COGNITIVE ELECTRONIC WARFARE

AN ARTIFICIAL INTELLIGENCE APPROACH

KAREN ZITA HAIGH • JULIA ANDRUSENKO

Cognitive Electronic Warfare

An Artificial Intelligence Approach

Cognitive Electronic Warfare

An Artificial Intelligence Approach

Karen Zita Haigh

Julia Andrusenko

ARTECH
HOUSE

BOSTON | LONDON
artechhouse.com

Library of Congress Cataloging-in-Publication Data
A catalog record for this book is available from the U.S. Library of Congress.

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library.

Cover design by Charlene Stevens

ISBN 13: 978-1-63081-811-1

10 9 8 7 6 5 4 3 2 1

Contents

Foreword

Advances in electronic warfare (EW) have determined the winners of wars. The research, development, and adoption of these technical advances shapes the tactics, operational maneuvers, and strategies employed by commanders at all levels in the battle space. Advanced research and development efforts contribute substantially to deterrence. The Department of Defense in the United States invests approximately $7B annually in EW; I led the development and acquisition of these systems in the Pentagon and continue to advance these systems as a chief technology officer in our national security innovation base.

Cognitive EW is one of the critical advances that will determine the outcomes of future battles. An EW researcher from World War II would recognize many of the systems operationally fielded today. The application of artificial intelligence (AI) to make EW systems cognitive is a critical advance to allow our systems to adapt and to learn during a mission. In a digital world of software-defined capability, EW systems must respond to previously unknown signals. In the densely connected battle space of today, the so-called internet of military things, feedback can be known, or at least estimated, continuously during a mission. Through aggregated sensor feeds and understanding, EW systems are capable of adapting based on this real-time feedback. With appropriate automation, this feedback and learning can occur faster than humans can reason on data. This adaptation during an individual mission or engagement can allow our soldiers and our airmen to successfully complete their missions, or in some cases survive an engagement.

While it's easy to state that an EW system will learn from its surroundings in real time, it's quixotic to assume that just any system is effective, and robust, in performing its function. Society, including its military elements, is excited at the future AI will enable. To realize this future, we need experts like Karen and Julia to link the theory to the application. While it's easy to showcase a demo, military systems require robustness in their operations and will undergo critical testing and evaluation.

The application of AI into our lives will change how we perform functions each day. Military applications of cognitive technology require security as well to protect their functionality. Machine learning and adaptivity begin to change this for the military. While there are diverse interesting philosophical questions around protecting algorithms, specific implementations, training data, and real-time learning, the academic foundation is quite robust and provides valuable insights for practitioners in the field. Given the rapid rate of innovation and advancements in AI, military systems must leverage these advances at the speed of relevance.

Karen and Julia do an excellent job in this book at providing the framework that will enable developers to include these recent advances into their capabilities. A core understanding of AI provides a foundation, followed by a detailed discussion of objective functions for cognitive EW. Each EW function is unique, and the nuances are described in necessary depth. Every acquisition professional knows that sustaining weapon systems is where the costs are truly borne. In this case, the management of data, building the right architectures, and testing these new systems is covered. Last, Karen and Julia provide a gateway to these new system functions. The balance of breadth and depth ensures alignment and success to meet the lofty goal of helping nations deter and, when needed, win wars.

Dr. William G. Conley
Chief Technology Officer, Mercury Systems
Former director of EW the Office of the Secretary of Defense
July 2021

Preface

I often hear that cognitive EW is too big a landscape and that people don't know where to start. I always say: "AI is like Math: One day it will be everywhere." Everybody, no matter their role or their age, should know enough about AI to know where it applies to the problems in their daily lives. AI is not just "autonomous vehicles" or "Terminator;" AI is face recognition on a camera, purchase recommendations when shopping, and predictions about hurricane paths. AI belongs in EW systems, just as math and physics belong in EW systems. This book aims at lighting the path, and igniting a passion for solving EW problems that can't be solved using only traditional methods.

Julia and I wrote the book for radio frequency (RF) people—experts in EW, cognitive radio, and/or cognitive radar—who want to learn more about how and where to use AI. Our goal is to help triage and guide EW system designers in choosing AI and machine learning (ML) solutions. ML is a key component, but AI is much more than just ML.

If it seems like we flip-flop between communications and radar, or electronic attack and electronic protect, we do. From an AI standpoint, these problems truly are interchangeable. We use the same AI techniques in all four of those areas. There are some important differences, such as EW battle damage assessment (BDA), but AI can bring these ideas together, solve common problems, and move from coexistence to codesign.

We use a running example throughout the book: the BBN Strategy Optimizer (BBN SO). It is a communications electronic protect effort that learns how to handle novel environments using in-mission learning at mission-relevant timescales, and then performs hard real-time optimization and decision-making to choose mitigation strategies. Example 7.1 presents the essential concept, and many of the discussion points use examples derived from this work.

We'd like to thank the people who directly contributed to the book in some concrete way: Aaron Walker, Chris Geib, Matt Markel, Bill Conley, Pete Reese, Feng Ouyang, Shaun McQuaid, and the anonymous reviewer from Artech House.

The first two people who started me thinking about the AI-for-RF problem are Chris Ramming who presented the initial problem, and Jim Freebersyser who funded the first effort from his Honeywell internal research budget. Without these two individuals, I would never have developed the interest or momentum to see it through. Greg Troxel and John Tranquilli patiently helped educate me—that weird AI person—teaching me how to spell RF and think about what makes things hard in this space.

Thanks also to the many others who have helped mature my thinking and develop AI-RF systems that actually work in the field. These include Louis Bouchard, Michael Cook, Amber Dolan, Dianne Egnor, Regina Hain, Brian Hood, David Landon, Li Lin, Scott Loos, Allan Mackay, Hala Mostafa, Andrew Nelson, Jason Redi, Vikram Sardana, Choon Yik Tang, Chris Vander Valk, Srivatsan Varadarajan, and Fusun Yaman. I'd also like to thank the program managers and fellow performers on my DARPA, AFRL, ONR, and transition contracts.

Julia has been an amazing partner on this effort, providing different perspectives, arguing with me when I needed it, and cheering me on when things weren't going well. I'd like to thank my Ph.D. advisor Manuela Veloso, who taught me to stick with an idea, even when the rest of society isn't ready for it yet. Finally, thanks to my family, who understand most of the nouns in this book but not the phrases, and still support me.

—Karen Haigh

Karen and I first met in November 2019 at the IEEE Military Communications Conference in Norfolk, VA. It is then when she asked for my help with this book's write-up. I believe my first answer was "no," then, "let me think about it." Karen, thanks for your persistence in bringing me on this wild journey with you. I echo Karen in expressing my immense gratitude to all of the people that helped us tackle this difficult, multifaceted problem of cognitive EW. Last, from my side, I wanted to thank my brilliant Johns Hopkins University Applied Physics Laboratory colleagues for their continual support of all of my endeavors. Thank you all.

—Julia Andrusenko

1

Introduction to Cognitive Electronic Warfare

The challenges of modern EW are beyond the ability of traditional approaches to solve. Incorporating AI techniques into EW systems is the only way to manage the complexity of this problem domain and its rapid timescales. This book describes how AI techniques can help address the challenges of modern EW. We expect readers to be familiar with at least one of EW, cognitive radio, or cognitive radar domains, and will focus instead on AI techniques and their associated challenges and trade-offs. In the future, AI will be part of every EW system, recording and analyzing the system's previous performance and then adapting behavior according to the current situation. AI—not just ML—is the heart of future cognitive EW solutions.

Cognitive radio concepts have existed since at least 1999, when Joe Mitola III [1] introduced the term, while the idea of cognitive radar has been around since at least 2006 [2]. Fielded examples, however, are few and far between and limited in their cognitive capabilities. A significant reason for this gap is due to the sense that building a cognitive system is a huge undertaking, exacerbated by the misconception that it is impossible to evaluate a cognitive system.

This book aims to address this gap: it introduces RF practitioners to relevant AI techniques for EW: situation-assessment (SA), decision-making (DM), and ML. It covers SA techniques for electronic support (ES), including characterization, classification, anomaly detection, causal reasoning, and intent recognition. The book discusses DM techniques for electronic protect (EP), electronic attack (EA), and electronic battle management (EBM), including planning, optimization, and scheduling, and how to manage the temporal trade-offs and distributed nature of the problem. It describes where to incorporate ML to improve both SA and DM. The book also examines a significant area of interest for EW system designers: real-time in-mission learning to recognize novel environments and respond to surprises.

To address the barrier-to-adoption created by the lack of understanding of how to evaluate a cognitive system, the book describes how to manage data and models to ensure quality and presents a learning-assurance process for validating the results and building trusted systems.

Finally, this book provides a few recommendations on how to get started with building an actual cognitive EW system.

Creating a cognitive EW system is not a huge "all-or-nothing" hurdle.

From an AI standpoint, EP and EA differ only in their objectives: EP defines objectives with respect to oneself, while EA defines objectives with respect to the adversary. Likewise, AI is agnostic to whether the solutions apply to radar or communications problems. This book therefore treats EP/EA and radar/communications equally, highlighting the few places where the problem domain impacts the choice of data or algorithm.

The AI techniques presented here apply to other related disciplines such as cybersecurity; information warfare; position, navigation, and timing (PNT); and intelligence, reconnaissance, and surveillance (ISR). We do not directly address related areas.

1.1 What Makes a Cognitive System?

AI, the field of computer science that attempts to mimic human behaviors, was founded as an academic discipline in 1956 at the Dartmouth Conference [3] and draws from a wide variety of fields including cognitive psychology, mathematics, computer science, systems engineering, and linguistics.

AI comprises many subfields, including knowledge management, planning, ML, natural language processing (NLP), and autonomy.

A *cognitive system,* or *intelligent agent,* perceives its environment and takes actions to achieve its goals, as illustrated in Figure 1.1. It reasons and understands at a higher level, dealing with symbolic and conceptual information, to make accurate decisions in complex situations. Cognitive systems are aware of context, handle uncertainty, and make judgments autonomously. They are iterative and interactive and learn from their experiences. Figure 1.2 illustrates the cognition loop with its three concepts:

- *SA* is the understanding of environmental elements and events with respect to time or space, the comprehension of their meaning, and the projection of their future status. (Situation awareness is result of a SA module.) It is a critical foundation for successful DM. Key steps are to *collect* the raw data,

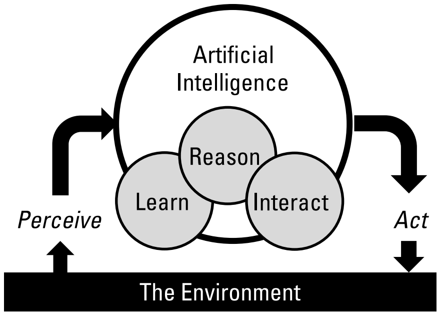


Figure 1.1 A cognitive system perceives the environment, reasons about the situation, and acts to accomplish goals. It learns from the interaction with the environment.

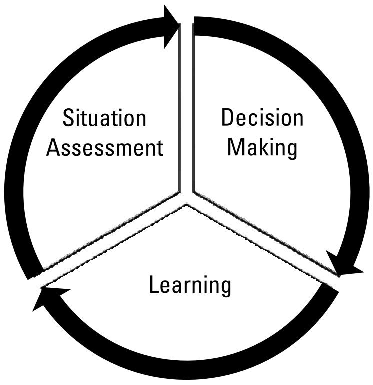


Figure 1.2 A cognitive system iteratively assesses the situation, plans and decides its actions, and learns from its experience.

validate the observations, *fuse* the data into higher-level concepts, *analyze the impact* of the situation, and *infer the intent* of users (both friendly and adversarial).

- *Execution monitoring* is the activity within SA that focuses on evaluating how well-planned actions succeeded. Key challenges for an SA module include diversity in the data, latency of when the data is received, and how observable the environment is.
- *DM* involves setting goals and determining feasible methods of achieving them. The cognitive system must prioritize goals, understand the benefits and costs of different actions, and resolve conflicts. Key challenges include managing resources, multiple actors, and nondeterministic actions, especially in problem-solving domains where the environment is constantly changing.

- *Learning* extracts information from prior experience to improve future performance. ML techniques may extract rules about how to interpret observations or behave, or they may build functions that approximate the performance of the data. Key challenges include data heterogeneity, missing data, and managing bias.

Figure 1.3 lays out these three functions to highlight increasing levels of cognition. A simple, traditional EW system uses the ES to identify known signals, and chooses a response based on a library lookup. Each AI capability increases the overall level of cognition. This layout also provides a means to evaluate and compare EW system capability. Horne et al. present a cognition mapping that breaks down the DM components by memory, forethought, and algorithmic sophistication [4].

The cognition loop is similar to the *observe, orient, decide, and act (OODA)* loop commonly used in military environments [5]. The key difference between the OODA loop and the cognition loop is that OODA describes a process; cognition happens in the humans following that process. As one example, SA performed by a cognitive system feeds the situation awareness of the human operators.

1.2 A Brief Introduction to EW

Electromagnetic (EM) spectrum operations (EMSO) comprise all coordinated military actions to exploit, attack, protect, and manage the electromagnetic envi-

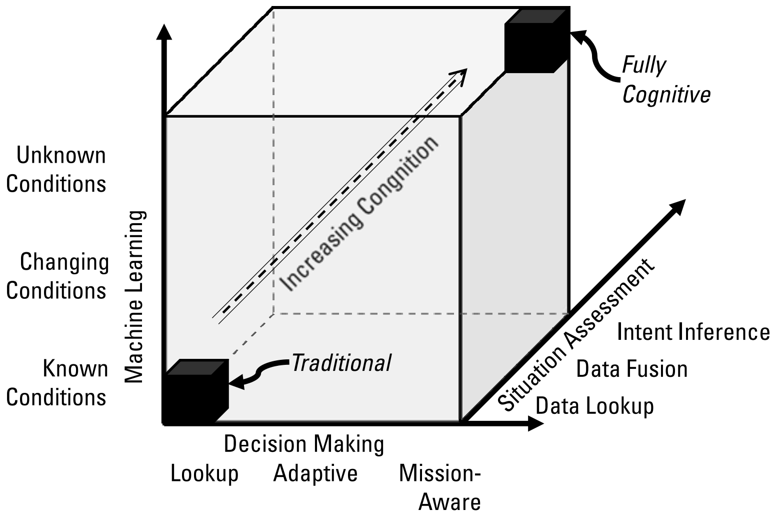


Figure 1.3 AI functions contribute to increasing levels of cognitive function, according to the concepts laid out in Figure 1.2.

ronment to achieve a commander's objectives. EM EW focuses on how to control the spectrum or attack an enemy using the EM spectrum, including infrared, optical, and microwave [6, 7]. Many of the AI techniques in this book apply to cyber and other multidomain aspects (e.g., information warfare [8]), but we do not specifically address these adjacent domains.

> Electronic warfare (EW) is any action involving the use of the electromagnetic spectrum (EM spectrum) or directed energy to control the spectrum, attack an enemy, or impede enemy assaults. The purpose of electronic warfare is to deny the opponent the advantage of, and ensure friendly unimpeded access to, the EM spectrum. EW can be applied from air, sea, land, and/or space by manned and unmanned systems, and can target humans, communication, radar, or other assets (military and civilian).
>
> —*Chairman of the Joint Chiefs of Staff, 2012 [9]*
> *Joint Publication 3-13.1: Electronic Warfare*

EW comprises the following core concepts:

- *ES* understands the spectrum—who is using it and how, when, and where. ES needs to detect, intercept, identify, and locate EMS energy, understand how it is being used, and determine whether there are any identifiable patterns that can be exploited.
- *EP* involves actions taken to protect the friendly nodes from any undesirable effects due to changes in the spectrum such as jamming or noise. This activity chooses strategies or countermeasures to maintain communications or radar performance. EP conversations revolve around antijamming and radar countermeasure techniques; in this book we use EP equally to discuss techniques used to protect communication and radar systems. Strategies may include frequency agility, waveform design, antenna direction, and signal processing to reduce jamming.
- *EA* denies the adversary access to its own RF spectrum. EA uses offensive EM energy to degrade or deny the adversary's access to the spectrum, or to deceive the adversary by conveying misleading information.

  Deny, degrade, disrupt, deceive, destroy

- *EBM* oversees all aspects of EMSO to increase mission effectiveness, including managing changing mission priorities, coordinating effects, and collaborating with other elements of mission command. A key aspect is to interact with and support the EW officer.

- *EW BDA* assesses the effectiveness of the EA and provides feedback that allows the operator or system to create more effective attacks. EW BDA modules feed into, and are often part of, the ES module.
- *EW reprogramming* changes the self-defense systems, offensive weapons systems, and intelligence-collection systems. EW reprogramming activities fall into three major categories—tactics, software, and hardware—and provide commanders with a timely capability to respond to changes in adversary threat systems, correct system deficiencies, and tailor equipment to meet unique theater or mission requirements.

The *EW Handbook* [10] is a good reference for many EW equations and concepts.

From an AI standpoint, EP and EA can be treated the same way; the only difference is that EP defines objectives with respect to oneself, while EA defines objectives with respect to the adversary (and is thus harder to measure). The same applies to communications and radar systems, in that the available actions are the same; communications systems have more challenging distributed DM requirements, with more latency, more coordination, and a more nuanced set of performance metrics.

Table 1.1 shows how these EW concepts map to the common AI terms introduced in Section 1.1. Other mappings exist, including Haykin's *Perception-Action Cycle* [11] and Mitola's *Observe-Orient-Plan-Decide-ActLearn (OOPDAL)* model [1].

Figure 1.4 illustrates how different AI concepts and techniques might appear in a traditional EW functional diagram. Note that the EW kill-chain of *find-fix-track-target-attack-assess (FFTTAA)* is quite similar to the OODA and cognition loops.

1.3 EW Domain Challenges Viewed from an AI Perspective

Like any other intelligent system, cognitive EW must overcome challenges associated with each AI concept or stage of the cognition loop. The domain is challeng-

Table 1.1
EW Activities and AI Counterparts

AI Term	**EW Term**
Situation assessment	Electronic support
Decision making	Electronic protect and electronic attack Electronic battle management
Execution monitoring	Electronic warfare battle damage assessment
Learning	Electronic warfare reprogramming (of data and software)

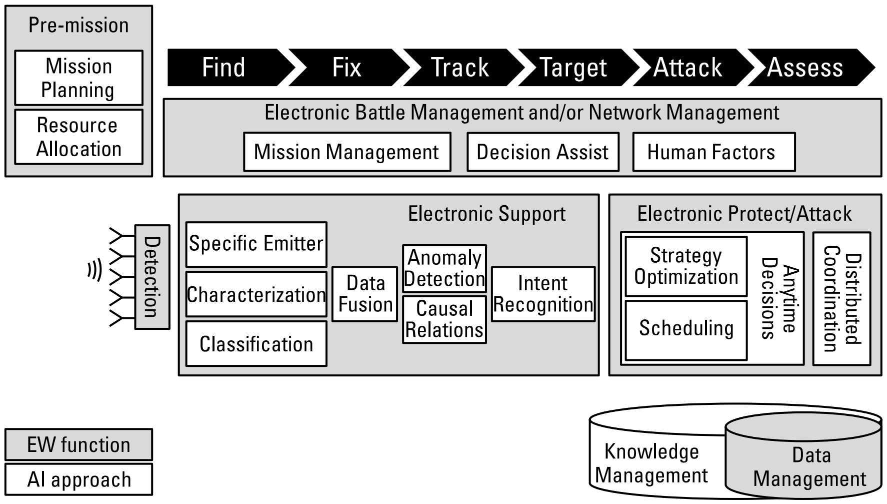


Figure 1.4 AI situation assessment, decision making, and learning capabilities are relevant for all EW functions.

ing to understand, and the decision space is large and complex. User requirements add an additional layer of complexity to the system. This book addresses two of the challenges of the U.S. Department of Defense's EW Community of Interest [12]: (1) cognitive/adaptive, and (2) distributed/coordinated.

1.3.1 Situation-Assessment for Electronic Support and Electronic Warfare Battle Damage Assessment

The dynamics and complexity of the RF/EW domain create a number of challenges for SA for ES and EW BDA:

- *Dynamic:* Nothing in a distributed cognitive EW system is static. Most assessments of the current situation are valid only fleetingly and may have even expired before DM activity begins.
- *Ambiguous:* Some observations may have multiple root causes or imply multiple conclusions. Detection and understanding of a change in situation is not always simple. For example, how does the system automatically tell the difference between short-term fade versus entering a building?
- *Partially observable:* Many factors that impact the EMS cannot be observed. Few platforms, for example, incorporate a "fog" sensor. EW BDA is notoriously difficult to accomplish accurately. Some nodes may be able to sense an event, while other nodes cannot.
- *Complex interactions:* Many control parameters have poorly understood interactions with each other and with system performance. While specific

pair-wise interactions can often be identified, such as that increased power reduces battery life, generally these pair-wise interactions are studied in laboratory environments, and the conclusions do not transfer directly to a fielded system. System calibration in the field addresses some pair-wise interactions, but it is typically limited to the most significant performance parameters because it is time-consuming and expensive to perform. Interactions across many parameters are rarely studied, even in the lab.

