same number of weights (8000) in either case. Towards the extreme, one could also imagine a set of 500 prototypes and 16 different η s, using 1/16th of the prototypes to calculate 8000 features and learn 8000 weights. Comparing these different settings against the best of the original SKC formulation gives the results presented in Figure 6.7. There seems to be little difference between the 4000 prototype settings other than a small decrease in variability when using more η s. A much more noticeable change is evident when using 500 prototypes. By using 16 η values ranging between 0.0025 to 0.075, the error drops substantially although not as good as the performance seen in the original SKC setting with 8000 prototypes and $\eta = 0.0025$.

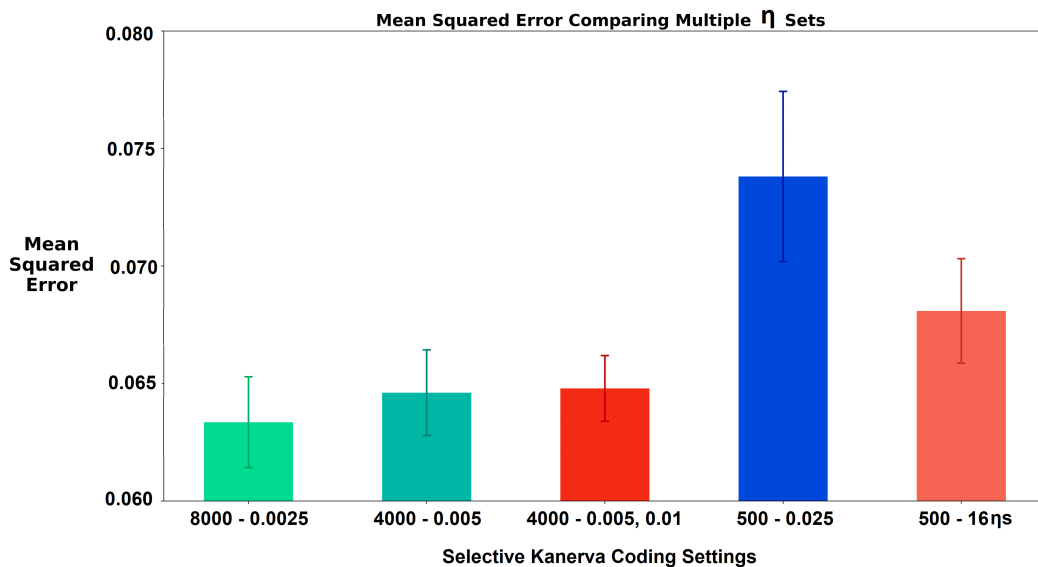


Figure 6.7: A comparison of the error for the best η for normal 500, 4000, and 8000 prototype sets against the error for 4000 prototypes with two η s and 500 prototypes with 16 η s ranging between 0.0025 to 0.075.

6.3 Future Research Areas

6.3.1 Optimal Algorithm Ordering

In Chapter 3, we proposed the class of Reactive learning algorithms as which aim to minimize the time it takes to respond to new stimulus. Reactive-SARSA fits within this class of algorithms by the simple reordering of the components of the classic algorithm from: observe, act, learn. A future research area going beyond this hard-coded reordering is to allow an agent the freedom to decide when it should learn, act, or observe.

As a thought experiment, imagine an oracle-agent that has perfectly modeled its environment, knowing the outcome of every possible action. If this environment is asynchronous and provides more positive rewards for completing a task as quickly as possible, then in order for this oracle-agent to maximize its reward, it should eliminate all unnecessary computation routines, as they delay the agent. Since it has perfectly modeled its environment, learning does not improve its model. Moreover, if by predicting the state using its perfect model, the agent can achieve a perfect state prediction without observing, observation is also an unnecessary computation. Thus the oracle-agent can eliminate learning and observing and simply act. Experts, such as video-game speed runners or musicians, are sometimes able to perform their talents without actually observing the consequences of their actions. This is because they know their environment and task so well that they can simply act. By viewing the order of algorithmic components of learning algorithms as modifiable, an agent may be able to find an optimal ordering of its learning protocol or to interrupt long-lasting computations (e.g., analyzing an image) for more pressing computations (e.g., avoiding a pedestrian).