- *Complex temporal feedback loops:* Within a node, certain activities occur at very rapid speeds requiring a very tight feedback loop to support cognitive control. Other activities occur on a longer timescale that requires analyzing a wider range of factors over broader windows of time. Between nodes, there is yet a longer feedback loop between decisions and their effects. The variety of temporal loops and their dramatic feedback latency differences means that correlating cause and effect of actions is particularly challenging.

These characteristics combine to create the significant challenge of a *lack of training data.* In a complex EW domain, it is impossible to collect sufficient examples of how the system behaves. While it may be possible to collect many examples of WiFi performance, where nodes are somewhat mobile, this is not true for highly mobile settings, multiple weather patterns, complex missions, and novel waveforms. Synthetic data can represent some of this variation (e.g., channel effects), but this data will be neither realistic enough or comprehensive enough to cover all situations. Chapter 8 addresses the training data challenge.

This challenge is exacerbated given the adversarial setting: A true representation of adversary tactics and capabilities is likely not available for day 0 operations, and becomes increasingly true for day 1 to *N* as the situation in a conflict changes.

The EW system must be able to learn from very small numbers of novel examples, even during a mission.

1.3.2 Decision-Making for Electronic Attack, Electronic Protect, and Electronic Battle Management

Many EW domain characteristics also create challenges for DM for EA, EP, and EBM. The operating environment and available equipment impose constraints to which the decision maker must conform, including:

- *Dynamic:* Military missions change, user requirements change, platforms join or leave the ensemble, hardware fails, and mobility causes continuous fluctuations in communications connectivity. Each change increases the complexity of the DM activity.

- *Resource constrained:* Nodes typically have tight constraints on size, weight, and power (SWaP). Power management is a critical requirement for remote operations. A given node may have the ability to accomplish a given EW task but will exhaust its resources when doing so. One strategy to reduce the resource use on a single node is to distribute the task across multiple nodes, but communications is *also* a resource that the system must manage.
- *Distributed:* As a result of the limited communication and frequent disconnections, nodes must intentionally trade off the benefit of rapid local DM against the advantages that can be gained by coordinating across nodes. A stand-alone EW system or platform has significantly lower latencies than an internetworked EW system.
- *Diverse:* Nodes in a distributed EW environment may have a wide variety of capabilities, from small hand-held radios, to large platforms with satellite communications (SATCOM), or aircraft with modern radar systems. Nodes may have different RF antenna components and supporting computational infrastructure such as memory and compute. Diversity also exists in the operation of similar equipment, sensors, produced by different manufacturers or different revisions. This heterogeneity requires different solutions on different nodes, causing additional diversity in the software layers.
- *Massive scale:* There are many observable and controllable parameters (possibly continuous-valued) to configure *per node.* No current system exposes all of theoretically available control parameters. The maximum number of strategies that each node can take, per time step, is $\Pi_{\forall c} v_c$, where v_c is the number of possible values for the control parameter c. Even if all $\overline{c_n}$ control parameters are simply binary on/off, then there are $2^{\overline{c_n}}$ strategies per node, per time step. When even one controllable is continuous-valued, there is an infinite number of configurations.
- *Macroscale impacts:* Elements essentially unrelated to the direct EW engagement can affect outcomes. For example, red platform maneuvers that exceed their antenna coverage, or lack of or incidental illumination during red threat engagement with other platforms both impact mission success.

These challenges create a distributed, heterogeneous, low-communication, partially observable, high-latency, exponential optimization problem. Cognitive control in the general case is therefore never simple: The level at which symptoms appear may not be the level at which changes to the node configuration must be made; symptoms may be ambiguous at one level or at a given time and require more context; changes in one module may impact other modules and may cause new issues; and the timing of changes may be critical. In practice, however, the challenge is not as great as it seems. First, from an engineering perspective, the "true" optimal is not required, and in practice there may be many "good enough" solutions (Section 5.3). Second, physics and progression of the engagement both constrain the problem.

In other domains, AI techniques have addressed the full richness of most of these challenges. In the cognitive radio (CR) and EW domains, AI techniques are just beginning to scratch the surface. AI techniques have addressed many of these challenges in physically embodied systems that have some similar characteristics to the EW domain (e.g., robotics). We need to bring these techniques into EMS operations and address them in depth.

The one notable exception where AI has only just begun to explore is that of automatic heterogeneous intercommunication. There has historically been a very strong norm in the networking community that all nodes must be designed and (preferably statically) configured to interoperate [13], typical ad hoc networks built from a group of homogenous nodes. CR networks break this assumption: Each node can have an independent cognitive controller, and thus network nodes *may be heterogeneous, and may fall into in noninteroperable configurations.*[1] Meanwhile traditional AI has always assumed that communication is "safe," negotiating and coordinating only the application-level tasks [14–16]; moreover, they also generally require very high communications overhead. These assumptions—homogeneity and safe communications—fall squarely into the set of eight distributed computing fallacies (Figure 1.5). ADROIT [18], by giving each node its own learning system, represented a *radical* departure from the traditional networking stance that requires homogeneous configurations. ADROIT was the first system to demonstrate an effective *heterogeneous* mobile ad hoc network (MANET) and, moreover, used ML to configure the nodes. Section 5.4 discusses approaches to addressing the distributed coordination challenge.

1.3.3 User Requirements

Finally, user requirements, described as follows, add another layer of complexity to the cognitive EW system:

- *Complex access policies:* Due to the heterogeneous nature of the data the nodes, and the missions, access policies may restrict the set of nodes that are permitted to hold or transmit specific data.

1. The alternative is to have one cognitive controller for several nodes; while coordination issues are reduced, communication overhead and latency increases dramatically and intelligent control is vulnerable to network partitions.

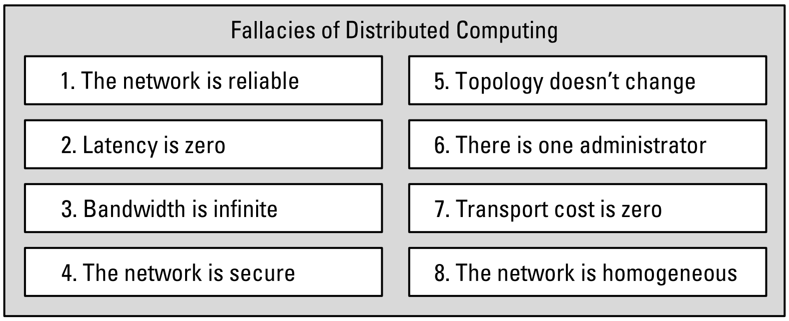

Fallacies of Distributed Computing
1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

Figure 1.5 When designing a distributed cognitive EW system, one must not fall victim to the classical eight fallacies of distributed computing defined in 1994 by Peter Deutsch and James Gosling [17]. These fallacies are more acute in EW.

- *Complex multiobjective performance requirements:* Multiple users have interacting requirements and policies, thus requiring a complex multiobjective function that captures mission and environmental standpoints [19]. It is rarely possible to optimize all objectives simultaneously. Instead, the system must choose a particular operating point that balances the tradeoffs. Examples include resource use, joint optimization of ES/EP/EA, joint optimization of comms and radar, and accuracy such as probability of detection (Pd) versus probability of false alarms (Pfa).
- *Constrained operations:* All cognitive EW system designers desire real-time, in-mission ML. The concern from a user perspective is that the system may learn inappropriate behaviors. Requirements must simultaneously constrain emergent behavior, and yet address novel, unexpected situations.

Ultimately, these factors lead to how much the human stakeholders *trust* the information gathering, SA, and DM of the AI system. Stakeholders include the acquisitions community, who want "verifiable" and "guaranteed" performance, as well as the individuals—electronic warfare officers (EWOs), air crew, sailors, and soldiers—whose lives depend on pragmatic actions. Section 5.1.1 discusses multiobjective optimization. Section 6.3 discusses the human-machine team, and Chapter 10 presents mechanisms for verification and validation of an AI-based system.

1.3.4 Connection between Cognitive Radio and EW Systems

CR networks accomplish EP objectives of cognitive EW systems in RF communications. Table 1.2 outlines some of the potential and well-known benefits of CR, which are directly applicable to cognitive EW.

Table 1.2
The Benefits of Cognitive Radio Systems and How They Transfer to Cognitive EW

Cognitive Radio Benefits		**Cognitive EW Benefits**
RF communications, seamless to the user	→	EMS operations must also be seamless to the user. Moreover, networked radar EW systems require underlying communications.
Self-regulating radio networks	→	EW systems must also be self-configuring and self-regulating, particularly given the rapid requirements of the EW kill chain.
Better performance in terms of cost, speed, power consumption, availability, timeliness, and resource use	→	Better performance in terms of cost, speed, power consumption, availability, timeliness, and resource use. EA activity, in communications or in radar, is also measured by effectiveness (EW BDA).

CR concepts were developed to address RF spectrum usage efficiency [20–22]. CR technology promises to become a transformative force within spectrum management. All EMS operations can leverage these novel spectrum management approaches and opportunistic spectrum access. Furthermore, opportunistic spectrum access implies that CR engines have achieved a certain level of spectrum situational awareness, which is an important aspect of an intelligent EW system as well. CR networks are also looking at heterogeneous radio frameworks' management, a concept also relevant for multiagent, heterogeneous EW systems [23–26]. 5G, the fifth-generation cellular network, also has similarities to the EW environment. Callout 11.1 outlines use cases and some cognitive approaches.

Much of the published CR work deals with multiaccess issues, not spectrum use in an adversarial environment. Moreover, CR research usually assumes cooperative entities with similar goals, while EW scenarios are always adversarial. Chapter 5 describes approaches to optimizing in the adversarial setting.

1.3.5 EW System Design Questions

Cognitive EW system design must provide SA, DM, and learning capabilities. System requirements drive a set of decisions about which components to incorporate, and what questions to ask during system design [20, 27–29]:

- *DM:* Should decisions be centralized or distributed? What DM algorithms should be used? How do we define optimization functions? Can we leverage the physics of the domain, and progression of the engagement to reduce the state space? Chapter 5 describes DM for EP and EA, while Chapter 6 describes DM for EBM.
- *Learning:* How do we define appropriate learning tasks? Should learning be supervised or unsupervised or semisupervised? What measurements

(features) should form the basis of the learning task? How do we handle the combinatorial explosion of the data that captures all past observations and decisions? Chapter 4 describes learning tasks, and Chapter 8 describes data-management approaches.

- *Sensing:* How do we assess the accuracy of spectrum sensing capabilities? Can we improve sensor accuracy? How do we leverage cooperative sensing mechanisms across spatially diverse nodes without incurring untenable latency? Can we leverage redundancy in data feeds to compensate or correct for failures? How do we address other topics such as long-range RF sensing, multifunctional RF sensing, and passive RF sensing? Section 4.3 discusses some of the data fusion concepts that AI can help with, while Section 8.1.1 discusses the need to record not only the data, but also its uncertainty and origin.
- *Security:* When do we need to encrypt data? How do we ensure policies to avoid security violations, particularly given the highly dynamic environment and missions? How do we protect data and models without jeopardizing performance, particularly accuracy and latency? Can we ensure that even if a model is compromised, it cannot be reverse-engineered? Section 8.3.5 talks about some security issues.
- *Software architectures:* How do we address global optimization of an EW system, both globally across the network of platforms, and internally within each node across each module? Section 9.1 describes software architectures in more detail.
- *Hardware design:* How do we design RF systems that are amenable to cognitive control? Can we effectively design multifunction hardware systems that can handle cognitive configurations? What memory and compute capabilities can we incorporate within the SWaP constraints of the platform? Because not all data can be recorded, what data compaction techniques effectively minimize data loss? Section 9.3 outlines some of the hardware considerations.

1.4 Choices: AI or Traditional?

AI, despite the hype in the media, is not applicable to every problem. AI solutions are less brittle, more transferrable to different problems, and more scalable than traditional approaches.

One does not need large computers to achieve impressive results. (That said, most AI practitioners would choose the largest compute and memory possible within the SWaP constraints.)

AI-based approaches scale to available hardware, including smaller hard real-time embedded systems.

For example, the cognitive EP system by Haigh et al. [30] learns how to maintain communications in a variety of novel interference conditions. The Strategy Optimizer (SO) uses the ML technique *support vector regression machines* [31], which enables it to achieve real-time in-mission learning on an ARMv7 processor in under a second. More details appear in Example 7.1.

The question is therefore how to choose between a traditional approach or an AI-based approach. There are times that a direct computation will be faster and more efficient. Broadly speaking, if an available model is accurate, then the traditional approach is likely to be more efficient than an AI-based approach. For example, if physics captures all relevant interactions, and empirical data doesn't change the function significantly, then the physics model is a better choice. (See the *EW Handbook* [10] for examples.)

AI approaches and traditional approaches should coexist on platforms, performing tasks appropriate for their strengths. Depending on measurements, conditions, and uncertainty, either could be the final arbiter for given tasks. Likewise, the two approaches could operate in parallel, and the results combined to generate the final outcome.

We can evaluate a problem domain according to any of the characteristics in Section 1.3. The most informative of these are how *distributed*, *complex*, or *dynamic* the problem is. Figure 1.6 illustrates these axes, indicating attributes that

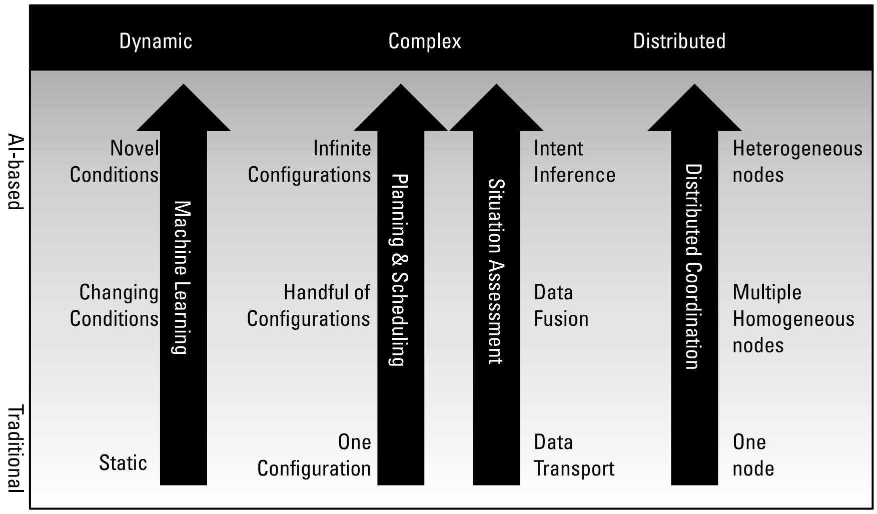


Figure 1.6 Domain characteristics determine whether AI is useful or necessary. Figure 1.3 places these AI functions to show levels of cognition.

determine when a more traditional approach is likely to be sufficient, and when an AI-based approach becomes necessary.

When the problem domain is highly *distributed,* distributed coordination (or multiagent techniques) become increasingly important. If a single node can accomplish the task, the system does not need to merge potentially disparate observations of the same object, negotiate task assignments, or coordinate actions. With a team of multiple homogeneous nodes, each node can make assumptions about its teammates, minimizing communications and facilitating coordination. Heterogeneous nodes must allocate resources and tasks across the nodes and dynamically update those allocations during a mission while minimizing communications overhead. Most AI-based distributed coordination approaches generate more resilient plans than conventional control systems. Networked EW systems have loosely coupled task coordination (e.g., deciding which nodes should do stand-in jamming or becoming a communications relay), and they also have tightly coupled task coordination (e.g., performing distributed coherent transmissions).

When the problem is highly *complex,* SA techniques become relevant for data understanding. When data simply needs to be moved from one node to another, no knowledge is extracted from the data. When a traditional model can extract useful information efficiently and accurately, then AI is unlikely to improve results. Data fusion brings together disparate data sources, possibly different types of data collected, or the same type of data on multiple nodes. The more disparate the types of data, and the more varied the temporal characteristics, the more that data fusion concepts need to be employed. Plan recognition and intent inference techniques enable the system to understand the intent of actors in the domain, whether they are users or adversaries. To maintain system performance, EP systems need to recognize the intent of the users. EA systems are more effective when they understand the intent of the adversary.

Planning and scheduling techniques address DM as domain complexity increases. If the system does exactly one thing (i.e., has one configuration), then traditional control approaches are appropriate. Simple rules can manage a handful of possible configurations. Planning, scheduling, and optimization approaches become necessary when there are many possible configurations. In most EW systems, the configuration parameters can be combined in many ways, yielding a possibly infinite number of configurations. At the same time, the EW system may encounter a possibly infinite number of scenarios. AI-based DM is critical.

The third key domain characteristic is *dynamism.* When the domain is static, a preconfigured system is sufficient for the task. When the domain is dynamic, and the system encounters changing conditions, an adaptive approach becomes necessary, for example, selecting EP countermeasures appropriate for the observed jammers. These responses can be designed a priori by hand or by a (non-real-time) ML approach, and then uploaded to the system via EW reprogramming.

When the system will encounter novel conditions, such as novel transmitters, then a real-time in-mission ML approach is crucial.

Each of these technical areas (distributed coordination, SA, DM, and ML) support a vibrant research community in AI, including dedicated technical conferences and publications. In each case, the research community is developing multiple approaches; the cognitive EW community can choose approaches appropriate for available hardware. Each of the chapters in this book will discuss relevant approaches, and highlight the trade space that system architects should consider.

The key lesson is to not select the solution (i.e., cool AI technology) without fully understanding the problem.

What desired outcome will the AI enable? What data exists upon which to base a ML solution? Do the characteristics of the problem warrant an AI-based solution?

1.5 Reader's Guide

The chapters in this book walk through the concepts required for building a fully cognitive EW system, organized by the components in Figure 1.4.

- Chapter 2 describes the objective function that drives DM.
- Chapter 3 presents a short primer[2] on ML, including a discussion of algorithmic trade-offs.
- Chapter 4 explains how to assess an ES situation.
- Chapter 5 describes how to choose a strategy for EP and EA in time-constrained and distributed settings.
- Chapter 6 presents EBM and the human interface, dealing with planning, including resource management, uncertainty, and adversaries.
- Chapter 7 examines online replanning and learning, including EW BDA, which maps expected outcomes to observed outcomes.
- Chapter 8 presents data management processes and practice.
- Chapter 9 covers software and hardware architecture considerations.
- Chapter 10 presents evaluation: how to test and what to test.

2. The word *primer* is pronounced *primmer* using the *i* from *sit* when it covers the basic elements of a subject. It is pronounced *pr-eye-mer* using the *i* from *five* when it is an underlayer of paint.

- Chapter 11 condenses all the lessons learned into recommendations on how to get started building a cognitive EW system.

1.6 Conclusion

Building a cognitive EW system requires understanding what and where AI can help: SA for ES and understanding the RF environment, DM for choosing EP/EA/EBM actions, and ML for continuous improvement.

This high-tempo complex environment is well-suited to the application of AI, but there are challenges to developing fieldable cognitive EW systems that can operate in that environment. A fully cognitive EW system needs algorithmic advancements for learning and inference, developments in DM at tempo, data management approaches that annotate characteristics, and architectures that support cognitive reasoning.

However, you can start with a small capability, and grow it for the problem at hand; Chapter 11 describes the incremental steps.

A journey of a thousand miles begins with a single step.

—Chinese proverb,
Dào Dé Jıng by LaoZi, 6th century BCE

References

[1] Mitola, J., III, "Cognitive Radio for Flexible Multimedia Communications," in *International Workshop on Mobile Multimedia Communications,* IEEE, 1999.

[2] Haykin, S., "Cognitive Radar: A Way of the Future," *IEEE Signal Processing Magazine,* Vol. 23, No. 1, 2006.

[3] Kline, R., "Cybernetics, Automata Studies, and the Dartmouth Conference on Artificial Intelligence," *IEEE Annals of the History of Computing,* Vol. 33, No. 4, 2011.

[4] Horne, C., M. Ritchie, and H. Griffiths, "Proposed Ontology for Cognitive Radar Systems," *IET Radar, Sonar and Navigation,* Vol. 12, No. 12, 2018.

[5] Boyd, J., *Destruction and Creation*, Unpublished essay, 1976. Online: http://www.goalsys.com/books/documents/DESTRUCTION_AND_ CREATION.pdf.

[6] US Air Force, Curtis E. Lemay Center for Doctrine Development and Education, *Annex 3-51: Electromagnetic warfare and electromagnetic spectrum operations,* Downloaded 2020-01-23, 2019. Online: https://tinyurl. com/ew-emso-pdf.

[7] De Martino, A., *Introduction to Modern EW Systems,* Norwood, MA: Artech House, 2013.

[8] Poisel, R., *Information Warfare and Electronic Warfare Systems,* Norwood, MA: Artech House, 2013.

[9] Chairman of the Joint Chiefs of Staff, *Joint publication 3-13.1: Electronic warfare*, 2012. Online: https://fas.org/irp/doddir/ dod/jp3-13-1.pdf.

[10] Avionics Department, "Electronic Warfare and radar systems engineering handbook," Naval Air Warfare Center Weapons Division, Tech. Rep. NAWCWD TP 8347, 2013.

[11] Haykin, S., *Cognitive Dynamic Systems: Perception-action Cycle, Radar, and Radio,* Cambridge University Press, 2012.

[12] Boksiner, J., *Electronic Warfare (EW) S&T Community of Interest (CoI),* 2018. Online: https://tinyurl.com/ew-coi-2018.

[13] Haigh, K. Z., et al., "Rethinking Networking Architectures for Cognitive Control," in *Microsoft Research Cognitive Wireless Networking Summit*, 2008.

[14] Malone, T., and K. Crowston, "The Interdisciplinary Study of Coordination," *ACM Computing Surveys,* Vol. 26, No. 1, 1994.

[15] Modi, P., et al., "ADOPT: Asynchronous Distributed Constraint Optimization with Quality Guarantees," *Artificial Intelligence,* Vol. 161, No. 1–2, 2005.

[16] Zhang, X., V. Lesser, and S. Abdallah, "Efficient Management of Multi-Linked Negotiation Based on a Formalized Model," *Autonomous Agents and Multi-Agent Systems,* Vol. 10, No. 2, 2005.

[17] Thampi, S., *Introduction to Distributed Systems,* 2009. Online: https://arxiv.org/ abs/0911.4395.

[18] Troxel, G., et al., "Enabling Open-Source Cognitively-Controlled Collaboration Among Software-Defined Radio Nodes," *Computer Networks,* Vol. 52, No. 4, 2008.

[19] Haigh, K. Z., O. Olofinboba, and C. Y. Tang, "Designing an Implementable User-Oriented Objective Function for MANETs," in *International Conference on Networking, Sensing and Control,* IEEE, 2007.

[20] Nguyen, V. T., F. Villain, and Y. Le Guillou, "Cognitive radio RF: Overview and challenges," *VLSI Design,* Vol. 2012, 2012.

[21] Mitola, J., and G. Q. Maguire, "Cognitive Radio: Making Software Radios More Personal," *IEEE Personal Communications,* Vol. 6, No. 4, 1999.

[22] Haykin, S., "Cognitive Radio: Brain-Empowered Wireless Communications," *IEEE Journal on Selected Areas in Communications*, Vol. 23, No. 2, 2005.

[23] Trigui, E., M. Esseghir, and L. M. Boulahia, "On Using Multi-agent Systems in Cognitive Radio Networks: A Survey," *International Journal of Wireless & Mobile Networks,* Vol. 4, No. 6, 2012.

[24] Rizk, Y., M. Awad, and E. W. Tunstel, "Decision Making in Multi-agent Systems: A Survey," *IEEE Transactions on Cognitive and Developmental Systems*, Vol. 10, No. 3, 2018.

[25] Dorri, A., S. S. Kanhere, and R. Jurdak, "Multiagent Systems: A Survey," *IEEE Access,* Vol. 6, 2018.

[26] Tang, H., and S. Watson, "Cognitive Radio Networks for Tactical Wireless Communications," Defence Research and Development Canada–Ottawa Research Centre, Ottawa, Tech. Rep., 2014.

[27] Haigh, K. Z., "AI Technologies for Tactical Edge Networks," in *MobiHoc Workshop on Tactical Mobile Ad Hoc Networking*, Keynote, 2011.

[28] Cabric, D., "Cognitive Radios: System Design Perspective," Technical Report No. UCB/EECS-2007-156, Ph.D. dissertation, Electrical Engineering and Computer Sciences, University of California at Berkeley, 2007.

[29] Guerci, J., *Cognitive Radar: The Knowledge-Aided Fully Adaptive Approach,* Norwood, MA: Artech House, 2010.

[30] Haigh, K. Z., et al., "Parallel Learning and Decision Making for a Smart Embedded Communications Platform," BBN Technologies, Tech. Rep. BBN-REPORT-8579, 2015.

[31] Üstün, B., W. Melssen, and L. Buydens, "Facilitating the Application of Support Vector Regression by Using a Universal Pearson VII Function-Based Kernel," *Chemometrics and Intelligent Laboratory Systems,* Vol. 81, No. 1, 2006.

2

Objective Function

The EW DM challenge is to build a cognitive controller that automatically maintains near-optimal configurations in highly dynamic environments. Every DM problem has the following components:

- A set of *objectives* for the system to accomplish, with each objective having an established importance;
- A set of *actions* available for the system to use to accomplish the objectives;
- A set of *methods* that evaluate the state to determine consequences (e.g., reinforce success and handle adverse consequences).

Armed with these three components, the system can choose the most appropriate actions for the evolving mission conditions. This chapter describes the heart of the EW decision maker: the set of objectives. Objectives must be operationally relevant, and optimizable in practice [1–4]. We must therefore capture mission goals and constraints in a mathematically optimizable *objective function,* also known as a *utility function.* Goals and constraints come from user and mission needs, from environmental constraints, and from equipment capabilities. The utility function measures all the factors by which to evaluate a decision, so the system can choose the "best" solution. Consider an EW system with N nodes; each node $n \in N$ has:

- A set of *observable parameters* $\mathbf{o_n}$ that describe the RF environment (Section 2.1);
- A set of *controllable parameters* $\mathbf{c_n}$ that the decision-maker can use to change system behavior (Section 2.2);

- One or more *metrics* m_n that provide feedback on how well it is doing, along with their corresponding *weights* w_n (Section 2.3);
- A *utility function* $\tilde{\mathcal{U}}$ that combines the metrics and weights into a single scalar-valued result (Section 2.4).

A *strategy* s_n is a combination of control parameters c_n. The goal is to have each node n choose its strategy s_n to maximize utility $\tilde{U}_n$ of the system.

Table 2.1 collects these symbols. In the main text of this book, we generally use informal notation for these concepts; Section 2.4 presents a formal definition of the true utility function $\mathcal{U}$, and its approximation $\tilde{U}_n$.

Depending on system requirements, a strategy may contribute to EP objectives, or EA objectives, or both. Likewise, a strategy may contribute to communications objectives, or radar objectives, or both. AI is agnostic to the problem domain and does not discriminate between objective types or problem domains. It is the problem definition that determines problem-specific observables, controllables, objectives, and heuristics.

AI techniques apply equally to EP/EA, and comms/radar. The utility function ties AI techniques to problem domain.

Table 2.1
Symbols That Support the Construction of a Utility Function

Symbol	Definition
$n \in N$	A node n in the set of nodes N
t	Timestamp
o_n	Observables o for node n; $o_n(t)$ is the value of o_n at time t; Denote $\overline{o_n} \triangleq \lvert o_n \rvert$
$\mathbf{z}$	Unobservable contextual information
c_n	Controllables c for n; $c_n(t)$ is the value of c_n at time t; Denote $\overline{c_n} \triangleq \lvert c_n \rvert$
m_n	Metrics m for node n; denote $\overline{m_n} \triangleq \lvert m_n \rvert$
w_n	Weights corresponding to each metric in m_n; $\lvert w_n \rvert = \overline{m_n}$
s_n	Strategy s for node n; combination of controllable values c_n
$\mathcal{U}$	The true utility function with no corresponding exact analytical expression
$\tilde{U}_n$	Node n's local estimate of the utility
$\tilde{\mathcal{F}}_n$	Node n's local model of the performance surface
U_n	Observed utility at node n
$\hat{U}_n$	Best-measured (or optimal) utility at node n
g_n	A function over metrics m_n and weights w_n to evaluate utility $\tilde{U}_n$
f_{nk}	A function over observables o_n and controllables c_n to evaluate the kth metric of node n: $m_{nk} = f_{nk}(o_n, c_n) = f_{nk}(o_n, s_n)$ or simply $m_k = f_k(o, s)$

2.1 Observables That Describe the Environment

In EW, observable features, or *observables*, are general descriptors of the signal environment and any relevant contextual information. Observables are computed by ES modules. Each node $n \in N$ has its own set of observables, o_n, and these may differ from other nodes $n' \in N$. There are $\overline{o_n}$ observables at each node. Observables may include raw values, deduced abstractions, or inferred concepts:

- Raw receiver in-phase/quadrature (I/Q) values.
- Descriptions of all detected emitters (friendly, neutral, or adversarial). These features might include noise levels, error rates, Gaussianness, repetitiveness, similarity to own communications signal, or geolocation information, emitter capabilities, and current operational mode.
- Descriptions of all detected targets, including carrier, pulse statistics, and Doppler.
- Descriptions of receiver state (e.g., saturation and antenna characteristics).
- Descriptions of internal software state, which include in a CR setting statistics from the IP stack or end-user applications, such as message error rate, queue lengths, and neighborhood size.
- Descriptions of the mission, including known or forecasted locations, tasks, users, and commander's intent.
- Descriptions of the environment, such as temperature, dust, or topography.

Figure 2.1 illustrates a handful of observables that capture different jammer types. Each detected emitter is associated with its own vector of observables.

Generally, when derived or inferred features can be abstracted or normalized relative to expectations, the decision-maker can more easily determine the relative importance of different features. For example, observables might range from –1 to 1 (strongly "is not" to strongly "is") relative to expectations. Partially observable values can be captured with NaN.

There are many different visualization tools useful for understanding how each environment is different. Figure 2.2 shows four different ways to look at the data; each may be useful for different reasons. The bar charts use the mean values of each observable to give a quick gestalt sense. The box plots show the distribution of values. The scatter plot shows detailed distributions of each observable for every observation of the environment.

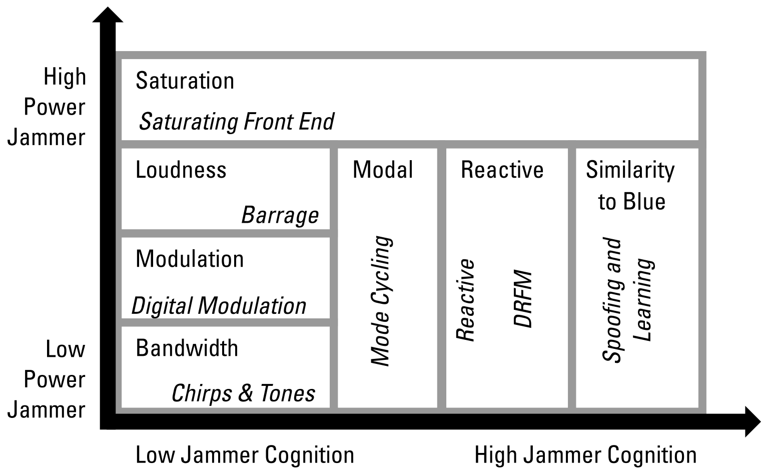


Figure 2.1 Observables are the features that allow a system to make decisions; these observables are useful to characterize different jammer types.

All models are wrong, but some are useful.
—*George Box* [5]

Note that there may be *unobservable* contextual information, *z*. These hidden variables may include broad environmental factors like politics, wind farms, weather, or detailed concepts like undetected emitters or failing hardware components. In theory, every butterfly could impact the performance of the system; in practice, designers should attempt to capture all relevant factors. Unobservable factors are the primary source of modeling error in ES. Models of physical systems are inherently incomplete, and the apparent randomness of a system depends on unseen or unmeasurable variables.

2.1.1 Clustering Environments

As our first foray into using ML in an EW system, we can use *clustering* to determine environment similarity. Clustering is an unsupervised ML method that groups similar items based on their features [6, 7]. We cluster the observable feature vectors into *RF environments* and then draw a corresponding dendrogram to visualize how similar or dissimilar each environment is to the others. Figure 2.3 shows a dendrogram built from 23 different RF environments and three of the corresponding boxplots. The shorter the lines connecting two environments, the more similar they are. Environments 06 and 05 are the two most similar environments and form the first cluster; these two then become a cluster with the pair (env16 and env13). The boxplots for env16 and env06 show that they are, in fact, very similar. Env21 is very different, and is very far away in the dendrogram.

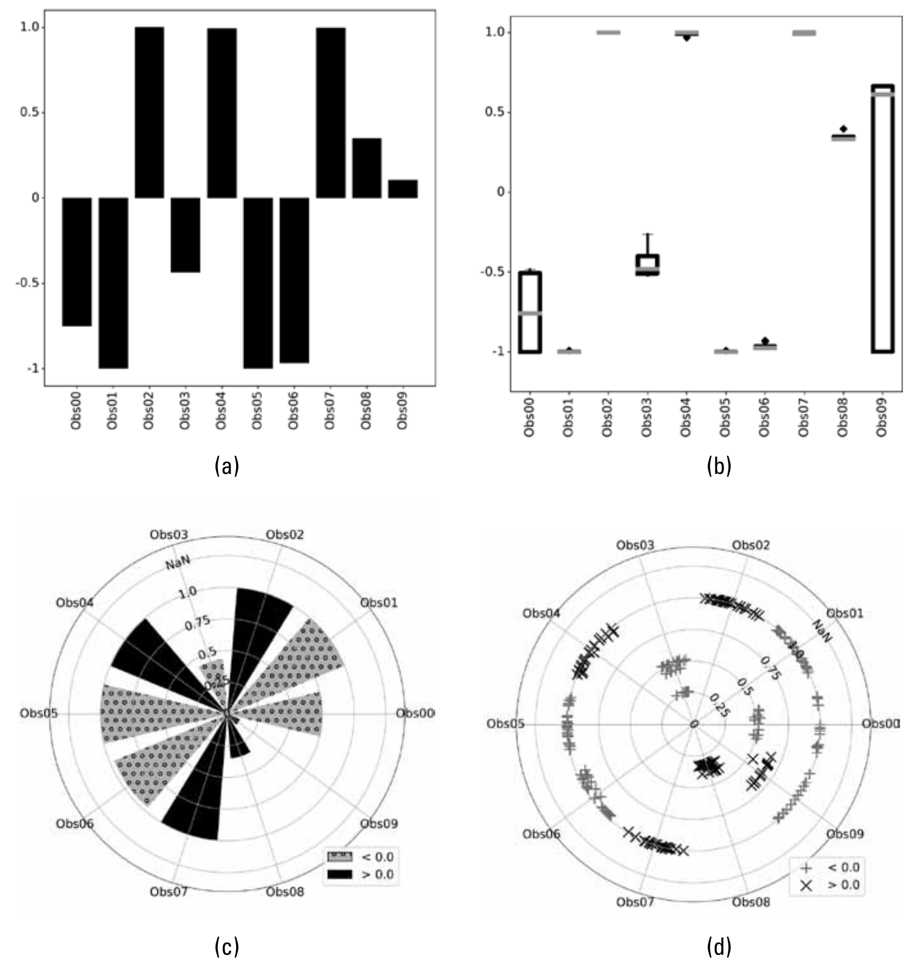


Figure 2.2 Four different ways to visualize environmental observations: (a) a bar chart that shows the mean of the non-NaN values, (b) a box-and-whiskers plot showing the distribution, (c) a bar chart plotted on a polar axis, and (d) a scatter plot of values on a polar axis. Each of these visualizations represent the same underlying observable data. (Figure 7.3 explains boxplots.)

Many clustering methods are available in standard ML libraries (Section 11.2.1). Algorithm 2.1 shows how to plot a simple dendrogram using scikit-learn. Scikit-learn's dendrogram functions require the data to contain no NaN values. The `cleanNaNs()` function therefore replaces NaN values by values: For each environment, for each observable, replace NaNs with the mean value of the non-NaN values of that observable in that environment.

Standard ML libraries will not take advantage of any computations used for multiple purposes and will need reimplementation for the embedded environment. K-means clustering is efficient and appropriate for embedded domains [8]; one can also use the known labels as the initial cluster seeds. If the system

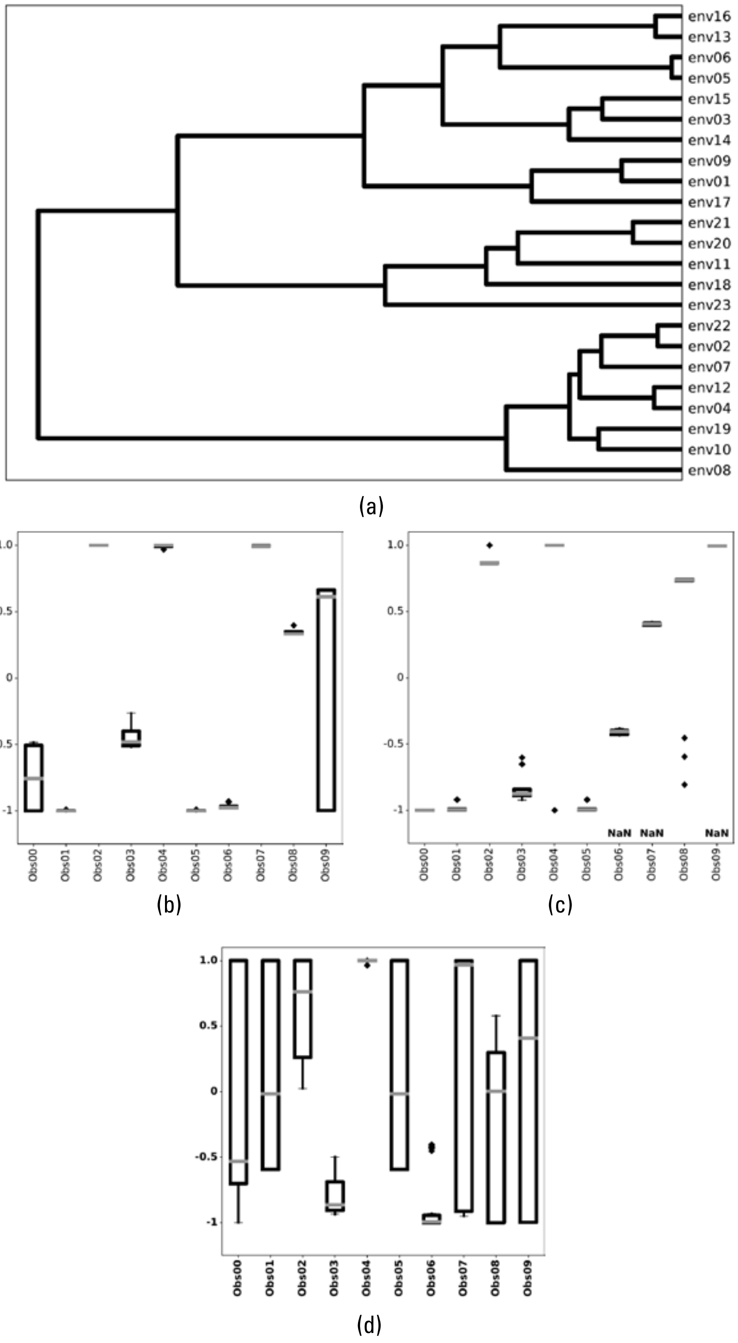


Figure 2.3 A dendrogram visualizes how similar items are to each other. Each environment is described by the multiple features shown in the boxplots. (a) the Dendrogram, where the x-axis indicates the distance between environments; (b) env16; (c) env06; and (d) env21. (Figure 7.3 explains boxplots.)

Algorithm 2.1 This scikit-learn code plots a dendrogram like the one in Figure 2.3 to show the similarity of RF environments.

```
import csv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram
from scipy.cluster.hierarchy import linkage

####################
# Load a dataset: columns are (Environ) label and
# multiple (obs)
def getData():
   environs = pd.read_csv( 'radardata.csv' )
   environs = environs.set_index('Environ')
   return environs

####################
# Efficient method to replace NaNs with the mu
# for that environ/obs
def cleanNaNs(df):
   newDF = pd.DataFrame()
   environs = np.unique( df.index )
   for environ in environs:
      # Instances for this environ
      instances = df.loc[ df.index == environ ]
      # Only replace NaN in columns with NaN values
      nanCols = instances.isnull().any(axis=0)
      for i in instances.columns[nanCols]:
         # Replace NaN in this col with mean of
         # other vals
         instances[i].fillna( instances[i].mean(),
                              inplace=True )
      newDF = newDF.append(instances)
   return newDF

####################
if __name__ == '__main__':
   df = getData()
   df = cleanNaNs(df)
   nEnvirons = len(np.unique( df.index ))
   fig,ax = plt.subplots(figsize=(12,8))
   Z = linkage(df,method='ward')
   dendrogram(Z,truncate_mode='lastp',
              p=nEnvirons,orientation='left')
   fig.tight_layout()
   plt.savefig('dendrogram.pdf')
   plt.close()
```

maintains distances between examples (e.g., to support training an SVM or maintaining diversity in the dataset), then hierarchical clustering methods are also appropriate. The final number of clusters can be driven by available memory or by intercluster distances.