6.3.2 Future Work with Selective Kanerva Coding

Chapters 4 and 5 and the explorations in the previous sections of this chapter have shown that selective Kanerva coding (SKC) can readily be used for linear function approximation. Like tile coding, it asserts that a constant number of features are activated at any time which allows the same freedom when

selecting a step size that tile coding provides. The variations of SKC presented in the preceding sections highlight that SKC as a starting framework is highly malleable and can easily be transformed into alternative function approximators which may have their own properties such as reducing memory requirements.

With such a malleable starting point, there are many branches of future work. One tempting direction is to apply the strategies presented by Ratitch, Mahadevan, and Precup 2004. Their collection of case studies shows some methods to redistribute the prototypes of classic Kanerva coding such that they can have a finer granularity over important areas. Combining these techniques with SKC should minimize the sensitivity of SKC to its prototype distributions although at the cost of more computation time.

Chapter 7

Conclusion

The overall goal of my research has been exploring how reinforcement learning can be effectively applied on resource bounded systems. Specifically, I explored how reinforcement learning with linear function approximation be deployed on responsive resource bounded systems and what limitations one should be aware of when doing so. Understanding the additional limitations present when deploying a reinforcement learning agent on a resource bounded system provides insight into what an agent is capable of and can lead to much more scalable architectures when not constrained computationally. I defined an asynchronous environment to better model the difficulties faced with RL in real-time domains and introduced a linear function approximation algorithm that provides an effective alternative to previous linear function approximation methods. While my long-term goals include freeing an artificial agent to choose how best to organize its own data and cognitive functions, this thesis primarily focused on how one could begin to approach these problems on limited hardware. Current methods of reinforcement learning do not evaluate the time it takes to react to a new situation. Minimizing the reaction time of RL algorithms is an important problem as RL agents become more common in the modern world. Although rarely, if at all, considered, this problem is not unique to the robotic domains explored in the previous chapters but can be found in many resource bounded systems including smart phones, the IoT, and self-driving cars. It is obvious that fast reaction time is good, provided the chosen action is appropriate. From this, one might theorize, as I do throughout

this thesis, that control over one’s cognitive components would be a beneficial tool to intelligent agents.

This thesis looked at several different aspects of reinforcement learning on resource bounded systems. Chapter 3 introduced an asynchronous environment to better model the control problem an agent faces when deployed in a real-time domain. Chapter 3 also proposed a class of reactive algorithms to deal with the issues caused by asynchronous environments by minimizing reaction time and presented empirical results justifying the claims. These results would benefit from deeper investigation on how the arrangement of learning components can be learned by the agent. Chapter 4, which is arguably the most important contribution of this thesis, introduced selective Kanerva coding as a novel approach to linear function approximation. The method was evaluated on a prediction task using data from a robotic arm performing a sorting task. This successfully demonstrated that selective Kanerva coding showed improved prediction accuracy on the prediction task while reducing the number of features necessary to complete the task compared to tile coding. Together these results show that SKC is an effective approach to linear function approximation. Chapter 5 extended these results by exploring how SKC performs with different modalities in the state space. Typically the addition of sensors and different modalities brings a curse of dimensionality to linear function approximation techniques. Taken together with the findings in Chapter 4, these results indicate that SKC is less susceptible to this curse, allowing for more modalities to be presented to a learning agent, affording more accurate predictions. Chapter 6 explored the sensitivity of the meta parameters of SKC, presented two variations, and provided direction for future research areas.

During the writing of this thesis, the field of reinforcement learning was heavily impacted by significant advancements using deep learning, specifically deep convolutional neural networks (Silver et al. 2016). These and previous findings revolutionized and motivated many industries and much of artificial intelligence research to focus on deep learning and its applications. Contrary to this direction, I focused on domains of resource bounded systems deployed

in real-time environments. The nature of the available computation and memory of these systems often establish deep learning as not a viable option for function approximation. Further, the environments tend to reward fast reaction times which may not be feasible when applying deep learning on limited hardware. It is from this perspective that I set out to write this thesis. First, I introduced a new perspective on algorithm performance, namely that the time it takes to react to a new state of an asynchronous environment can impact the performance of a reinforcement learning agent. Secondly, I focused the rest of the thesis, chapters 4, 5, and part of 6, on novel linear function approximation in regards to prediction learning and showed improved prediction accuracy over common methods. Besides the importance of these contributions detailed in the previous chapters, the findings of this thesis provide a perspective that challenges the current state of the field of reinforcement learning.

References

- Antfolk, Christian, Marco D'Alonzo, Birgitta Rosen, Göran Lundborg, Fredrik Sebelius, and Christian Cipriani (2013). “Sensory feedback in upper limb prosthetics.” In: *Expert Review of Medical Devices* 10, pp. 45–54. 33
- Atkins, Diane J., Denise C. Y. Heard, and William H. Donovan (1996). “Epidemiologic overview of individuals with upper-limb loss and their reported research priorities.” In: *Journal of Prosthetics and Orthotics* 8, pp. 2–11. 33
- Badamasi, Yusuf Abdullahi (2014). “The working principle of an Arduino.” In: *Electronics, Computer and Computation (ICECCO), 2014 11th International Conference on*. IEEE, pp. 1–4. 5
- Baird, Leemon C. and A. Harry Klopff (1993). “Reinforcement learning with high-dimensional continuous actions.” In: *Wright Laboratory, Wright-Patterson Air Force Base, Tech. Rep. WL-TR-93-1147*. 15
- Barto, Andrew G., Steven J. Bradtke, and Satinder P. Singh (1995). “Learning to act using real-time dynamic programming.” In: *Artificial intelligence* 72.1-2, pp. 81–138. 19
- Barto, Andrew G., Richard S. Sutton, and Charles W. Anderson (1983). “Neuronlike adaptive elements that can solve difficult learning control problems.” In: *IEEE transactions on systems, man, and cybernetics* 5, pp. 834–846. 20
- Brenneis, Dylan J. A., Michael R. Dawson, and Patrick M. Pilarski (2017). “Development of the Handi Hand: An Inexpensive, Multi-articulating, Sensorized Hand for Machine Learning Research in Myoelectric Control.” In: *Proc. of MEC'17: Myoelectric Controls Symposium*. Fredericton, New Brunswick. 9
- Caarls, Wouter and Erik Schuitema (2016). “Parallel online temporal difference learning for motor control.” In: *IEEE transactions on neural networks and learning systems* 27.7, pp. 1457–1468. 19, 29
- Castellini, Claudio, Panagiotis Artemiadis, Michael Wininger, Arash Ajoudani, Merkur Alimusaj, Antonio Bicchi, Barbara Caputo, William Craelius, Strahinja Dosen, Kevin Englehart, Dario Farina, Arja Gijsberts, Sasha B. Godfry, Levi Hargrove, Mark Ison, Todd Kuiken, Marko Markovic, Patrick M. Pilarski, Rudiger Rupp, and Eric Scheme (2014). “Proceedings of the first workshop on peripheral machine interfaces: Going beyond traditional surface electromyography.” In: *Frontiers in Neurorobotics* 8, p. 22. 1, 32, 33