2.2 Control Parameters to Change Behavior

Control parameters, or *controllables*, can include any action available to the node. Each node n has a set of controllables, c_n, that conform to the platform's available capabilities; these are the platform's "knobs", or degrees of freedom. There are $\overline{c_n}$ controllables at each node. A strategy s_n is a combination of control parameters, wherein each controllable has an assigned value. The maximum number of strategies per node, per timestep, is $\Pi_{\forall c} v_c$, where v_c is the number of possible values for the control parameter c; if all $\overline{c_n}$ controllables are binary on/off, then there are $2^{\overline{c_n}}$ strategies, well beyond the ability of a human to manage. For example, if there are five controllables: c_1, c_2, c_3 are binary on/off, c_4 can take 3 values, and c_5 has 10, then there are a total of $2^3 \times 3 \times 10 = 240$ different strategies. When controllables can take continuous values, or many discrete values, there may be an infinite number of strategies per node.

Control parameters are intentionally exposed by the platform for tuning; crosslayer issues are implicitly captured by selecting control parameters from multiple components. For a CR, these are typically parameters in the protocol stack or on the radio, such as those settable through a management information base (MIB) or the mission data file (MDF).

The choice of algorithm or module allows for online reconfiguration of software and firmware flow. Modeled as a binary on/off, each available module has a control parameter x, and $x = 1$ when the module should be invoked, and $x = 0$ when the module should not operate [9].

Hierarchical control parameters are those parameters that are only valid when affiliated parameters or algorithms are enabled. For example, a proactive routing protocol requires the periodicity of *hello* messages, while a reactive protocol requires *time-to-live* thresholds.

We assume a team of heterogeneous nodes. Each node has an optimizer that can configure its capabilities, including techniques in the RF hardware, the FPGAs, the IP stack, and even outside the RF system when appropriate, such as telling the platform to move, or requesting action from a human user. Some examples include:

- Antenna techniques such as beamforming, nulling, and sensitivity time control;

- RF front-end techniques such as analog tunable filters or frequency-division multiplexing;
- Physical layer parameters such as transmit power, notch filters, or number of Fourier transform bins;
- Medium access control (MAC) layer parameters such as dynamic spectrum access, frame size, carrier sense threshold, reliability mode, unicast/broadcast, timers, contention window algorithm (e.g., linear and exponential), neighbor aggregation algorithm, dwell time, pulse repetition interval, and pulse compression length;
- Network layer or multinode coordination parameters such as neighbor discovery algorithm, thresholds, timers, number of transmitter/receiver pairs to combine in a multistatic radar, and number of receivers to jam;
- Encryption settings such as cipher and hash function;
- Application layer parameters such as compression (e.g., jpg 1 versus 10), method (e.g., audio versus video), scan pattern, or how to weight individual radar returns;
- Radar/counterradar items such as modulation type, antenna scan rate and beam-pointing sequence, receiver bandwidth and dwell duration, space-time adaptive processing parameters (STaR), and EA technique(s);
- Concepts outside the RF framework, such as when interacting with a human or the platform (e.g., deploying chaff to change the electromagnetic properties of the air, or deploying a towed array to increase spatial diversity).

Controllables may have associated *costs* that must be accounted for in the metrics. Costs can include ramp-up, maintenance, or ramp-down. Costs may reflect any relevant metric, including time, power, or memory usage. Costs may depend on the value of the controllable; for example, frequent *hello* interval timers will cost more power and bandwidth than infrequent ones. Costs in a set of controllables may be combined in different ways. For example, latency-to-implement is a parallel cost, while transmit-power is summed.

Controllable values may be *mutually exclusive*. For example, a synthetic aperture radar can control integration time *or* azimuth resolution, but not both.

Controllables may not always be available. Different phases of a mission, for example, may enable or disable platform capabilities (e.g., the information-gathering phase of a mission may disable actions that reveal the presence of the node, and then the prosecution phase enables them).

The AI concept of *adjustable autonomy* may also enable or disable capabilities. In an adjustably autonomous system, the human user may empower the platform with different levels of autonomy [10]. For example, a human may choose a low level of autonomy, where she "joysticks" an aerial vehicle, or a high level of autonomy, where she geotags a region of interest that the platform should monitor.

2.3 Metrics to Evaluate Performance

System requirements directly drive the process of identifying appropriate metrics in a complex system. Choosing a good set of metrics is possibly the hardest part of creating a system that effectively chooses actions autonomously. Each metric must capture the connection between available actions and system high-level requirements. The set should be as complete as possible and include no extraneous measurements. Metrics should be relevant, measurable, responsible, and resourced [11].

There are multiple layers of metrics, but the two that matter most are those that impact *mission success* and *EW performance*. Mission success metrics, including platform survival and probability of mission kill, are most relevant to the military decision-makers and users. Metrics evaluate the performance of the subcomponents under different operational conditions and are directly optimizable, either at design time or online through self-assessment and changing behavior.

Metrics quantify how well the EW system satisfies requirements from mission, and situational standpoints. There are $\overline{m_n}$ metrics at each node. A decision-maker chooses controllables $\mathbf{c}_n(t)$ to impact the metrics $m_{nk}(t' > t)$. Metrics may be computed from a model or empirically learned per Section 4.2 and Chapter 7. Metrics include:

- *Effectiveness concepts* include throughput, latency, sensitivity, bit-error rate (BER), message error rate, Pd, Pfa, clutter-to-noise ratio, and broad concepts of jamming effectiveness (J/S or EW BDA). Note that EW BDA is an inferred metric of adversary systems and cannot be directly measured, for example, radar track quality (angle, range, velocity, covariance matrix estimate).
- *Cost factors* include time, power, control overhead, timeline usage, probability of detection, or even platform wear-and-tear.
- *Other concepts* include DM *uncertainty*, *heuristic* preference, or *flexibility* to surprise. For example, designers may prefer FPGA-based actions over software-based actions, or for the system to remain relatively stable over time (not changing actions at each time step). The *value of information* (Section 6.1.4) can also be an important metric. When the system uses ML to learn the models $f_k(o,s)$, *decision confidence* or *applicability domain* may become important metrics. (The applicability domain of a statistical model is the

subset of the underlying patterns on which the model is expected to give reliable predictions.)

Haigh et al. [1] examines four potential metrics: (1) application-level quality-of-service (QoS) requirements of multiple heterogeneous traffic flows, (2) node-level constraints on battery lifetime, (3) node-level constraints on probability of detection, and (4) cost in terms of how much bandwidth the network wastes as a byproduct of application-level data transfer. Table 4.1 outlines some metrics for 5G localization.

There is an inherent tension between metrics and observables, in that many items could be used either way. BER, for example, may be an observable to support a low probability of intercept/low probability of detection (LPI/LPD) metric, or it may be a metric that the EP system aims to minimize. The choice of whether to call a feature a metric or an observable is a function of the mission and the problem definition. When building the system, it is frequently most appropriate to start with the simplest metric (i.e., the easiest to measure), while using more complex or inferred features as observables. Once the system has stabilized, one can convert features from observables to metrics. For example, the initial system may optimize BER, and subsequently augment to broader concepts of throughput and latency.

Each metric may have an associated *weight* that specifies how the metric impacts the overall performance. A scalar value could be used to accumulate a weighted sum of metrics. A lambda function could specify more flexible structures to handle metrics that are not uniform or linear. For example, if BER is above a threshold, or latency is beyond an acceptable time limit, then the metric can be zeroed. Figure 2.4 sketches some possible weight functions.

Metric values should be normalized, so that large values are evaluated at the same level as small values. Metrics may be measured in linear scale or logarithmic scale (e.g., dB). The normalization approach may differ based on the

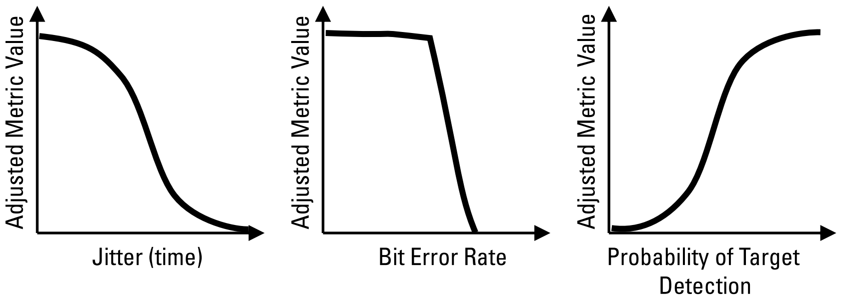


Figure 2.4 Weights can modify metric values in different ways and moreover may change depending on the mission (e.g., whether the emitter is in search, acquisition, or track mode).

scale of choice. Weights control their relative values. When maximizing utility, effectiveness metrics would have positive weights, while costs would have negative weights.

The most effective metrics are ones with rapid, measurable feedback. Feedback is computed by ES/BDA and reflects the observed value of the metric to compare against the value estimated by learned models. Useful questions include:

- Can the metric be directly measured? BER can be directly measured, while EW BDA is instead inferred. The system can use both types, but a directly measurable one is significantly easier to develop and leverage.
- How many nodes contribute to the measurement? BER can be measured on a single node; roundtrip time is measured by all of the nodes in a single flow, and EW BDA could be constructed locally or over a small team of nodes. Generally, fewer nodes means that the computation is faster and more accurate. Multinode measurements introduce latency and can be computed with techniques such as consensus propagation [12].
- How rapid is the feedback? BER is relatively instantaneous, Throughput has delay usually measured in seconds. Target tracking accuracy is usually measured in minutes, and number of lives lost on a mission is measured in hours or days. As the feedback interval increases, it is harder for the system to connect specific actions to their effect. While in theory, the system could automatically determine these relationships, the amount of time and data required becomes prohibitive.
- Does the metric value change with time or other contexts? For example, target tracking in a surveillance radar is measured in minutes, but fire control only needs seconds to generate a firing solution. Moreover, accuracy may matter for only the initial shot selection and midcourse/end-game aspects of weapons guidance. If the EW system deceives an enemy missile guidance radar for 90% of flight-out time but did not deny the final 10% of the fly-out, it still fails. Metrics must be evaluated when they are most applicable to ensure they are meaningful in the context of the overall engagement outcome in terms of success. Some of these concerns can be handled with lambda functions, others may require changing the structure of the utility function.

2.4 Creating a Utility Function

In systems that autonomously choose actions, a utility function captures the objectives of the stakeholders. The structure of the utility function depends on the specific tasks and capabilities of the system. The function combines the metrics in a way that is operationally meaningful and optimizable in practice. Callout 2.1 presents a formal definition of a utility function, and how to simplify it so that it can be optimized it in practice.

Generally, a comms objective function will be more nuanced than a radar objective function, because there are more metrics available. Joint optimization (EP/EA or comms/radar) adds another layer of complexity.

In a complex system, requirements drive the process of choosing metrics and weights and how to combine them. Goals and constraints come from user and mission needs, from environmental constraints, and from equipment capabilities. Haigh et al. [1] present an approach to allow in-mission manipulation of the utility function through *policy bundles*. It is important to capture goals from all stakeholders.

A simple objective function computes the weighted sum of the metrics:

$$\tilde{U}_n = \sum_{k=1}^{\overline{m_n}} \left(w_{nk} \times m_{nk} \right)$$

where each weight w_{nk} is a scalar value, and each metric m_{nk} is a function of the node's observables and controllables, $m_{nk} = f_{nk}(\mathrm{o}_n, \mathrm{c}_n)$. When weights are lambda functions, a possible function is:

$$\tilde{U}_n = \sum_{k=1}^{\overline{m_n}} w_{nk} \left(m_{nk} \right)$$

When metrics are not independent, alternate structures may be appropriate. For example, any communications flow before a deadline should be evaluated using jitter, but after the deadline, there is no benefit for delivering data. If latency $m_1 < 3$ is a hard constraint, and jitter $m_2 = 0$ is a soft constraint, then a good mathematical representation of this concept is:

$$\tilde{U}_n = w_1 \left(3 - m_1 \right) \cdot e^{-w_2 m_2^2}$$

where $w_1(\cdot)$ is the unit step function, and $w_2 > 0$ is a parameter that characterizes how soft the jitter constraint is. Figure 2.5 shows the utility values from this function. Note that estimates for m_1 and m_2 may still depend on learned models $f_1(o,s)$ and $f_2(o,s)$ respectively.

As an example, ES systems have a goal of 100% for probability-of-intercept (POI). The only way to achieve 100% is to construct a staring architecture that monitors the spectrum continuously. Architectures that cannot provide continuous monitoring must implement scan patterns with sequential tuning and dwelling to intercept emitters when they are transmitting and while the receiver is dwelling at their frequency. A simplified utility function in this case is POI = f(RF environment observables, scan rate, revisit patterns, dwell, ..., frequency). As with

Callout 2.1 Formal problem definition: the true utility function $\mathcal{U}$ must be simplified to make it optimizable in practice.

Consider an EW system with N heterogeneous nodes. Each node $n \in N$ has the following:

- A set of $\overline{o_n}$ observable features, denoted $\mathbf{o}_n \triangleq (o_{n1}, o_{n2}, \ldots, o_{n\overline{o_n}})$.
- A set of $\overline{c_n}$ control parameters, denoted $\mathbf{c}_n \triangleq (c_{n1}, c_{n2}, \ldots, c_{n\overline{c_n}})$.

Denote unobservable contextual information as z.

To capture changes over time, denote $\mathbf{c}_\mathbf{n}(t)$ to be the value of c_n at time t, and do likewise for $\mathbf{o}_\mathbf{n}$ and $\mathbf{z}$.

Associated with the system is a real-valued scalar utility measure $\mathcal{U}(t)$ that characterizes the global, network-wide performance measure at time t. This measure is a function $\mathcal{F}$ of all the control parameters, observable features, and unobservable factors, for all nodes, since the beginning of the mission:

$$\mathcal{U}(t+1) = \mathcal{F}\Big(\forall n \in N\big(\mathbf{o}_n(0), \ldots, \mathbf{o}_n(t), \mathbf{c}_n(0), \ldots, \mathbf{c}_n(t), z(0), \ldots, z(t)\big)\Big)$$

The goal is to solve the following distributed optimization problem:

> Design a fully distributed algorithm where every node n determines its control parameter values $\mathbf{c}_n(t)$ using only its own previous observable features $\mathbf{o}_n(0)$, ..., $\mathbf{o}_n(t)$ and control values $\mathbf{c}_n(0)$, ..., $\mathbf{c}_n(t)$ such that $\mathcal{U}(t + 1)$ is maximized (or minimized) for each t.*

The challenge is that we cannot define an exact analytical expression for $\mathcal{F}$, due to the unobservable factors z and complex intra- and internode interactions. Therefore, the algorithms described in this book and in the related work generally simplify the problem by using local, memory-less approximations of $\mathcal{F}$: Each node approximates the true utility $\mathcal{U}$ with $\tilde{U}_n$.

$$\tilde{U}_n(t+1) = \tilde{\mathcal{F}}_n\big(\mathbf{o}_n(t), \mathbf{c}_n(t)\big)$$
$$\forall n \in N, \mathcal{U}(t) \approx \tilde{U}_n(t)$$

Essentially, node n assumes that decisions made historically ($\mathbf{c}_n(t' < t)$), and by nearby nodes ($n' \neq n$), will be *implicitly* observable in $\mathbf{o}_n(t)$. For

* The optimization goal can be expanded to compute the Utility of the entire mission, rather than for each consecutive instantaneous time value t.

example, if a neighbor n' increases data rate, node n will observe increased congestion. When nodes $n' \neq n$ *explicitly* share the previous observables or control settings with node n [i.e., $\mathbf{o}_{n' \neq n}(t' < t)$ or $\mathbf{c}_{n' \neq n}(t' < t)$], these values become additional features in $\mathbf{o}_n(t)$.

To configure the nodes, at each timestep t, each node selects a strategy $\mathbf{s}_n(t)$ that is a particular setting of controllables $\mathbf{c}_n(t)$ that optimize performance on this surface:

$$\mathbf{s}_n(t) = \arg\max_{c_n(t)} \tilde{\mathcal{F}}_n(\mathbf{o}_n(t), \mathbf{c}_n(t))$$

(Or alternatively, argmin.) There are $\Pi_{\forall c_n} v_{c_n}$ candidate strategies for node n at each timestep, where v_{c_n} is the number of values that a given controllable c_n can assume. This number becomes infinity when controllables are continuous-valued.

To support changing the system's behavior during the mission, we express $\tilde{U}_n$ as a set of $\overline{m_n}$ metrics, m_n, allowing a user to change corresponding weights w_n during the mission.† Thus for a node $n \in N$, $\tilde{U}_n(t)$ is a function of its metrics and their weights, and each metric is in turn a function of the observables and controllables at that node:

$$\begin{aligned} m_{nk}(t+1) &= f_{nk}(o_n(t), c_n(t)) \\ v_{nk}(t) &= (w_{nk}(t), m_{nk}(t)) \\ \tilde{U}_n(t) &= g_n\left(v_{n1}(t), v_{n2}(t), \ldots, v_{n\overline{m_n}}(t)\right) \end{aligned}$$

The functions f_{nk} are models created by hand or empirically learned. The model f_{nk} at time t estimates the metric value $m_{nk}(t+1)$ from the observables $\boldsymbol{o}_n(t)$ and controllables $\boldsymbol{c}_n(t)$, allowing a decision-maker on node n to select the best control settings.

Elsewhere in this book, we rarely need this level of detailed notation. We therefore use the following shorthand: $m_k = f_k(o, s)$, or simply $m = f(o, s)$. Likewise, $\tilde{U}(s_i)$ is shorthand for the utility of candidate strategy s_i.

Unlike more common notations, where $y = f(x)$, we explicitly distinguish between observables and controllables to expose the system's ability to make decisions. This notation is similar to that of MDPs, where the reward is expressed as a function of state and action: $U = R(s, a)$, in Section 6.1.3.

† Weights do not need to be scalar values; lambda functions may be appropriate for some metrics.

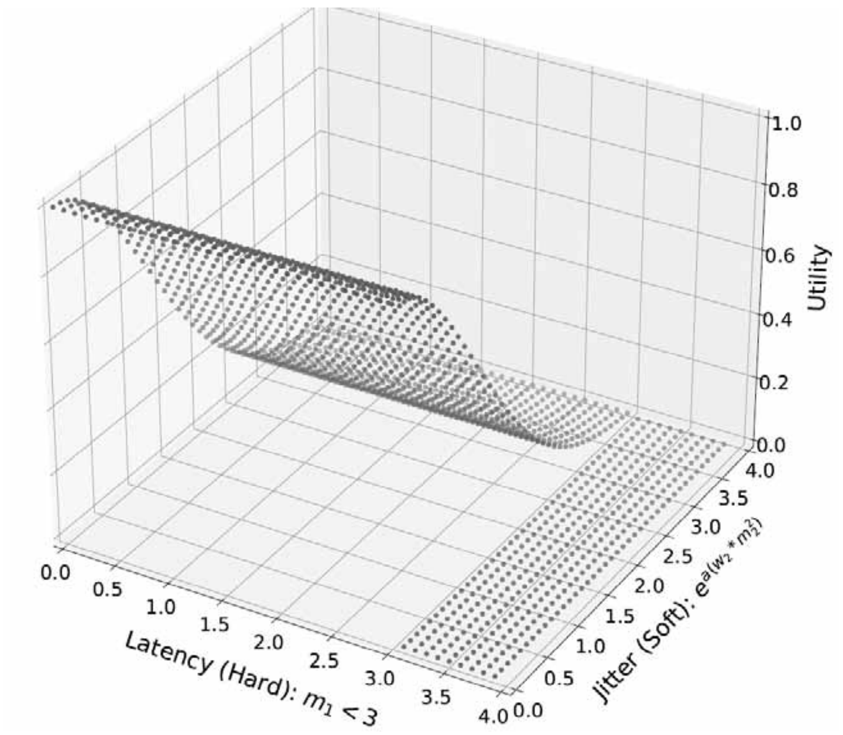


Figure 2.5 This QoS metric combines the hard deadline of latency and a soft constraint of jitter [1]; here we use $w_2 = 0.3$.

most BDA metrics, POI cannot be directly measured and must be inferred by ES/BDA functions.

Section 5.1.1 describes different approaches for using and optimizing a multiobjective utility function. Game theory (Section 6.2) may add a layer of probabilities over the utilities to handle environments with selfish-but-rational individuals.

2.5 Utility Function Design Considerations

From an architectural design and implementation perspective, consider the following suggestions:

- Choose a single, rapid, easy-to-measure metric, such as BER, to test the decision pipeline.
- Structure the function so it is easy to add new metrics and new requirements, thus handling both planned and unplanned changes in the system.

- Keep each metric computationally independent from other metrics until the last logical moment where they need to be combined, thereby supporting rapid changes in priorities *during a mission.* Notably, this separation of concerns is critical if you use ML techniques to learn the relationship between the controllables and the metrics $m_k = f_k(o,s)$. Some systems will build a single model $\tilde{U}_n = \tilde{\mathcal{F}}_n(o,s)$, burying the component metrics; this approach means either that the system can never be retasked or that previous experience has to be completely restructured.

  As an example, by keeping Pd separate from Pfa, the system can dynamically choose its operating point along the receiver operating characteristic (ROC) curve.

  Section 4.2 describes approaches to learning metrics from empirical data, while Chapter 7 describes the process of learning *during* a mission.
- Because different nodes might have different capabilities and requirements, different nodes (or types of nodes) may have different utility functions. There is no need for the utility function to be identical on every node: Each node may have its own unique function. Utility across a heterogeneous set of nodes is additionally complicated because of intermittent connectivity and dynamic network membership. The true utility function $\mathcal{U}$ takes into account a fully heterogeneous team, wherein each node has different capabilities and tasks. The approximate utility function $\tilde{U}_n$ captures utility for a specific node n and may use different metrics, weights, observables, and controllables from any other node.
- Identify mechanisms for the nodes to share information efficiently when a metric is an inherently multinode concept. Consensus propagation is an effective approach that minimizes measurement latency and does not require the system to know how many nodes are in the network [1, 12]. Each node n computes its local contribution to the shared metric, incorporating estimates from other nodes n' when available. Section 6.1.4 discusses knowledge sharing.
- Do not create a centralized node that determines either strategies or utilities for distributed nodes. A centralized system introduces communication latencies, information inconsistencies and overload, and most importantly, a single point of failure. Only use a centralized node for advice that does not need to happen rapidly, and as an unreliable information archive (assuming sufficient communications are available).

The objectives defined in the EW doctrine may need to be adjusted or elaborated to include concepts not traditionally incorporated. For example, degrading the adversary's DM ability, or causing those processes to fail (e.g., defeating aircraft track automation or reducing the number of targets that an adversary operator can individually manage by requiring more human involvement).

2.6 Conclusion

The goal of a fully capable, effective EW system rests on a foundation of a utility function that measures system performance, so that it can select good strategies for the mission. The utility function captures user objectives, mission goals, and environmental constraints in a mathematically optimizable approach.

Chapter 4 describes ES and the observable features upon which to base decisions; the structure of the objective function supports observables that are raw values, derived from physics models, or inferred by learned models. Chapters 5 and 6 show how to use the objective function for DM: how to choose strategies for EP and EA using optimization and scheduling techniques, and the longer-term EBM DM horizon using planning approaches.

References

[1] Haigh, K. Z., O. Olofinboba, and C. Y. Tang, "Designing an Implementable User-Oriented Objective Function for MANETs," in *International Conference on Networking, Sensing and Control,* IEEE, 2007.

[2] Jouini, W., C. Moy, and J. Palicot, "Decision Making For Cognitive Radio Equipment: Analysis of the First 10 years of Exploration," *EURASIP Journal on Wireless Communications and Networking,* No. 26, 2012.

[3] Arrow, K., "Decision Theory and Operations Research," *Operations Research,* Vol. 5, No. 6, 1957.

[4] Roth, E., et al., "Designing Collaborative Planning Systems: Putting Joint Cognitive Systems Principles to Practice," in *Cognitive Systems Engineering: A Future for a Changing World,* Ashgate Publishing, 2017, Ch. 14.

[5] Box, G., "Robustness in the Strategy of Scientific Model Building," in *Robustness in Statistics,* Academic Press, 1979.

[6] Rokach, L., and O. Maimon, "Clustering Methods," in *Data Mining and Knowledge Discovery Handbook,* Springer, 2005.

[7] Bair, E., "Semi-Supervised Clustering Methods," *Wiley Interdisciplinary Reviews. Computational Statistics,* Vol. 5, No. 5, 2013.

[8] MacQueen, J., "Methods for Classification and Analysis of Multivariate Observations," in *Berkeley Symposium on Mathematical Statistics and Probability,* 1967.

[9] Haigh, K. Z., S. Varadarajan, and C. Y. Tang, "Automatic Learning-Based MANET Cross-Layer Parameter Configuration," in *Workshop on Wireless Ad Hoc and Sensor Networks,* IEEE, 2006.

[10] Mostafa, S., M. Ahmad, and A. Mustapha, "Adjustable Autonomy: A Systematic Literature Review," *Artificial Intelligence Review,* No. 51, 2019.

[11] US Army Headquarters, "Targeting," Department of the Army, Tech. Rep. ATP 3-60, 2015.

[12] Moallemi, C., and B. Van Roy, "Consensus Propagation," *Transactions on Information Theory,* Vol. 52, No. 11, 2006.

3

Machine Learning Primer

AI is a field within computer science that draws from mathematics, engineering, and neuroscience. Figure 3.1 illustrates a few of the many subfields of AI. ML is just one of many subfields, and artificial neural networks (ANNs) are one of many families of techniques within ML.

"AI" is not synonymous with "deep learning."
$AI \supset ML \supset DL$.

This book is emphatically not a treatise on AI. It is a sampling of AI techniques that are most immediately applicable to cognitive EW. The book *AI: A Modern Approach* [1] has an in-depth discussion of AI concepts, written from the perspective of building a complete intelligent agent.

Chapters 5 and 6 discuss the AI techniques of planning, optimization, distributed AI, and human factors concepts, because these form a logical unit within DM.

ML, however, is more broadly relevant throughout the EW system. Cognitive EW techniques rely on ML to model the spectrum, understand the participants, and learn how to effectively plan and optimize. In-mission learning is an essential part of a complete cognitive EW system; without this capability, the system will never handle novel emitters.

A learning algorithm uses empirical data to learn a model of the space. Figure 3.2 illustrates the steps for building an ML model. Every ML model is trained and tested with the same basic process. Algorithm 4.1 shows an example of code underlying this loop.

Learning algorithms are commonly grouped into *supervised* and *unsupervised* approaches, where the algorithm is given the ground truth labels Y for training, or not, respectively. A supervised learner creates a function $f: X \rightarrow Y$ from an

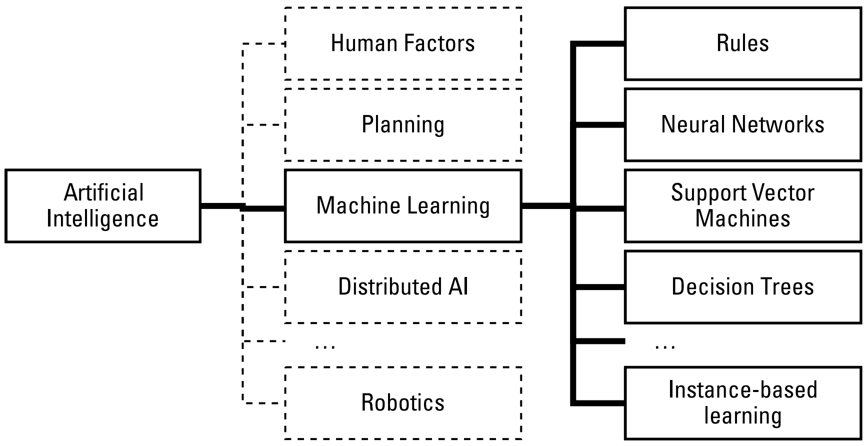


Figure 3.1 Like math, which has calculus, geometry, and algebra, AI has many subfields, including planning, ML, and robotics; the area of ML comprises many techniques.

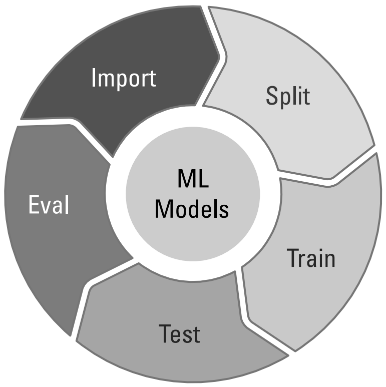


Figure 3.2 Every ML model is trained and tested with the same underlying process.

input space X to an output space Y, using a finite number of instances (X, $f(X)$). The model f approximates the true output space $\mathcal{F}$. An unsupervised learner uses only X to build the model that captures hidden patterns in the unlabeled data. For example, a supervised modulation classifier would attempt to label every observation of the RF environment with the true transmitted modulation. An unsupervised modulation-clustering algorithm would group similar observations without knowing their labels, and the output Y is a value corresponding to the cluster number for each observation.

Semi-supervised learning approaches have a small number of labeled examples, and many unlabeled ones. In a *reinforcement learning (RL)* approach, the learner gathers the labels itself by taking actions in the environment. RL forms the basis of in-mission learning (Chapter 7).

Table 3.1 lists some of the common uses of ML; this book presents many other examples for EW. Section 3.6 presents some of the trade-offs that an EW engineer should consider when choosing an ML algorithm. Section 11.2 lists ML toolkits useful for rapid prototyping and supporting datasets.

3.1 Common ML Algorithms

There are many ML algorithms; Figure 3.1 touches on only a few of the more common ones used in EW. For a deeper analysis, refer to *The Hundred-page Machine Learning Book* [2], or even deeper to *The Elements of Statistical Learning* [3]. Alternatively, Vink and de Haan [4] have short descriptions of ML algorithms in the context of target recognition, and Kulin et al. [5] presents some of the math around ML for spectrum learning.

Instance-based methods store all training instances in memory, and the inference phase compares new instances against all training instances. There is no training time, but inference time is a function of the size of the training data. Table-lookup, hashing, and nearest neighbor methods are common instance-based approaches [1]. If an EW system automatically adds new threats to the library, then the traditional EW activity of “looking up the threat in the library” is an instance-based learning method.

Model-based methods use the training data to create a model that has parameters; the model size (number of parameters) does not change with the amount of data. Training time is a function of the size of the training data, while inference time is constant. Support vector machines (SVMs) (Section 3.1.1) and ANNs (Section 3.1.2) are two that have particular relevance for EW.

3.1.1 Support Vector Machines

SVMs are a class of ML methods used for clustering, classification, regression, and outlier detection [6–8]. SVMs have high generalization performance, par-

Table 3.1
Common Applications for ML Algorithms

Application	Description	EW Example
Classification	Assigning a label from a set of discrete classes (supervised)	Section 4.1.3
Regression	Estimating a numerical value (supervised)	Section 4.2
Clustering	Grouping similar instances together (unsupervised)	Section 2.1.1
Outlier detection	Identifying instances that are very different from typical examples (unsupervised)	Section 4.4

ticularly for complex nonlinear relationships and when the available training data is limited.

SVMs use a subset of training data points (called support vectors) in the decision function of the SVM algorithm. Each SVM constructs a decision function that estimates the decision boundary, known as the *maximum-margin hyperplane,* and an acceptable hard or soft margin around that hyperplane. The support vectors are the samples on the margin, illustrated in Figure 3.3.

SVMs proficiently accomplish nonlinear classification by using kernel functions or "tricks" [9]. With a kernel trick, nonlinear classifiers map their inputs into high-dimensional feature spaces.

3.1.2 Artificial Neutral Networks

ANNs are a class of algorithms inspired by the way biological nervous systems, such as the brain, process information. These algorithms simulate neurons and their interconnections to replicate how the brain learns. ANNs have their conceptual roots in the 1940s [10], and in 1958, Rosenblatt designed the *perceptron,* an algorithm for pattern recognition [11]. Werbos's 1975 *back-propagation* algorithm enabled practical training of multilayer networks: It distributes the error term back up through the layers by modifying the weights at each node [12].

ANNs consist of input, hidden, and output layers, illustrated in Figure 3.4. The number of layers refers to the *depth* of the network. Modern ANNs typically have many layers and are thus referred to as *deep learning*, *deep neural networks*, or simply *DeepNets.* DeepNets can discover patterns or *latent features* that are too complex for a human to extract or design.

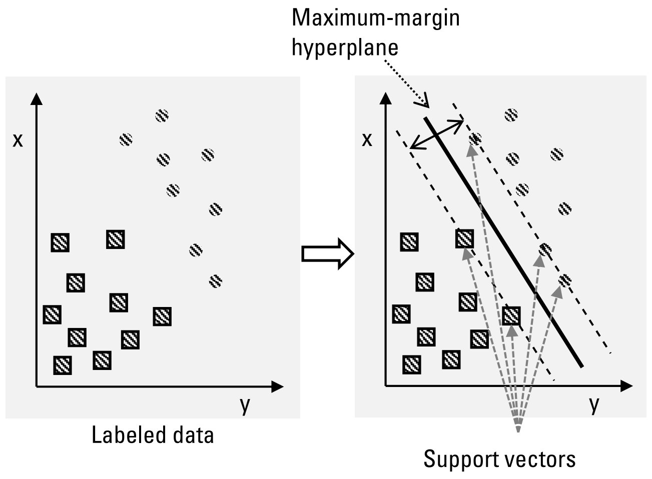


Figure 3.3 SVMs efficiently learn complex functions with limited data.

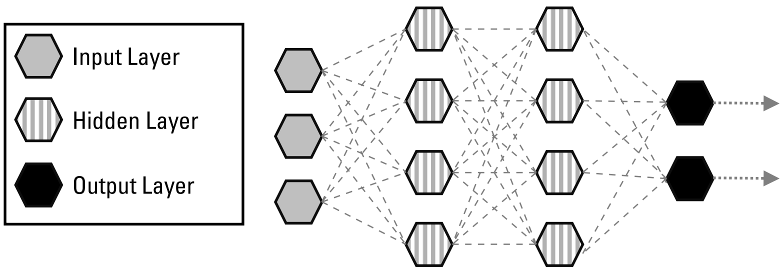


Figure 3.4 ANNs consist of input, hidden, and output layers. The number of layers refers to the depth of the network.

Many DeepNet architectures exist, with new ones emerging regularly. The most comprehensive references are by Bengio, Hinton, and LeCun, who won the Turing award for their work in 2018 [13, 14]. Common architectures include:

- *Convolutional neural networks* (CNNs) [15, 16] are neural networks for processing data with a known grid-like topology. A typical CNN consists of three types of layers: convolutional, pooling, and fully connected, illustrated in Figure 3.5. *Convolutional layers* allow the network to learn filters that respond to local regions of the input. *Pooling layers* streamline data-processing: They reduce the outputs of neuron clusters at one layer into a single neuron in the next layer. *Fully connected layers* connect every neuron in one layer to every neuron in the next layer.
- *Recurrent neural networks* (RNNs) [17] belong to a family of neural networks for processing sequential data [14]. RNNs have feedback connections (i.e., memory). They address the temporal nature of their input data by maintaining internal states that have memory [18].
- *Temporal CNNs* handle sequence-related, time-related, and memory-related deep learning [19] and may render RNNs obsolete.

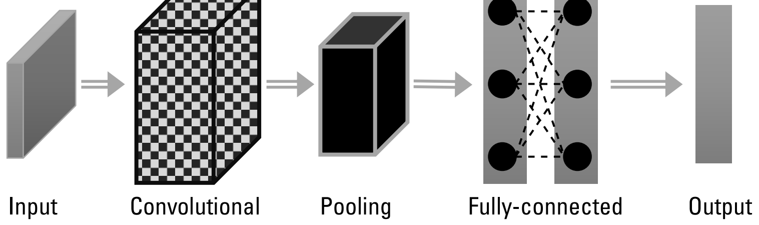


Figure 3.5 CNNs process data with a known grid-like topology.

- *Autoencoders* learn efficient data encodings, building a model with a bottleneck layer; the bottleneck is the efficient encoding, and the output layer is the reconstruction of the input. The autoencoder tries to generate, from the reduced encoding, a representation as close as possible to its original input. They effectively learn to eliminate noise (i.e., they learn the common case) and are often used as anomaly detectors [14, 20].
- *Siamese neural nets* use the same weights on multiple networks, while training on different input data. They require less training data than other methods [21].
- *Kohonen networks,* also known as *self-organizing maps* (SOMs), produce low-dimensional representations of the input space and are used for dimensionality reduction and visualization.
- *Generative adversarial networks (GANs)* are a system of two competing neural networks, each trying to improve the accuracy of its predictions. A GAN sets a *generative network* against a *discriminative network* in a competition: The generative network's goal is to fool the discriminative network. GANs are often used to create synthetic data.

The ability for a model to estimate its own confidence (i.e., when it should be trusted) is an important emerging capability [22].

Popular ANN architectures are available online [23–25] and have been presented in the context of RF [26].

3.2 Ensemble Methods

Using multiple classifiers in an *ensemble* [1, 27–32] can improve prediction accuracy and increase robustness to adversarial attacks, because an ensemble combines the predictions from multiple diverse models. Election predictions, for example, combine the results from many different polls, with the hope that flaws in one poll will compensate for flaws in other models. Section 7.1.1 shows a simple example in the EW BDA environment.

Common ensemble approaches include bagging, boosting, and Bayesian-model averaging; most ML toolkits offer additional techniques. Simple *majority voting* chooses the most common prediction. *Bagging* gives each model in the ensemble equal weight. *Boosting* incrementally builds the ensemble by training each new model to emphasize training instances that previous models misclassified. *Bayesian-model averaging* uses weights based on the posterior probabilities of each model.

3.3 Hybrid ML

Symbolic AI approaches manipulate symbols (often human-readable), while *non-symbolic* approaches operate on raw data. Decision trees are frequently symbolic in nature, while DeepNets are usually nonsymbolic. In recent years, *hybrid* approaches combine the two; they use symbolic knowledge to construct features, reduce the search space, improve search efficiency, and explain resulting models. Hybrid approaches often exploit domain-specific knowledge and heuristics to find solutions more quickly [33–41]. Hybrid approaches are also known as *knowledge-based ML* or *neural-symbolic AI*.

Hybrid approaches essentially "bootstrap" the learning process. Figure 3.6 illustrates the idea: An analytical model provides an initial framework or educated guess about the result, and the empirical data refines the prediction to match observed experience. The hybrid approach enables the learner to work adequately even with no training data and work well after real-world training. Figure 3.7 sketches how DeepNets, classic ML, and hybrid approaches perform as a function of available data.

Section 4.1.1 presents an example of how to combine traditional features with DeepNet models. Section 7.3.3 describes how to combine a DeepNet for latent features with an SVM for rapid in-mission-learning. Section 6.3 presents a variety of ways to leverage human expertise in model design.

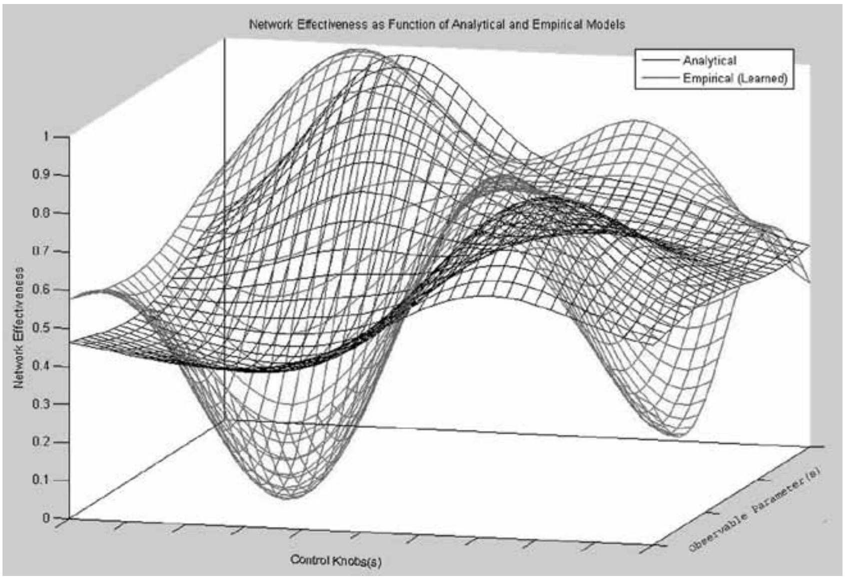


Figure 3.6 Analytical models estimate the performance surface as a function of $\overline{o_n}$ dimensions of observables and $\overline{c_n}$ dimensions of controllables; empirical models refine the prediction. (From [35].)