- Ceaparu, Irina, Jonathan Lazar, Katie Bessiere, John Robinson, and Ben Shneiderman (2004). “Determining causes and severity of end-user frustration.” In: *International journal of human-computer interaction* 17.3, pp. 333–356. 2
- Chen, Guangyu, B-T Kang, Mahmut Kandemir, Narayanan Vijaykrishnan, Mary Jane Irwin, and Rajarathnam Chandramouli (2004). “Studying energy trade offs in offloading computation/compilation in java-enabled mobile devices.” In: *IEEE Transactions on Parallel and Distributed Systems* 15.9, pp. 795–809. 2
- Cheng, Wu and Waleed M. Meleis (2008). “Adaptive Kanerva-based function approximation for multi-agent systems.” In: *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 1361–1364. 3, 38
- Cohen, Jacob (1977). *Statistical power analysis for the behavioral sciences (revised ed.)* New York: Academic Press. 57
- Coley, Gerald (2013). “Beaglebone black system reference manual.” In: *Texas Instruments, Dallas*. 5
- Cook, Albert M. and Janice M. Polgar (2015). “Chapter 12 - Technologies That Aid Manipulation and Control of the Environment.” In: *Assistive Technologies (Fourth Edition)*. Ed. by Albert M. Cook and Janice M. Polgar. Fourth Edition. St. Louis (MO): Mosby, pp. 284–313. ISBN: 978-0-323-09631-7. DOI: <https://doi.org/10.1016/B978-0-323-09631-7.00012-0>. URL: <https://www.sciencedirect.com/science/article/pii/B9780323096317000120>. 7
- Curran, William, Tim Brys, David Aha, Matthew Taylor, and William D. Smart (2016). “Dimensionality Reduced Reinforcement Learning for Assistive Robots.” In: *2016 AAAI Fall Symposium Series*. 3
- Dawson, Michael R., Craig Sherstan, Jason P. Carey, Jacqueline S. Hebert, and Patrick M. Pilarski (2014). “Development of the Bento Arm: An improved robotic arm for myoelectric training and research.” In: *Proc. of MEC’14 : Myoelectric Controls Symposium*. Fredericton, New Brunswick, pp. 60–64. 2, 7, 8, 25, 33, 40
- DeGol, Joseph, Aadeel Akhtar, Bhargava Manja, and Timothy Bretl (2016). “Automatic grasp selection using a camera in a hand prosthesis.” In: *Engineering in Medicine and Biology Society (EMBC), 2016 IEEE 38th Annual International Conference of the IEEE*, pp. 431–434. 48
- Degrís, Thomas and Joseph Modayil (2012). “Scaling-up Knowledge for a Cognizant Robot.” In: *AAAI Spring Symposium: Designing Intelligent Robots*. 19
- Došen, Strahinja, Christian Cipriani, Miloš Kostić, Marco Controzzi, Maria C. Carrozza, and Dejan B. Popović (2010). “Cognitive vision system for control of dexterous prosthetic hands: experimental evaluation.” In: *Journal of neuroengineering and rehabilitation* 7, p. 42. 49
- Edwards, Ann L. (2016). “Adaptive and Autonomous Switching: Shared Control of Powered Prosthetic Arms Using Reinforcement Learning.” Master’s thesis. University of Alberta. 7, 14, 16, 34

- Edwards, Ann L., Michael R. Dawson, Jacqueline S. Hebert, Craig Sherstan, Richard S. Sutton, K. Ming Chan, and Patrick M. Pilarski (2016). “Application of real-time machine learning to myoelectric prosthesis control: A case series in adaptive switching.” In: *Prosthetics & Orthotics International* 40.5, pp. 573–581. 34
- Edwards, Ann L., Jacqueline S. Hebert, and Pilarski M. Pilarski (2016). “Machine learning and unlearning to autonomously switch between the functions of a myoelectric arm.” In: *Proceedings of the 6th IEEE RAS/EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob2016)*, pp. 514–521. 16, 34
- Edwards, Ann L., Alexandra Kearney, Michael R. Dawson, Richard S. Sutton, and Patrick M. Pilarski (2013). “Temporal-difference learning to assist human decision making during the control of an artificial limb”. 16
- Farrell, Todd R. and Richard F. Weir (2007). “The optimal controller delay for myoelectric prostheses.” In: *IEEE Transactions on neural systems and rehabilitation engineering* 15.1, pp. 111–118. 34, 42
- Fougner, Anders, Erik Scheme, Adrian D. C. Chan, Kevin Englehart, and Øyvind Stavdahl (2011). “Resolving the limb position effect in myoelectric pattern recognition.” In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 19.6, pp. 644–651. DOI: 10.1109/TNSRE.2011.2163529. 34
- Ghazaei, Ghazal, Ali Alameer, Patrick Degenaar, Graham Morgan, and Kianoush Nazarpour (2017). “Deep learning-based artificial vision for grasp classification in myoelectric hands.” In: *Journal of Neural Engineering* 14.3, p. 036025. 49
- Guyon, Isabelle and André Elisseeff (2003). “An introduction to variable and feature selection.” In: *Journal of machine learning research* 3.Mar, pp. 1157–1182. 50, 54
- Hernandez, Carlos, Mukunda Bharatheesha, Wilson Ko, Hans Gaiser, Jethro Tan, Kanter van Deurzen, Maarten de Vries, Bas Van Mil, Jeff van Egmond, Ruben Burger, Mihai Morariu, Jihong Ju, Xander Germann, Ronald Ensing, Jan Van Frankenhuyzen, and Martijn Wisse (2016). “Team Delf’s Robot Winner of the Amazon Picking Challenge 2016.” In: *arXiv:1610.05514[cs.RO]*. 48
- Hester, Todd, Michael Quinlan, and Peter Stone (2010). “Generalized model learning for reinforcement learning on a humanoid robot.” In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on.* IEEE, pp. 2369–2374. 3, 19
- Hoare, Charles A. R. (1961). “Algorithm 65: Find.” In: *Communications of the ACM* 4.7, pp. 321–322. 39
- Hong, Sukjoon, Habeom Lee, Jinhwan Lee, Jinhyeong Kwon, Seungyong Han, Young D. Suh, Hyunmin Cho, Jaeho Shin, Junyeob Yeo, and Seung Hwan Ko (2015). “Highly stretchable and transparent metal nanowire heater for wearable electronics applications.” In: *Advanced materials* 27.32, pp. 4744–4751. 6