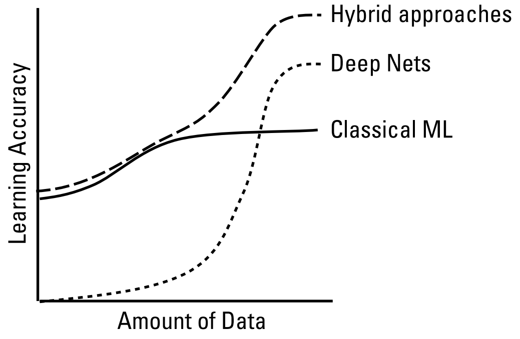


Figure 3.7 DeepNets identify latent features in the data, whereas classical ML approaches rely on traditional feature engineering.

3.4 Open-Set Classification

Open-set classification techniques create new classes on-the-fly, after the initial model is trained. They handle unknown data encountered during the mission that was not anticipated during the training phase.

One option is to choose an ML model that effectively learns from one example; *k*-nearest neighbor (*k*NN) and SVMs are good choices [42]. *k*NNs have no retraining time, but each inference is linear in the number of training examples. SVMs recompute the model (roughly $O(n^2)$ for n as number of training examples), but can be very efficient at inference.

Other approaches include autoencoders, zero- or low-shot learning [43–48] and anomaly detection (Section 4.4). Low-shot learning, for example, uses the training data to constructs a *latent embedding* of the important features and then uses this embedding to create new classes on-the-fly. Data augmentation (Section 8.3.3) can also help ensure that the original training data covers more of the unknown classes.

3.5 Generalization and Meta-learning

Generalization refers to a model's ability to adapt properly to new, previously unseen data, drawn from the same distribution as the one used to create the model. A table-lookup, for example, has no ability to generalize. *Overfitting* means that the model captures the training data too well and therefore is unlikely to do well with new data, while *underfitting* doesn't even capture the training data well (Figure 3.8). Good generalization means finding the right trade-off between under- and overfitting the data.

Over- and underfitting is controlled by tuning *hyperparameters* that control how the algorithm works, in a process known as meta-learning [49–52]. Each algorithm has its own hyperparameters, for example decision trees tune *MaxDepth*

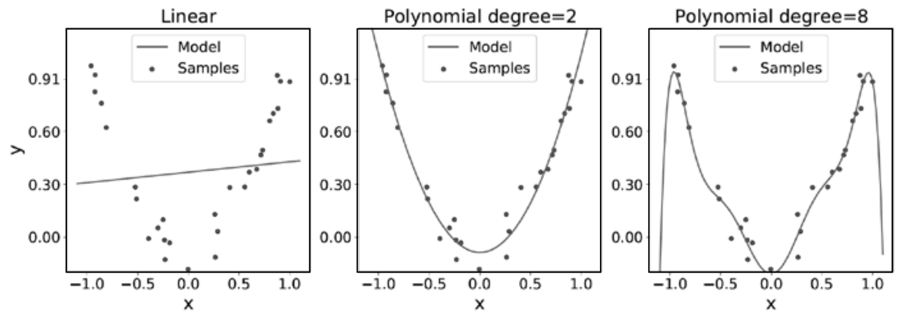


Figure 3.8 Linear models often underfit, and models tend to overfit if there are many more features than training instances. (Underlying data is $y = x^2 + \varepsilon$.)

and *MinSamples*, while SVMs tune C and γ (cost for misclassified points, and influence of single instances, respectively), and DeepNets might use early stopping and activation dropout. If using one of the standard ML toolkits (Section 11.2.1), hyperparameters are usually the arguments to the function. Section 5.1.3 talks more about meta-learning, emphasizing its role in optimization. Meta-learning is often part of a RL solution.

Other approaches to improve generalization include ensemble methods (Section 3.2), hybrid models (Section 3.3), and training approaches like batch-norm [53, 54], Langevin methods [55], or entropy filtering [56].

High-dimensional problems, with more features than training examples, (i.e., $(\overline{o_n} + \overline{c_n}) \gg n$) often need different approaches [3]. Managing the data with diversity, augmentation, and forgetting is another important step (Section 8.3).

The *domain of expertise* indicates what the model should be expected to capture well: When a new observation falls within the domain of expertise, the model should do better than when it falls outside. In Figure 3.8, for example, the training data falls in the range [–1.0,1.0]. A value of 2.0 is outside the domain of expertise. The two-degree polynomial generalizes well outside this range, but the other two models generate increasing error the further away from the expected range. Ideally, the model should compute its own confidence.

3.6 Algorithmic Trade-Offs

Choosing which ML technique to apply to the spectrum understanding problem depends on a variety of factors. There is no one model that will always outperform every other model [2, 57, 58]. These factors include what the task is, what kind of data is available, the solution goals, and operating constraints. Table 3.2 lists some of the questions that must be addressed when choosing the algorithm.

As one example, DeepNets effectively identify latent features in the data. DeepNets, however, depend on a large set of diverse well-labeled training data.

Table 3.2
Questions to Address When Choosing an Algorithmic Solution to Fit an ML Task

Factor	Questions to consider
Task	Are you trying to understand the environment? Predict what will happen? Control actions? Adapt Continuously? Scott Page, in *The Model Thinker* [57], presents the uses of models (REDCAPE): • *Reason:* To identify conditions and deduce logical implications. • *Explain:* To provide (testable) explanations for empirical phenomena. • *Design:* To choose features of institutions, policies, and rules. • *Communicate:* To relate knowledge and understandings. • *Act:* To guide policy choices and strategic actions. • *Predict:* To make numerical and categorical predictions of future and unknown phenomena. • *Explore:* To investigate possibilities and hypotheticals.
Data	How much training data is available (pre-mission and in-mission)? How well labeled is the data? Is it partially observable (i.e., can features be missing)? Is the data numeric or categorical? What features can be computed by a priori models? Has data changed since training (e.g., concept drift, component updates)?
Goals	How accurate must the solution be? How fast must the system arrive at conclusions? Do false positives and false negatives impact performance differently? Are confidence levels appropriate or necessary as annotations on decisions? Does the solution need to be interpretable or explainable to an EW operator? Does the solution need to be scalable to more emitters, more environments, or more tasks? What security/privacy considerations matter?
Constraints	What hard real-time requirements must be satisfied? What computation is available (CPU, GPU, FPGA, custom ASIC)? How much storage is available, for the model and for the data? How much of the data must be persistent over very long durations (e.g., mission-to-mission)?

While there might be a lot of data available for cellular or WiFi networks, EW domains do not have this advantage.

Figure 3.7 illustrates the performance trade-off: DeepNets perform better when there is a lot of data, while classical ML approaches do better when data is limited, in large part because features are crafted for the problem. Section 4.1.1 describes how DeepNets and SVMs might be combined effectively in a deployed system, so that DeepNets compute general features of the RF domain, while SVMs perform the in-mission model updates.

In some cases, data augmentation and adversarial training approaches (Section 8.3.3) can compensate for the lack of data [59–63]. Data augmentation increases the diversity of data, without actually collecting new data. Common data augmentation techniques in image recognition include cropping, flipping, rotating, and changing lighting schemes. In RF, passing the signal through a channel model and/or a noise model accomplishes many of the same effects.

In EW missions where the mission expects to encounter novel emitters during the mission, system design must incorporate techniques for learning novel models, from one or two examples, at mission-relevant timescales. Open-set classification (Section 3.4) discusses some approaches.

Table 3.3 summarizes a few of the design trade-offs for common ML algorithms. *This list is not comprehensive* but simply illustrates the need to analyze the task, data, goals, and constraints of the EW system before settling on one particular approach. Some general rules of thumb include [2]:

- When predicting the future, larger complex models tend to capture the performance surface more accurately. When trying to explain, smaller more concise models are better.
- When the number of training instances is limited, feature engineering can be very helpful, and classical ML approaches are more likely to be useful.

Table 3.3
Design Trade-Offs for Common ML Algorithms

ML Algorithm	Common Uses	Advantages	Disadvantages
SVMs	Stock market; RF quality; radar emitter signal recognition; modulation recognition; anomalies	Excellent accuracy on little data; extremely efficient; good for many features	Primarily numerical data
DeepNets	Image understanding; natural language; signal characteristics; modulation recognition; target tracking; anomalies; SEI	Extracts latent features in the data; inference can be very fast	Requires a lot of training data; computationally intensive; hard to explain; may suffer from long training times
Logistic regression	Risk; process failure	Efficient; interpretable; computes feature relevance	Requires clean data; one-dimensional output
Naïve Bayes classifier	Sentiment analysis; document categorization; spam filtering	Categorical input data; relatively little training data; probability of outcomes	Requires conditional independence
k-nearest neighbor	Document similarity; emitter similarity	Efficient; interpretable; no training step; instantly incorporates new data	Weaker with high-dimensional spaces; imbalance causes problems; outlier sensitivity; can be computationally intensive
Decision trees	Design decisions; sentiment analysis	Easy to explain; very strong with discrete data	Fragile with many features; weaker with numerical data
Causal models	Relationships	Patterns when experimentation is not possible	Hard to control for unobservable features

- When the number of training instances is large, and there is ample time for training, DeepNets can be effective.
- When inference-time is limited, model-based techniques are better than instance-based techniques. Model-based techniques may suffer from long training times, but inference is fast. Instance-based techniques do not have a training phase, instead incurring computation time at inference.

Lim et al. [64] evaluate 33 algorithms using metrics of training time and accuracy. A longer list of metrics appears in Callout 10.2.

3.7 Conclusion

AI is a subject area as broad as mathematics or engineering.

AI is like math. One day, it will be everywhere.

AI incorporates many subfields, covering the broader concepts of situation assessment and DM. Techniques such as planning, optimization, data fusion, and learning support application areas such as machine vision, NLP, robotics, and logistics.

The key thing to remember about AI is that ML is a concept *within* AI, and that DeepNets are a set of techniques *within* ML: AI is not equivalent to ML, and AI is not equivalent to DeepNets. ML is more than just deep learning. DeepNets have their place for EW problems, but do not neglect or ignore classical ML approaches or any of the other AI approaches simply because DeepNets currently have the broadest visibility.

References

[1] Russell, S., and P. Norvig, *Artificial Intelligence: A Modern Approach,* Pearson Education, 2015.

[2] Burkov, A., *The Hundred-Page Machine Learning Book,* Andriy Burkov, 2019.

[3] Hastie, T., R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning,* Springer, 2009.

[4] Vink, J., and G. de Haan, "Comparison of Machine Learning Techniques for Target Detection," *Artificial Intelligence Review,* Vol. 43, No. 1, 2015.

[5] Kulin, M., et al., "End-to-End Learning from Spectrum Data," *IEEE Access,* Vol. 6, 2018.

[6] Cristianini, N., and J. Shawe-Taylor, *An Introduction to Support Vector Machines and other Kernel-based Learning Methods,* Cambridge University Press, 2000.

[7] Schölkopf, B., and A. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond,* MIT Press, 2002.

[8] *Support vector machines*, Accessed: 2020-03-21. Online: https:// scikit-learn.org/stable/modules/svm.html.

[9] Üstün, B., W. Melssen, and L. Buydens, "Facilitating the Application of Support Vector Regression by Using a Universal Pearson VII Function-Based Kernel," *Chemometrics and Intelligent Laboratory Systems,* Vol. 81, No. 1, 2006.

[10] McCulloch, W., and W. Pitts, "A Logical Calculus of Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics,* Vol. 5, No. 4, 1943.

[11] Rosenblatt, F., "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," *Psychological Review,* Vol. 65, 1958.

[12] Werbos, P., *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences,* Harvard University, 1975.

[13] LeCun, Y., Y. Bengio, and G. Hinton, "Deep Learning," *Nature,* Vol. 521, 2015.

[14] Goodfellow, I., Y. Bengio, and A. Courville, *Deep Learning,* MIT Press, 2016.

[15] LeCun, Y., et al., "Gradient-Based Learning Applied to Document Recognition," *Proceedings of the IEEE,* Vol. 86, No. 11, 1998.

[16] Khan, A., et al., "A Survey of the Recent Architectures of Deep Convolutional Neural Networks," *Artificial Intelligence Review,* Vol. 53, 2020.

[17] Rumelhart, D., et al., "Schemata and Sequential Thought Processes in PDP Models," *Parallel Distributed Processing: Explorations in the Microstructures of Cognition,* Vol. 2, 1986.

[18] Miljanovic, M., "Comparative Analysis of Recurrent and Finite Impulse Response Neural Networks in Time Series Prediction," *Indian Journal of Computer Science and Engineering,* Vol. 3, No. 1, 2012.

[19] Baevski, A., et al., *wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations,* 2020. Online: https://arxiv.org/abs/2006.11477.

[20] Kingma, D., and M. Welling, "An Introduction to Variational Autoencoders," *Foundations and Trends in Machine Learning,* Vol. 12, No. 4, 2019.

[21] Chicco, D., "Siamese Neural Networks: An Overview," in *Artificial Neural Networks,* Springer, 2020.

[22] Amini, A., et al., "Deep Evidential Regression," in *NeurIPS,* 2020.

[23] Culurciello, E., *Analysis of Deep Neural Networks,* Accessed 2020-10-06, 2018. Online: https://tinyurl.com/medium-anns.

[24] Culurciello, E., *Neural Network Architectures*, Accessed 2020-10-06, 2017. Online: https:// tinyurl.com/tds-anns-2018.

[25] Canziani, A., A. Paszke, and E. Culurciello, "An Analysis of Deep Neural Network Models for Practical Applications," in *CVPR,* IEEE, 2016.

[26] Majumder, U., E. Blasch, and D. Garren, *Deep Learning for Radar and Communications Automatic Target Recognition,* Norwood, MA: Artech House, 2020.

[27] Bauer, E., and R. Kohavi, "An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants," *Machine Learning,* Vol. 36, 1999.

[28] Sagi, O., and L. Rokach, "Ensemble Learning: A Survey," *Data Mining and Knowledge Discovery,* Vol. 8, No. 4, 2018.

[29] Ardabili, S., A. Mosavi, and A. Várkonyi-Kóczy, "Advances in Machine Learning Modeling Reviewing Hybrid and Ensemble Methods," in *Engineering for Sustainable Future,* Inter-Academia, 2020.

[30] Dogan, A., and D. Birant, "A Weighted Majority Voting Ensemble Approach for Classification," in *Computer Science and Engineering,* 2019.

[31] Abbasi, M., et al., "Toward Adversarial Robustness by Diversity in an Ensemble of Specialized Deep Neural Networks," in *Canadian Conference on AI,* 2020.

[32] Tramèr, F., et al., "Ensemble Adversarial Training: Attacks and Defenses," in *ICLR,* 2018.

[33] Aha, D., "Integrating Machine Learning with Knowledge-Based Systems," in *New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems,* 1993.

[34] Mao, J., et al., "The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences from Natural Supervision," in *ICLR,* 2019.

[35] Haigh, K. Z. ,S. Varadarajan, and C. Y. Tang, "Automatic Learning-Based MANET Cross-Layer Parameter Configuration," in *Workshop on Wireless Ad hoc and Sensor Networks,* IEEE, 2006.

[36] Towell, G., and J. Shavlik, "Knowledge-Based Artificial Neural Networks," *Artificial Intelligence,* Vol. 70, No. 1, 1994.

[37] T. Yu, S. Simoff, and D. Stokes, "Incorporating Prior Domain Knowledge into a Kernel Based Feature Selection Algorithm," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining,* 2007.

[38] Muralidhar, N., et al., "Incorporating Prior Domain Knowledge into Deep Neural Networks," in *International Conference on Big Data,* 2018.

[39] Agrell, C., et al., "Pitfalls of Machine Learning for Tail Events in High Risk Environments," in *Safety and Reliability—Safe Societies in a Changing World,* 2018.

[40] d'Avila Garcez, A., et al., "Neural-Symbolic Computing: An Effective Methodology for Principled Integration of Machine Learning and Reasoning," *Journal of Applied Logics,* Vol. 6, No. 4, 2019.

[41] *Workshop Series on Neural-Symbolic Integration*, Accessed 2020-07-23. Online: http://www.neural-symbolic.org/.

[42] Haigh, K. Z., et al., "Parallel Learning and Decision Making for a Smart Embedded Communications Platform," BBN Technologies, Tech. Rep. BBN-REPORT-8579, 2015.

[43] Mattei, E., et al., "Feature Learning for Enhanced Security in the Internet of Things," in *Global Conference on Signal and Information Processing,* IEEE, 2019.

[44] Robyns, P., et al., "Physical-Layer Fingerprinting of LoRa Devices Using Supervised and Zero-Shot Learning," in *Conference on Security and Privacy in Wireless and Mobile Networks,* 2017.

[45] Tokmakov, P., Y.-X. Wang, and M. Hébert, "Learning Compositional Representations For Few-Shot Recognition," in *CVPR,* IEEE, 2019.

[46] Wang, W., et al., "A Survey of Zero-Shot Learning: Settings, Methods, and Applications," *ACM Transactions on Intelligent Systems and Technology,* Vol. 10, No. 12, 2019.

[47] Geng, C., S.-J. Huang, and S. Chen, "Recent Advances in Open Set Recognition: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 2020.

[48] Cao, A., Y. Luo, and D. Klabjan, *Open-Set Recognition with Gaussian Mixture Variational Autoencoders*, 2020. Online: https://arxiv.org/abs/2006.02003.

[49] Brazdil, P., and C. Giraud-Carrier, "Meta Learning and Algorithm Selection: Progress, State of the Art and Introduction to the 2018 Special Issue," *Machine Learning,* Vol. 107, 2018.

[50] Schaul, T., and J. Schmidhuber, *Metalearning,* Scholarpedia, 2010. doi:10.4249/scholarpedia.4650.

[51] Finn, C., P. Abbeel, and S. Levine, *Model-agnostic Meta-Learning for Fast Adaptation of Deep Networks,* 2017. Online: https://arxiv.org/ abs/1703.03400.

[52] Hospedales, T., et al., *Meta-Learning in Neural Networks: A Survey,* 2020. Online: https://arxiv.org/abs/2004.05439.

[53] Principe, J., et al., "Learning from Examples with Information Theoretic Criteria," *VLSI Signal Processing-Systems for Signal, Image, and Video Technology,* Vol. 26, 2000.

[54] Santurkar, S., et al., *How Does Batch Normalization Help Optimization?* 2019. Online: https://arxiv.org/abs/1805.11604.

[55] Barbu, A., and S.-C. Zhu, *Monte Carlo Methods,* Springer, 2020.

[56] Chaudhari, P., et al., "Entropy-SGD: Biasing Gradient Descent into Wide Valleys," in *ICLR,* 2017.

[57] Page, S., *The Many Model Thinker,* Hachette, 2018.

[58] Wolpert, D., "The Lack of A Priori Distinctions Between Learning Algorithms," *Neural Computation*, 1996.

[59] Rizk, H., A. Shokry, and M. Youssef, *Effectiveness of Data Augmentation in Cellular-Based Localization Using Deep Learning,* 2019. Online: https://arxiv.org/abs/1906.08171.

[60] Shi, Y., et al., "Deep Learning for RF Signal Classification in Unknown and Dynamic Spectrum Environments," in *International Symposium on Dynamic Spectrum Access Networks,* IEEE, 2019.

[61] Shorten, C., and T. Khoshgoftaar, "A Survey on Image Data Augmentation for Deep Learning," *Big Data,* Vol. 6, No. 60, 2019.

[62] Sinha, R., et al., "Data Augmentation Schemes for Deep Learning in an Indoor Positioning Application," *Electronics*, Vol. 8, No. 554, 2019.

[63] Xie, F., et al., "Data Augmentation for Radio Frequency Fingerprinting via Pseudo-Random Integration," *IEEE Transactions on Emerging Topics in Computational Intelligence,* Vol. 4, No. 3, 2020.

[64] Lim, T.-S., W.-Y. Loh, and Y.-S. Shih, "A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-Three Old and New Classification Algorithms," *Machine Learning,* Vol. 40, 2000.

4

Electronic Support

The first step in every cognitive EW system is ES to understand the RF spectrum. Known as situation assessment in the AI community, ES determines who is using the spectrum, where and when they are using it, and whether there are patterns that can be exploited.

This chapter discusses AI/ML techniques that can be used for feature estimation, emitter characterization and classification, data fusion, anomaly detection, and intent recognition. ES analyzes the environment and creates the observables that drive DM. The chapter is organized according to the increasing complexity implied by Figure 1.4.