- Jaderberg, Max, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu (2016). “Reinforcement learning with unsupervised auxiliary tasks.” In: *arXiv preprint arXiv:1611.05397*. 48
- James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani (2013). *An introduction to statistical learning*. Vol. 112. Springer. 54
- Jiang, Ning, Strahinja Dosen, Klaus-Robert Muller, and Dario Farina (2012). “Myoelectric control of artificial limbs—is there a need to change focus?[In the spotlight].” In: *IEEE Signal Processing Magazine* 29.5, pp. 152–150. 49
- Johannes, Matthew S., John D. Bigelow, James M. Burck, Stuart D. Harshbarger, Matthew V. Kozlowski, and Thomas Van Doren (2011). “An overview of the developmental process for the modular prosthetic limb.” In: *Johns Hopkins APL Technical Digest* 30.3. JHU/APL, pp. 207–216. 2, 7
- Jost, Kristy, Daniel Stenger, Carlos R. Perez, John K. McDonough, Keryn Lian, Yury Gogotsi, and Genevieve Dion (2013). “Knitted and screen printed carbon-fiber supercapacitors for applications in wearable electronics.” In: *Energy & Environmental Science* 6.9, pp. 2698–2705. 6
- Kanerva, Pentti (1988). *Sparse distributed memory*. Cambridge: MIT Press. 17, 35
- Kober, Jens, J. Andrew Bagnell, and Jan Peters (2013). “Reinforcement learning in robotics: A survey.” In: *The International Journal of Robotics Research* 32.11, pp. 1238–1274. 1, 3
- Kopetz, Hermann (2011). “Internet of things.” In: *Real-time systems*. Springer, pp. 307–323. 5
- Kortuem, Gerd, Fahim Kawsar, Vasughi Sundramoorthy, and Daniel Fitton (2010). “Smart objects as building blocks for the internet of things.” In: *IEEE Internet Computing* 14.1, pp. 44–51. 1, 5
- Kou, Liang, Tieqi Huang, Bingna Zheng, Yi Han, Xiaoli Zhao, Karthikeyan Gopalsamy, Haiyan Sun, and Chao Gao (2014). “Coaxial wet-spun yarn supercapacitors for high-energy density and safe wearable electronics.” In: *Nature communications* 5. 6
- Krasoulis, Agamemnon, Iris Kyranou, Mustapha Suphi Erden, Kianoush Nazarpour, and Sethu Vijayakumar (2017). “Improved prosthetic hand control with concurrent use of myoelectric and inertial measurements.” In: *Journal of neuroengineering and rehabilitation* 14.1, p. 71. 49
- Kumar, Karthik, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava (2013). “A survey of computation offloading for mobile systems.” In: *Mobile Networks and Applications* 18.1, pp. 129–140. 2
- Kumar, Karthik and Yung-Hsiang Lu (2010). “Cloud computing for mobile users: Can offloading computation save energy?” In: *Computer* 43.4, pp. 51–56. 2
- Kyranou, Iris, Agamemnon Krasoulis, Mustafa Suphi Erden, Kianoush Nazarpour, and Sethu Vijayakumar (2016). “Real-time classification of multimodal sensory data for prosthetic hand control.” In: *Biomedical Robotics and Biomechatronics (BioRob), 2016 6th IEEE International Conference on*. IEEE, pp. 536–541. 49