Howland et al. [1] define spectrum situation awareness (SSA) as "a means of collecting disparate information about spectrum use and processing this information to produce a fused spectrum picture." SSA collects, organizes, and processes the spectrum data required for EW. In addition to pre- and post-mission analysis, SSA needs to occur in rapid real time, according to the needs of the decision maker.

To reduce brittleness and handle novel emitters and adversaries, AI and ML capabilities can improve SSA at every level. Figure 4.1 depicts a view of these AI/ML technologies, in the context of other relevant SSA technologies. A complete EW system must have multifaceted SSA. Future SSA systems can be trained with deep learning models for latent feature generation, classical ML models for in-mission-updates, and hybrid models to offset limited data (Section 4.1). Moreover, SSA does not have to solely rely on the RF data: It can be fused with non-RF data such as video and still imagery, free-space optics, or open-source, tactical, or operational intelligence (Section 4.3). Distributed data fusion across multiple heterogeneous sources must create a coherent battlespace spectrum common operating picture that is accurate in space, time, and frequency. Anomaly detection

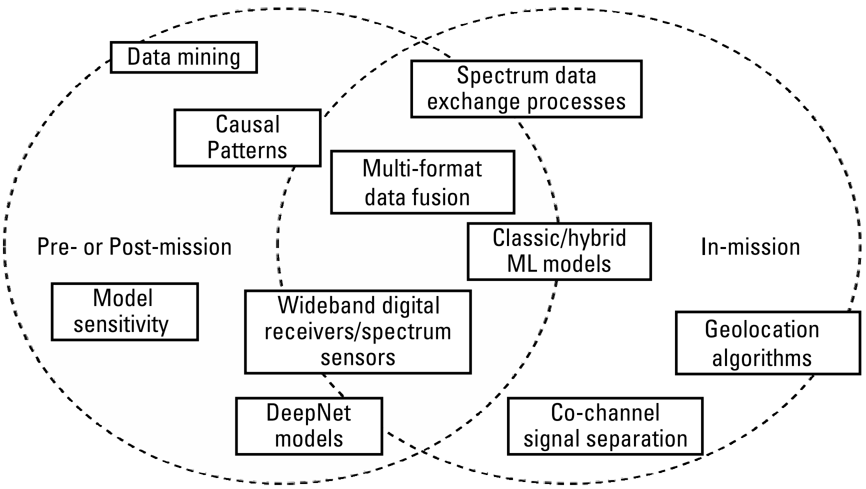


Figure 4.1 SSA must combine a variety of supporting technologies, both traditional and cognitive. One of the challenges lies in the integration and demonstration of relevant technologies, only a few of which are cognitive.

(Section 4.4), causal reasoning (Section 4.5), and intent inference (Section 4.6) complete the picture to understand the impact of events and support DM.

4.1 Emitter Classification and Characterization

A foundational component of ES is to understand what emitters are in the environment and what they are doing. Emitter *classification* identifies known categories of the signal (e.g., modulation classification or platform type). In AI, *classification* refers to *any* discrete category or label; thus the EW concept of specific emitter identification (SEI) is also a form of classification. *Characterization,* on the other hand, refers to capturing behaviors of the signal environment, without necessarily placing a named label on the behavior. One might for example recognize that a signal is loud, or is Gaussian, or repeats over a given time interval.

Characterization and classification can feed each other, in the sense that a given behavior may support a given classification, or that a given classification may indicate a need to search for a particular behavior characteristic.

A variety of challenges can impact the accuracy of emitter classification and characterization:

- Emitter characterization and long-term pattern analysis can reveal *dynamic properties,* where signal types change over time.
- *Spoofing,* wherein cognitive emitters replay signals to mask their identity, can be mitigated with RF fingerprints that capture microimperfections unique to the emitter.

- *Unknown emitters,* where there is no a priori training data, usually have known behavior characteristics (e.g., canonical modulation, frequency, and pulse parameters), but combine them in unexpected ways or use them for different purposes. Open-set classification techniques recognize and cluster these novel feature groups and construct new classes on-the-fly.

4.1.1 Feature Engineering and Behavior Characterization

Behavior characterization describes the signal environment, including *features* such as instantaneous energy, frequency, scattering, and repetition patterns and their *probabilities.* Traditionally, *feature engineering* is computed via a suite of a priori modules, corresponding to characteristics that experts know to be relevant for the expected signals. These mature signal intelligence (SIGINT) approaches can be very accurate, even at low signal-to-noise ratios (SNRs), because they rely on underlying immutable properties of the domain, such as physics of signal propagation or kinematics of the targets. However, classical signal processing approaches tend to be brittle in a complex problem domain and typically cannot incorporate novel emitters or conditions.

Traditional methods use statistical analysis of features such as pulse width, pulse repetition interval, and frequency, and map them to a database of known signals. These may be *channel-specific,* such as channel impulse response, or *transmitter-specific,* such as signal encoding. They may be *time-varying* or *time-independent.* Fractal features can provide a deeper understanding of the properties of the signal [2–4].

DeepNet approaches can identify *latent* features in the signal environment when trained with sufficient data, potentially capturing characteristics at a much higher fidelity than hand-generated computations. These approaches can capture behavior characteristics that are hard to model with explicit mathematical expressions or that vary significantly with environmental conditions. They also tend to be more effective when the domain is partially observable (i.e., when known observables o_n cannot be measured temporarily, or for unobservable features z).

Hand-generated features are a better choice when little training data is available, or when the known model closely captures the empirical performance. Features can also be pulled from non-RF data sources, such as topography [5], road networks [6], and weather conditions [7]. Section 4.3 discusses some approaches to data fusion.

Combining approaches is an effective way to accelerate learning accuracy. Radio signal strength, for example, generally follows the inverse square law with distance but is impacted by factors such as antenna gain, multipath, and fading. One way to create a hybrid model in RF uses both traditional features and DeepNets. The DeepNets would learn the latent features, while the traditional features would include physics, human expertise, and other metadata. Figure 4.2

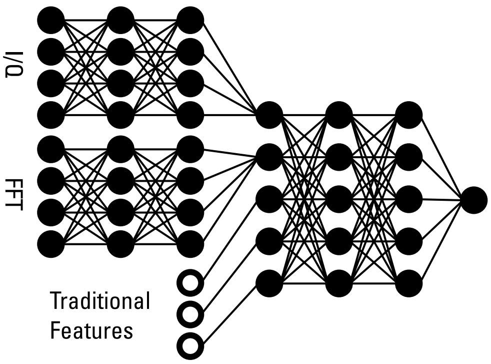


Figure 4.2 A possible hybrid architecture uses independent DeepNets to analyze I/Q data and FFTs, then hierarchically combines the generated features with the traditional features.

illustrates a concept of how the overall network could be constructed, combining I/Q data with a fast Fourier transform (FFT) and traditional features.

Feature engineering should focus on those that lead to better assessment of the environment and thus actionable inferences.

For example, features that indicate a loss of adversary performance include (1) reduced comms network throughput that suggests changing to a more robust modulation technique, (e.g., switch from QAM64 to QPSK), or (2) in the radar space a loss of track indicated by cessation of track waveforms or detection of acquisition waveforms suggests increasing the beam revisit rate or changing the waveform (e.g., longer pulse widths).

An interesting characterization problem determines the degree of sophistication of observed emitters [8–10]. Extended observation of the emitter computes its adaptivity and behavior model, thus supporting inference of its strategy and probable next actions (Section 4.6).

4.1.2 Waveform Classification

One of the most popular current uses for ML techniques is in the area of waveform classification. Older work frequently used SVMs for radar emitter signal recognition [11, 12], radar antenna scan type recognition for EW applications [13], and automatic digital modulation recognition [14, 15]. Newer work is migrating to using deep networks, largely because of their ability to identify latent features of the RF signals [16, 17]. Examples include classifying radio emitters using CNNs [18–20], radar emitters using CNNs [21], and RNNs to capture temporal patterns [22]. GANs are effective methods for augmenting the data, so that a model can be trained with fewer examples [23, 24].

In 2018, the U.S. Army Rapid Capabilities Office (RCO)[1] conducted a Signal Classification Challenge to promote advanced research to explore deep learning algorithms for blind signal classification and characterization [25]. While accuracies demonstrated by this challenge were not ready for fielding, research exploded, and perhaps more importantly, the importance of data quality and diversity has been recognized by the community. Hybrid ML approaches may solve some of the open problems. Figure 4.2 illustrates one possible hybrid architecture; Section 7.3.3 describes a different structure appropriate when a priori training data is abundant, but the system needs to rapidly update during a mission. Notably, the winning team used a hybrid approach, combining traditional features, ensemble learning, and DeepNets [26, 27].

Algorithm 4.1 outlines the basic steps required to build a radar class detector. Create a set of *emitter descriptor words* (EDWs) from ground truth–observed parameters including the duty cycle, the frequency, the pulse repetition interval, and the pulse width. Assign each EDW its associated class. This example trains the naïve Bayes algorithm, which computes the probability of each possible class. Using Bayes' rule, the posterior probability $P(y|x)$ is the probability that the radar whose EDW is x is actually a member of the class y: $P(y|x) = P(x|y) \times P(y) \div P(x)$ where $P(x|y)$ is the likelihood of the EDW given the class, $P(y)$ is the prior probability of the class, and $P(x)$ is the prior probability of the EDW. The train/test cycle is the same for every model type, and other ML models may yield more accurate results.

Figure 4.3 illustrates where these steps function during design and operation. Using the iterative validation process of Section 10.4, the design phase creates the data formats, develops the model, and evaluates its performance. Always experiment with multiple data formats (and features) and types of models to determine which is most effective for the problem: accuracy, data, goals, and constraints (Section 3.6). When it meets requirements, the model is used to classify new radars during operation. Figure 4.4 illustrates where the classifier belongs in the RF processing chain.

4.1.3 Specific Emitter Identification

SEI, or RF fingerprinting, is the process of uniquely identifying each RF emitter, independent of receiver properties or how the transmitter changes its transmissions, its mobility, or environmental factors. Xu et al. [28] provide an overview of the challenges and opportunities of this classification problem.

Chen et al. [29] use clustering and infinite hidden Markov random fields to handle both time-dependent and time-independent features. Nguyen et al. [30]

1. The Army RCO has since been renamed the Army Rapid Capabilities and Critical Technologies Office (RCCTO).

Algorithm 4.1 To build a radar class detector, train a model on some of the data, and validate the model on the rest of the data. Every ML model is trained and tested with the same process (Figure 3.2).

Step 1: Load the data set of known radars. Split the data set into a *train* and *test* set (approximately 80%/20%).[*]

```
from sklearn.model_selection import train_test_split
radars = pandas.read_csv( 'radar-classes.csv' )
X = radars.drop(columns=['Class'])
y = radars[ 'Class' ]
( xTrain,xTest ),
( yTrain,yTest ) = train_test_split( X, y,
                                    test_size=0.20)
```

Step 2: Train the algorithm. Select a classification algorithm, for example naïve Bayes, and train the model on the training data.

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
trainedGNB = gnb.fit( xTrain, yTrain )
```

Step 3: Validate the results. Evaluate the model with the test data. Compare the predicted classes against the known test classes using a confusion matrix (Section 10.3.2).

```
from sklearn.metrics import confusion_matrix
yPred = trainedGNB.predict( xTest )
accuracy = (yTest==yPred).sum() / len(yTest) * 100
confMtx = confusion_matrix( yTest, yPred )
```

Step 4: Classify new radars and estimate class probabilities. In this snippet, we assume no '`Class`' column. Open-set classification (Section 3.4) and Ablation trials (Section 10.2) address this concept.