- Lee, Gyu Myoung, Noel Crespi, Jun Kyun Choi, and Matthieu Boussard (2013). “Internet of things.” In: *Evolution of Telecommunication Services*. Springer, pp. 257–282. 5
- Lee, Jaehong, Hyukho Kwon, Jungmok Seo, Sera Shin, Ja Hoon Koo, Changhyun Pang, Seungbae Son, Jae Hyung Kim, Yong Hoon Jang, Dae Eun Kim, and Taeyoon Lee (2015). “Conductive Fiber-Based Ultrasensitive Textile Pressure Sensor for Wearable Electronics.” In: *Advanced materials* 27.15, pp. 2433–2439. 6
- Legenstein, Robert, Niko Wilbert, and Laurenz Wiskott (2010). “Reinforcement learning on slow features of high-dimensional input streams.” In: *PLoS Comput Biol* 6.8, e1000894. 35
- Lenz, Ian, Honglak Lee, and Ashutosh Saxena (2015). “Deep learning for detecting robotic grasps.” In: *The International Journal of Robotics Research* 34.4-5, pp. 705–724. 49
- Leonov, Vladimir and Ruud J. M. Vullers (2009). “Wearable electronics self-powered by using human body heat: The state of the art and the perspective.” In: *Journal of Renewable and Sustainable Energy* 1.6, p. 062701. 6
- Li, Wei, Fan Zhou, Waleed Meleis, and Kaushik Chowdhury (2017). “Dynamic Generalization Kanerva Coding in Reinforcement Learning for TCP Congestion Control Design.” In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, pp. 1598–1600. 3
- Lin, Mingfeng, Henry C. Lucas Jr, and Galit Shmueli (2013). “Research commentary—too big to fail: large samples and the p-value problem.” In: *Information Systems Research* 24.4, pp. 906–917. 57
- Lin, Stephen and Robert Wright (2010). “Evolutionary Tile Coding: An Automated State Abstraction Algorithm for Reinforcement Learning.” In: *Abstraction, reformulation, and approximation*. 3
- Littman, Michael L. and Richard S. Sutton (2002). “Predictive representations of state.” In: *Advances in neural information processing systems*, pp. 1555–1561. 48
- Lobos-Tsunekawa, Kenzo, David L. Leotta, and Javier Ruiz-del-Solar (2017). “Toward Real-Time Decentralized Reinforcement Learning using Finite Support Basis Functions.” In: *arXiv preprint arXiv:1706.06695*. 4
- Markovic, Marko, Strahinja Dosen, Dejan Popovic, Bernhard Graimann, and Dario Farina (2015). “Sensor fusion and computer vision for context-aware control of a multi degree-of-freedom prosthesis.” In: *Journal of neural engineering* 12.6, p. 066022. 49
- Miller, Thomas W. and Filson H. Glanz (1996). “UNH CMAC version 2.1: The University of New Hampshire implementation of the Cerebellar Model Arithmetic Computer - CMAC.” Robotics Laboratory Technical Report, University of New Hampshire, Durham, New Hampshire. 41
- Parker, Philip A., Kevin B. Englehart, and Bernard Hudgins (2006). “Myoelectric signal processing for control of powered limb prostheses.” In: *Journal*

- of *Electromyography and Kinesiology* 16.6, pp. 541–548. DOI: 10.1016/j.jelekin.2006.08.006. 32
- Pilarski, Patrick M., Michael R. Dawson, Thomas Degrís, Jason P. Carey, K. Ming Chan, Jacqueline S. Hebert, and Richard S. Sutton (2013). “Adaptive artificial limbs: A real-time approach to prediction and anticipation.” In: *IEEE Robotics and Automation Magazine* 20.1, pp. 53–64. DOI: 10.1109/MRA.2012.2229948. 34
- Pilarski, Patrick M., Michael R. Dawson, Thomas Degrís, Jason P. Carey, and Richard S. Sutton (2012). “Dynamic switching and real-time machine learning for improved human control of assistive biomedical robots.” In: *Biomedical Robotics and Biomechanics (BioRob), 2012 4th IEEE RAS & EMBS International Conference on*. IEEE, pp. 296–302. 16
- Pilarski, Patrick M., Michael R. Dawson, Thomas Degrís, Farbod Fahimi, Jason P. Carey, and Richard S. Sutton (2011). “Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning.” In: *Proceedings of the IEEE International Conference on Rehabilitation Robotics (ICORR)*. Zurich, Switzerland, pp. 134–140. 16, 34
- Pilarski, Patrick M., Ann L. Edwards, and K. Ming Chan (2015). “Novel Control Strategies for Arm Prostheses: A Partnership between Man and Machine.” In: *The Japanese Journal of Rehabilitation Medicine* 52.2, pp. 91–95. 48
- Pilarski, Patrick M. and Jacqueline S. Hebert (2017). *Upper and lower limb robotic prostheses Robotic Assistive Technologies: Principles and Practice*, Eds. P. Encarnação and A. M. Cook. Boca Raton, FL: CRC Press ISBN: 978-1-4987-4572-7, pp. 99–144. 6, 32, 33
- Pilarski, Patrick M., Richard S. Sutton, and Kory W. Mathewson (2015). “Prosthetic Devices as Goal-Seeking Agents.” In: *2nd Workshop on Present and Future of Non-Invasive Peripheral-Nervous-System Machine Interfaces, Singapore*. 7, 48
- Pilarski, Pilarski M., Travis B. Dick, and Richard S. Sutton (2013). “Real-time prediction learning for the simultaneous actuation of multiple prosthetic joints.” In: *Proceedings of the 13th IEEE International Conference on Rehabilitation Robotics (ICORR)*. Seattle, USA, pp. 1–8. 34
- Radmand, Ashkan, Erik Scheme, and Kevin Englehart (2014). “On the suitability of integrating accelerometry data with electromyography signals for resolving the effect of changes in limb position during dynamic limb movement.” In: *JPO: Journal of Prosthetics and Orthotics* 26.4, pp. 185–193. 49
- Ramsay, Judith, Alessandro Barbési, and Jenny Preece (1998). “A psychological investigation of long retrieval times on the World Wide Web.” In: *Interacting with computers* 10.1, pp. 77–86. 2
- Ratitch, Bohdana, Swaminathan Mahadevan, and Doina Precup (2004). “Sparse distributed memories in reinforcement learning: Case studies.” In: *Proc. of the Workshop on Learning and Planning in Markov Processes-Advances and Challenges*, pp. 85–90. 3, 38, 70

- Ratitch, Bohdana and Doina Precup (2004). “Sparse distributed memories for on-line value-based reinforcement learning.” In: *European Conference on Machine Learning*. Ed. by Springer Berlin Heidelberg, pp. 347–358. 3, 35, 38
- Richardson, Matt and Shawn Wallace (2012). *Getting started with raspberry PI*. ” O’Reilly Media, Inc.” 5
- Scheme, Erik and Englehart Englehart (2011a). “Electromyogram pattern recognition for control of powered upper-limb prostheses: State of the art and challenges for clinical use.” In: *Journal of Rehabilitation Research and Development* 48.6, pp. 643–660. 1, 32, 33
- Scheme, Erik and Kevin Englehart (2011b). “Electromyogram pattern recognition for control of powered upper-limb prostheses: State of the art and challenges for clinical use.” In: *Journal of rehabilitation research and development* 48.6, p. 643. 6, 7
- Sherstan, Craig, Joseph Modayil, and Patrick M. Pilarski (2015). “A collaborative approach to the simultaneous multi-joint control of a prosthetic Arm.” In: *Proceedings of the 14th IEEE/RAS-EMBS International Conference on Rehabilitation Robotics (ICORR)*. Singapore, pp. 13–18. 7, 34, 48
- Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Dominik G. Dieleman, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis (2016). “Mastering the game of Go with deep neural networks and tree search.” In: *Nature* 529.7587, pp. 484–489. 29, 72
- Singh, Satinder, Tommi Jaakkola, Michael L. Littman, and Csaba Szepesvári (2000). “Convergence results for single-step on-policy reinforcement-learning algorithms.” In: *Machine learning* 38.3, pp. 287–308. 19
- Sutton, Richard S. and Andrew G. Barto (1998). *Reinforcement learning: An introduction*. Cambridge: MIT Press. 13, 14, 18, 20, 29, 34, 36
- Sutton, Richard S., Hamid R. Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvari, and Eric Wiewiora (2009). “Fast gradient-descent methods for temporal-difference learning with linear function approximation.” In: *Proceedings of the 26th Annual International Conference on Machine Learning. ACM*, pp. 993–1000. 45
- Sutton, Richard S., Joseph Modayil, Michael Delp, Thomas Degris, Patrick M. Pilarski, Adam White, and Doina Precup (2011). “Horde: A Scalable Real-time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction.” In: *Proc. the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. Taipei, Taiwan, pp. 761–768. 14, 25, 48
- Sutton, Richard S, Csaba Szepesvári, Alborz Geramifard, and Michael P. Bowling (2012). “Dyna-style planning with linear function approximation and prioritized sweeping.” In: *arXiv preprint arXiv:1206.3285*. 2

- Tanner, Brian and Adam White (2009). “RL-Glue: Language-independent software for reinforcement-learning experiments.” In: *Journal of Machine Learning Research* 10.Sep, pp. 2133–2136. 18
- Tao, Xiaoming (2005). *Wearable electronics and photonics*. Elsevier. 6
- Tesauro, Gerald (1995). “Td-gammon: A self-teaching backgammon program.” In: *Applications of Neural Networks*. Springer, pp. 267–285. 18
- Travnik, Jaden B. and Patrick M. Pilarski (2017). “Representing high-dimensional data to intelligent prostheses and other wearable assistive robots: A first comparison of tile coding and selective Kanerva coding.” In: *Rehabilitation Robotics (ICORR), 2017 International Conference on*. IEEE, pp. 1443–1450. 14
- Van Seijen, Harm, A. Rupam Mahmood, Patrick M. Pilarski, Marlos C. Machado, and Richard S. Sutton (2016). “True online temporal-difference learning.” In: *Journal of Machine Learning Research* 17.14, pp. 1–40. 13
- Watkins, Christopher J. C. H. and Peter Dayan (1992). “Q-learning.” In: *Machine learning* 8.3-4, pp. 279–292. 20
- Watkins, Christopher John Cornish Hellaby (1989). “Learning from delayed rewards.” PhD thesis. King’s College, Cambridge. 12
- White, Adam (2015). “Developing a predictive approach to knowledge.” PhD thesis. PhD thesis, University of Alberta. 14
- Whiteson, Shimon, Matthew E. Taylor, and Peter Stone (2007). *Adaptive tile coding for value function approximation*. Computer Science Department, University of Texas at Austin. 3
- Wortmann, Felix and Kristina Flüchter (2015). “Internet of things.” In: *Business & Information Systems Engineering* 57.3, pp. 221–224. 5
- Wu, Cheng and Waleed Meleis (2009). “Function approximation using tile and Kanerva coding for multi-agent systems.” In: *Proceedings of adaptive learning agents workshop (ala) in aamas*. 3
- Xia, Feng, Laurence T. Yang, Lizhe Wang, and Alexey Vinel (2012). “Internet of things.” In: *International Journal of Communication Systems* 25.9, p. 1101. 5
- Yang, Shuang-Hua (2014). “Internet of things.” In: *Wireless Sensor Networks*. Springer, pp. 247–261. 5
- Zeng, Wei, Lin Shu, Qiao Li, Song Chen, Fei Wang, and Xiao-Ming Tao (2014). “Fiber-based wearable electronics: a review of materials, fabrication, devices, and applications.” In: *Advanced Materials* 26.31, pp. 5310–5336. 6