```
newRadars = pandas.read_csv( 'newRadars.csv' )
yPred = trainedGNB.predict( newRadars )
yProbs = trainedGNB.predict_proba( newRadars )
```

[*] Most ML algorithms expect no NaN-values in the data. We therefore replace NaNs for each feature with the non-NaN mean of that feature (Algorithm 2.1). When replacing NaNs before training a ML model, do not replace NaNs in the complete dataset (i.e., before splitting it for train and test data). Instead, first split the data, and use only the training data to compute the means.

use Gaussian mixture models (GMMs) to cluster and then classify signals. Cain et al. [31] use CNNs on image-based representations of the signal.

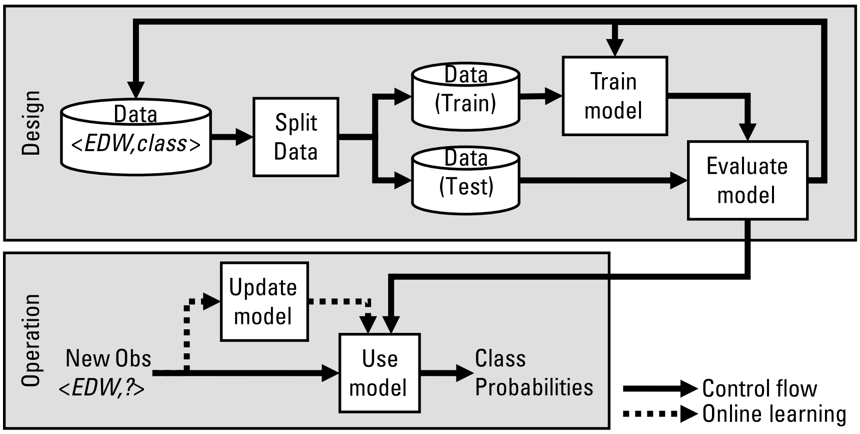


Figure 4.3 Each of the steps in Algorithm 4.1 must be evaluated during the design phase before moving to operation.

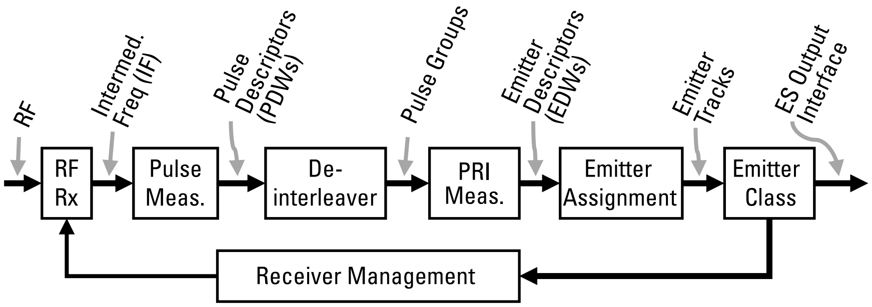


Figure 4.4 The radar classifier is the last step of the common RF processing chain. (Other AI techniques could use the information for further downstream analysis.)

A lot of current work feeds I/Q samples to a CNN, moving away from hand-generated features. These models do not perform channel estimation and do not assume prior knowledge of the system. Examples include feeding the I/Q samples to the CNN [32–35], adding a finite input response (FIR) [36], and using a wavelet transform to handle temporal patterns [33].

Existing work has not yet achieved the accuracy of traditional approaches for highly dense, complex real-world environments, particularly for low signal strengths. Hybrid approaches will likely yield interesting results in the future.

4.2 Performance Estimation

One can also use ML techniques to capture the relationship between observations o and performance metrics m, usually as regression models: $m = f(o,s)$. Section 2.3

presents possible metrics, organized by effectiveness and cost. Example techniques include SVMs [37], ANNs [38–43], Bayesian networks [44], and Markov decision processes (MDPs) [45, 46]. Because these approaches measure system performance, many of them are incorporated into RL frameworks, where the learned model is then used to control the decision process (see Chapter 7).

Performance estimation learns the effectiveness of each technique, using the set of observables that describe the jammer. Figure 4.5 presents example results from estimating the performance of different controllables against different RF environments. Only the points marked with a dot are known a priori; the ML model estimates other performance values based on observable features of the different environments. This example uses an SVM to model the performance space and traditional features to describe the environments. Each observation of an environment generates a corresponding set of observables *o*. Each use of a strategy *s* in that environment creates feedback of the metric m_k [i.e., an example of $m_k = f_k(o,c)$] and marked with a dot on the graphic. The trained model then estimates the performance of all strategies against all environments, generalizing from the known cases to other conditions. Figure 7.5 extends this example to show how the model can incrementally learn during a mission.

As a concrete example, Table 5.1 outlines several techniques to mitigate a tone jammer. The learner can model both the associated benefits and associated costs. In this case, BER is a good metric to evaluate performance.

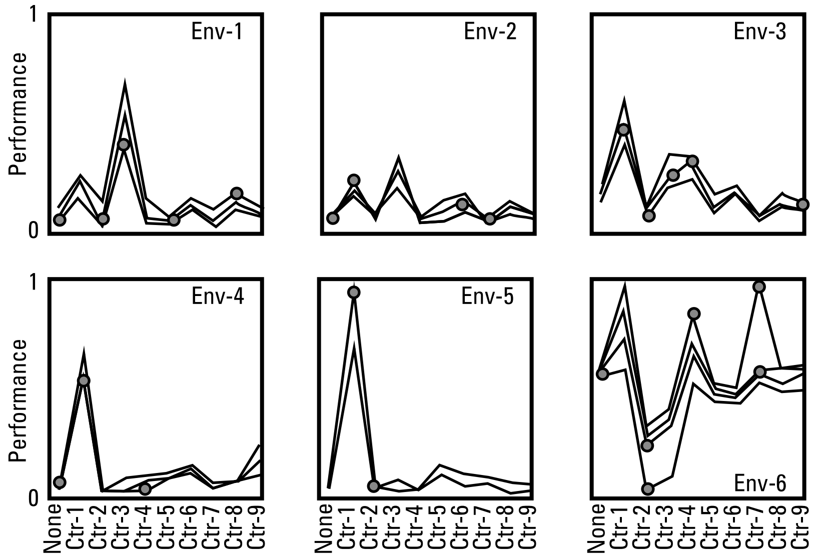


Figure 4.5 The ML model f_k learns to estimate the performance m_k of different controllables *c* against different environments. (Real data from Example 7.1.) Dots represent observations; all other points are estimated by the model. Figure 8.5 shows plots like these in a 3D layout.

ML can also predict performance in terms of jamming effectiveness [47], learning which EA technique to use for which observed emitter behaviors. Performance feedback on these predictions comes from EW BDA (Section 7.1.1).

4.3 Multi-Intelligence Data Fusion

For optimal situational awareness, decision-makers need to interpret data from multiple sources (e.g., radars, unmanned aerial, ground, or underwater systems, ships, space assets, fighter jets, antennas, and sensor networks). To support this capability, a cognitive EW system must perform data fusion.

Data fusion is the process of data integration from multiple sources to produce more accurate inferences. Multi-intelligence (multi-INT) data fusion compares, correlates, and combines data from multiple sources of different types to achieve improved accuracy and more specific inferences than those that can be achieved by a single sensor alone [48–50]. Accurate data fusion is a hard task to accomplish due to a variety of challenges [51, 52].

In all cases, metadata tagging (Section 8.1.1) tracks he upstream sources of data, allowing downstream consumer modules to compensate for error.

Data fusion challenges include:

- Data quality issues, such as measurement uncertainty, sensor calibration, and bias can skew accuracy and confidence. Data curation (Section 8.2) helps mitigate this issue.
- Conflicting data can produce counterintuitive results. Traceability (Section 8.1.3) and managing information uncertainty (Section 6.1.4) facilitate correct decisions [53].
- Data association connects the dots between concepts, and if overeager, can produce ridiculous results. Causal modeling (Section 4.5) techniques help identify correct relationships.
- High-dimensional data makes it difficult to find good items to fuse, akin to looking for the needle in the haystack. Section 5.1.3 describes some techniques for dimensionality reduction.
- Operational timing can cause relevant information to be delayed or arrive out of sequence. Metadata is *crucial* here: Time is probably the most important feature to record faithfully so the fusion engine doesn't fuse inconsistent items. Section 4.3.3 delves more into this issue.

Data fusion techniques can be grouped into three nonexclusive categories: (1) data association, (2) state estimation, and (3) decision fusion [52]. The Joint Directors of Laboratories/Data Fusion Information Group (JDL/DFIG) model comprises seven levels of data fusion shown in Figure 4.6 [54, 55]. Each level increases the breadth of data and scope of analysis. Some of the JDL/DFIG model's benefits include exploration of situation awareness, user refinement, and mission management [54]. The JDL/DFIG model has also been successfully used for the data fusion process visualization, discussion facilitation and common understanding [55], and systems-level data fusion design [54, 56]. The latest advances in the data fusion field are captured by the International Society of Information Fusion (ISIF) in [57].

4.3.1 Data Fusion Approaches

The book *Sensor and Data Fusion* [58] presents a comprehensive compilation of research papers on various topics related to sensor and data fusion. Some of the addressed problems include target tracking, air traffic control, remote sensing, anomaly detection and behavior prediction, and sensor networks.

Data fusion techniques have been widely used in multisensor environments with the objective of amalgamating data from different sensors. The advantages of multisensor data fusion predominately include enhancements in data authenticity, reliability, and availability [49, 51, 52, 59–62]. Examples of data authenticity enhancements consist of improved detection, confidence, reliability, and reduction in data ambiguity. Data availability enhancements focus on extending spatial and temporal multisensor coverage.

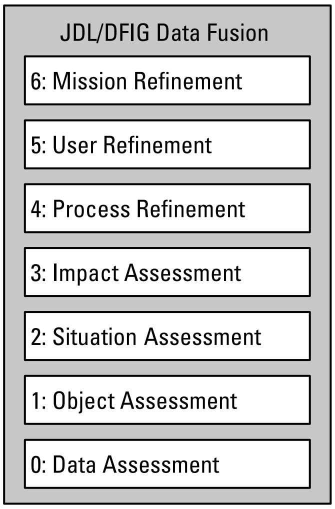


Figure 4.6 The JDL/DFIG model comprises seven levels of data fusion.

Data fusion helps solve scalability issues in wireless sensor networks (WSNs) that are caused by a large number of sensor nodes, packet collisions, and redundant data transmissions. When the routing process performs data fusion (i.e., sensor data is merged and only the result is forwarded), the total number of messages is reduced, collisions are avoided, and energy reduced [49, 51, 59]. WSN nodes typically rely on battery power for energy that can be consumed by computation, communication, sensing, and idling states [63].

Some interesting applications of multi-INT/multisensor data techniques in EW pull features from other types of data sources. Brandfass et al. [5] develop an application that adds map data to radar. Katsilieris and Charlish [6] use road networks for ground moving target indicator. Zhang et al. use weather data to improve path loss predictions [7]. Levashova et al. [64] present an approach to data fusion to predict terrorist threat activity for applications such as area surveillance, command and control, communications, and access rights.

4.3.2 Example: 5G Multi-INT Data Fusion for Localization

One concrete example of how ML can assist with the multi-INT data fusion is 5G localization and the concept of *context awareness*.[2] Context-aware mobile devices such as smart phones may become aware of their location or attempt to make assumptions about the end user's current situation. Table 4.1 shows 5G's location-based key performance indicators (KPIs) [65, 66] and some related use cases.

The complex localization KPI requirements in 5G scenarios can be fulfilled by exploiting different ML-enabled data-fusion techniques and measurements summarized in Table 4.2 [66–71]. The reliability of 5G localization also requires efficient network strategies such as node prioritization, node activation, and node deployment. Cooperative operation, robustness guarantees, and distributed design of network operation are also important in localization, because they affect energy consumption and determine localization accuracy [72].

One of the novel localization approaches for 5G is soft information (SI)-based localization from heterogeneous sensors and contextual information. Conventional localization methods rely on *single-value estimates* (SVEs) such as time of arrival (TOA), observed-time difference-of-arrival (OTDOA), angle of arrival (AoA), or received-signal strength indicator (RSSI). Novel techniques, however, rely on a *set of possible values* rather than on a single-distance estimates, for example, soft range information [67]. ML-enabled data-fusion approaches include data association, state estimation, decision fusion, classification, prediction/regression, unsupervised ML, dimension reduction, and statistical inference and analytics [68].

2. 5G is a set of fifth-generation commercial cellular standards developed by the 3rd Generation Partnership Project (3GPP) standardization body.

Table 4.1
5G's Location-Based KPIs and Some Related Use Cases

KPI	Description
Position accuracy (PA)	Difference between the user equipment (UE) estimated position and its true position
Speed accuracy (SpA)	Difference between the estimated magnitude of the UE's velocity to the true magnitude
Bearing accuracy (BA)	Difference between the UE measured bearing and its true bearing
Latency (L)	Time elapsed between the event that triggers the determination of the position-related data and its availability at the positioning system interface
Time to first fix (TTFF)	Time elapsed between the event triggering the determination of the position-related data and its availability at the positioning system interface
Update rate (UR)	Rate at which the position-related data is generated by localization
Power consumption (PC)	Electrical power (typically in milliwatts) used by localization to produce the position-related data
Energy per fix (EPF)	Electrical energy (typically in millijoules per fix) used by localization to produce the position-related data
System scalability (SS)	Number of devices for which the positioning system can determine the position-related data in a given time unit, and/or for a specific update rate

Using these KPIs, 3GPP outlined several use cases and their potential requirements, listed as follows:

- *First responders indoor:* PA: 1-m horizontal, 2-m vertical; availability: 95%; TTFF: 10s; L: 1s.
- *Augmented reality outdoor:* PA: 1–3-m horizontal, 0.1–3-m vertical; velocity: 2 m/s; 10 degrees; availability: 80%; TTFF: 10s; L: 1s; UR: 0.1–1s; PC: low energy.
- *Traffic monitoring and control outdoor:* PA: 1–3-m horizontal, 2.5-m vertical; availability: 95%; UR: 0.1s; TTFF: 10s; L: 30 msec; antispoofing; antitampering.
- *Asset tracking and management outdoor:* PA: 10–30-m horizontal; 5m/s; availability: 99%; UR: 300s–1 day; antispoofing; antitampering; out of coverage; EPF=20 mJ/fix.
- *UAV:* (Data analysis) Outdoor PA: 0.1-m horizontal, 0.1-m vertical; availability: 99%; TTFF: 10s; low energy; antispoofing; antitampering.

4.3.3 Distributed Data Fusion

Distributed data fusion is the process of intelligent agents (1) sensing their local environment, (2) communicating with other agents, and (3) collectively trying to infer knowledge about a particular process [73]. Distributed data fusion is an important area to consider as it must respect many key constraints imposed by the EW domain, particularly limited communication. Lang [74] surveys work that addresses collective DM with incomplete knowledge. Makarenko et al. [75] examine distributed data fusion as a distributed inference problem and apply Shafer-Shenoy and Hugin junction tree algorithms to target tracking. Decentralized estimation increases robustness against node and link failures [76]. The book collection edited by Hall et al. [77] describes a variety of data-fusion approaches for distributed network-centric operations.

Table 4.2
ML-Enabled Radio Access Technology (RAT)–Dependent and RAT-Independent Data-Fusion Techniques and Measurements

RAT-Dependent	RAT-Independent
4-4.5G: Cell-ID, enhanced-CID, OTDOA, uplink TDA, radio-frequency pattern matching [66]	*Traditional:* GNSS, radio-frequency identification (RFID), terrestrial beacon systems, Bluetooth, wireless local area network, sensors (e.g., RF, acoustic, infrared, radar, laser, inertial, vision tech, and barometric), ultra-wideband [66]
Novel: multipath-assisted localization [69], single-anchor localization with massive millimeter-wave antenna arrays [70], cooperative localization (e.g., device-to-device and vehicle-to-everything) [71]	*Novel:* Distributed soft information-based localization from heterogeneous sensors plus context [67], such as digital maps, dynamic channel models, geographic information system data, and real-time traffic data

High-quality metadata (Section 8.1.1) is a key requirement of distributed data fusion. The challenge is not accuracy in spectrum per se, but rather the challenge of taking multiple signals across spectrum collected on multiple nodes, and aggregating them into a single consistent view of the observed emitters. This step requires consistent signal interpretation and emitter behavior aggregation via sharing and a very consistent behavior representation. Additionally, provenance and credibility (Section 8.1.3) notably impact data transformation, inferences made, and downstream modules' functionalities.

Collectively, distributed data fusion may change the overall sensitivity of the data (e.g., temperature at a singular sensor X may not be sensitive, but the results from the sensor array are).

Distributed analysis is a powerful way to reduce data ambiguity and increase accuracy of slower-time conclusions.

4.4 Anomaly Detection

Anomaly or outlier detection aims to find patterns in data that do not follow known behavior [78]. The statistics community has been studying anomaly detection in data as early as the 19th century [79], and many anomaly-detection techniques exist.

Figure 4.7 provides a simple example of anomalies in a two-dimensional data set; previously seen well-behaved data is depicted by black circles, while anomalies are shown as triangles. Anomalies in RF can be due to factors such as spurious noise, measurement error, and new observed behaviors.

Chandola et al. [78] distinguish between simple anomalies and complex anomalies. *Simple* or *point* anomalies address individual data instances that are anomalous with respect to the rest of the data (e.g., a radar deliberately using a

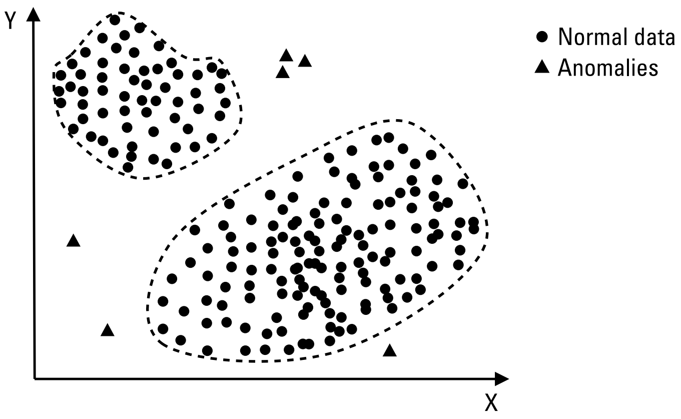


Figure 4.7 Anomaly detection aims to find data that does not follow conventional behavior.

different frequency or making a spurious error). The majority of anomaly-detection research focuses on point anomalies.

Complex anomalies can further be subdivided into contextual and collective anomalies. A *contextual* or conditional anomaly is an anomalous data instance in a specific context, but not otherwise [80]. A *collective* anomaly is an anomalous collection of related data instances with respect to the entire data set. The individual data instances in a collective anomaly may not be anomalies by themselves, but their combined occurrence is anomalous. For example, when a radar uses a particular pulse burst [e.g., defined parametrics, RF, or pulse-descriptor word (PDW)] with a specific scan pattern (say, box search) ordinarily used for target acquisition, and instead uses it for tracking.

Anomaly-detection techniques rely on ML techniques such as classification, nearest neighbor, clustering, statistics, information theory, and spectral theory [78, 81–88]. Common classification-based anomaly-detection techniques often involve SVMs, Bayesian networks, rule-based methodologies, and DeepNets, particularly autoencoders.

Anomaly detection is commonly employed for network intrusion-detection and categorization using various algorithms including: feed-forward ANNs [89], RNNs [90], autoencoders [91], SVMs [92, 93], supervised ML techniques such as linear regression and random forest for multicloud environments [94], and restricted Boltzmann machines [95].

An interesting use for anomaly detection at runtime is detecting evolving normal data where the current (normal) state is not sufficiently representative of the future state. This idea is known as *concept drift* in the AI literature, because the current concept of normal has drifted away from the concept at the beginning of the mission. One way to handle concept drift is to estimate the properties of statistical distribution models that capture the range of observations. When the distribution(s) begin to separate with a weight (support) of "anomalous" signals then the distribution models can be split to capture the statistics of all modes observed. This idea lies behind much of the open-set classification research (Section 3.4). In this way one does not lose the history of known behavior and get distracted by the new anomalies or fail to adapt to new behavior. It is not a precise solution since there are trade-offs on when to spawn a new distributional model component, but having a persistent memory as well as mechanisms to learn new behavior is critical.

4.5 Causal Relationships

Causal inference attempts to extract causal relationships from the observed data, typically when there is no explicit way to perform an experiment to validate a hypothesis [96, 97]. The causal hierarchy groups models according to the kinds of questions that each layer is capable of answering, described as follows.

- *Association.* Observing. How does my belief in a concept y depend on an observation x: $P(y|x)$?
- *Intervention.* Acting. How does doing an action a impact the outcome y: $P(y|\mathrm{do}(a))$
- *Counterfactuals.* Imagining. What would have happened under other circumstances: $P(y|x')$

Causal models are commonly used for analysis of medical, warfare, and other social settings. Effective for cybersecurity [98], they have also been used for radar and air traffic safety [99]. The goal is to detect sequences of events that lead to likely outcomes, such as a precursor "red flag" of imminent danger. In RF, Martin and Chang [100] use *probabilistic structural causal models* (SCMs) and multiattribute utility theory for *dynamic spectrum access* (DSA) situational awareness and decision-making. The DSA causal inference system develops and maintains situational awareness based on spectral occupancy, spectrum user location estimates, and path loss characterizations. The authors derive the relationship between SA uncertainty and DSA system performance [101, 102] yielding path loss estimates that enable DSA systems to operate with acceptable levels of risk.

Other techniques that analyze causal relationships include statistical patterns [103], Granger causality [104, 105], hysteresis [106], precursor detection [107], n-gram analysis to determine common phrases, and hidden Markov models (HMMs) to determine likely transitions [9]. Intent recognition (Section 4.6) takes the pattern analysis "up" one level, determining full grammars and hierarchical structure.

4.6 Intent Recognition

Plan recognition, activity recognition, and intent recognition all make inferences about another actor by observing their interactions with the environment and each other [108, 109]. In a cognitive EW system, such techniques can be applied to observe and understand the plans, activities, and goals of all the participants in the theater, including adversaries, coalition partners, and human EWOs. Such inference actually requires three separate tasks.

First, *activity recognition* identifies the lowest-level actions being executed by an agent. As such, it focuses on the problem of processing raw, noisy, frequently continuous data into high-confidence discrete events (often with much longer duration). For example, signal-processing algorithms remove noise from a continuous stream of sensor values to provide a clear unambiguous identification of the event that is happening.

Second, *goal recognition* identifies the highest-level desired end or goal states of the observed agent. Such a process asks at a high level what the agent is trying to accomplish. In the abstract, activity and goal recognition can be seen as the same problem and may be amenable to similar algorithms. That is, both of these problems are instances of labeling: Given a sequence of observations, both of these tasks produce a single label to identify either the activity being performed or the goal being achieved.

Finally, and in contrast, rather than producing a single label for a sequence of activities, *plan recognition* produces a structured (hierarchical) representation that relates observed activities to the desired goal states of the agent [110]. Thus, a plan recognition system takes as input the sequences of events identified by activity recognition. It organizes the sequence of observations into structured representations that capture the causal relations for the particular sequence of recognized activities and their final objective. This inference of the final objective means goal recognition is frequently performed as a byproduct of plan recognition. This process also frequently requires not only capturing the relations between the observed activities, but also hypothesizing future activities that the agent will have to undertake if the algorithm is correct in identifying the actor's plan and goal. As such, it can enable prediction of future actions at multiple levels of abstraction. These predictions can then be used as a comparison feature for EW BDA (Section 7.1.1).

Adversarial intent recognition creates a challenge for plan recognition algorithms because the actor(s) attempt to hide or disguise their actions or deliberately change their patterns of activity to make recognition harder. However, given a model of what the adversary can do, the process is no different for adversarial actions than for benign actions. The algorithm must be agnostic about whether an action is benign or deceptive until an additional action differentiates the two.

Algorithms for plan recognition fall into the following two categories [111]:

- Plan recognition as *planning* describes observed actions/activities as functions from states to states and captures the changes in the world state caused by their execution. As such, plan recognition is a world-state-based search for a sequence of actions that will provably carry the actor from a known initial state of the world to a goal state of the world.
- Plan recognition as *parsing* views the problem as a search in the space of possible plans. Such systems explicitly represent the set of possible plans (frequently using a formal grammar similar to those used in NLP) and then search this space for the plan that is most consistent with the input observations. This process constructs hierarchical structures to capture the plans.

As an example, the *probabilistic hostile agent task tracker (PHATT)* [112] used a parsing-based approach for a number of diverse domains including network intrusion detection [113] and insider threat/misuse detection. To represent the plans to be recognized, it used a probabilistic context-free tree grammar most closely associated with *Greibach normal form (GNF)* grammars. GNFs are commonly used in compilers for programming languages. While PHATT was very effective, its wide applicability was limited by its computational costs. It made early commitments to high-level goals and required a complete search of the space of consistent plans. For example, imagine observing an airplane take off; PHATT would immediately consider all possible flight paths to all possible destinations, no matter how wildly unlikely. This problem prevented its scaling to large, real-world deployment.

Geib's more recent work on the LEXrec component of the *Engine for Lexicalized Intent Reasoning (ELEXIR)* [114] addressed this limitation. Instead of tree grammars, LEXrec uses probabilistic *combinatory categorial grammars (CCGs)* from state-of-the-art NLP research. LEXrec supports recognizing multiple concurrent and interleaved plans, partially ordered plans, and partial observability in the domain.

In addition, LEXrec leverages Monte-Carlo tree search—the algorithm behind AlphaGo [115]—to perform a heuristic search of the space of possible plans in an "anytime" fashion (Section 5.3). This approach has allowed LEXrec to process thousands of observations in minutes in the task of recognizing grey-zone operations. Further, the grammars for LEXrec can be configured to further accelerate processing and are thus promising for large-scale deployment on real-world problems. LEXrec's search can be modified so that more likely explanations for the observations are searched first. In such a case, high-probability explanations that include hostile plans will be explored before lower-probability benign explanations.

Similar grammar-based systems have been successfully applied to the multifunction radar problem [116, 117]. Further, the plan recognition community has developed both supervised and unsupervised learning techniques to learn new grammars (potential plans) based on sequences of observations. While this work is still in its initial stages, there is evidence that it is possible to learn more accurate grammars than those built by hand.

A system like LEXrec could take as input a large number of categorized radar tracks and observations from diverse and distributed multifunction radars. On the basis of these observations, and a grammar capturing both the potential individual and joint plans that the targets could be following, the system could build a set of hypothesized plans and goals (both individual and team) and establish conditional probabilities over these possible plans. Such plans could capture complex radar modes and temporally extended plans. Further, the system could predict the future course of the targets, not just on the basis of their immediate trajectory, but based on a higher-level understanding of the target's goals.

Recent work has started to incorporate game-theoretic approaches to recognize plans where the observed agent actively tries to avoid detection [118–120]. The *adversarial plan recognition* task consists of inferring a probability distribution over the set of possible adversarial plans, and a *strategy formulation* task selects the most appropriate response by computing a Nash equilibrium. Section 6.2 discusses other uses for game theory.

4.6.1 Automatic Target Recognition and Tracking

One common intent recognition task in EW is that of target tracking. ML techniques have been applied to the target recognition and tracking task since at least the early 1990s (e.g., [121–124]). Many of the techniques popular in the computer vision community have been successfully adopted by the RF community. Techniques demonstrated in RF include knowledge-based approaches and strictly empirical approaches.

Knowledge-based approaches build on prior distributions and knowledge-aided models from similar environments to perform tracking. Xiang et al. [125] use recursive Bayesian state estimation that iterates between choosing the targets to track and acquiring the track. Gunturken [126] starts with a weak target estimation and refines it in two adaptive filtering stages.

Empirical ML approaches use only environment observations to build a model of the environment. GMMs, for example, are an effective method to identify ground surveillance radar targets, and can outperform trained human operators [127]. GMMs also perform well at assigning target measurements to multiple existing tracks in radars [128, 129]. Schmidlin et al. [124] presented an early ANN approach for multiple target radar tracking problems. In this work, one ANN recognizes trajectories and delivers track predictions, while a second RNN associates plots to tracks. Compared to Kalman filters, the ANN approach is more accurate, while Kalman filters are more robust to noise. CNNs have been very effective for object detection in radar and SAR images [17]. The Visual Object Tracking challenge of 2019 had no trackers that relied on hand-generated features, which the organizers state is a "stark contrast to VOT2018" [130].

CNNs and SVMs have also been successfully combined [131–133]: The CNNs are trained and then SVMs replace the final classification layer, similar to Figure 7.6. The combination takes advantage of the CNN's ability to learn features, and the SVM's ability to generalize well for high-dimensional input spaces. The method achieves very high accuracy.

Combining traditional techniques including hand-generated features, knowledge-based approaches, and empirical methods in hybrid approaches improves the learning rate and overall effectiveness of the models. Multiple techniques often provide complementary capabilities [130, 134–137]. Some recent work augments the tracking methods with reinforcement learning to create systems with both perception and action, using for example MDPs [45], or deep Q-learning [138]. Xiang et al. [125] combine recursive Bayesian state estimation to track targets with optimization over dynamic graphical models to select the optimal subset of radars, waveforms, and locations for the next tracking instant.

4.7 Conclusion

ES is at the heart of every effective EW system. EP and EA performance is directly tied to the quality of the ES. ES analyzes the environment and creates the observables that drive DM. This chapter introduced a sampling of SA techniques that address some of the particular challenges of an ES system.

Classical ML techniques have had a presence in EW systems for years. ML is exploding with new insights and results, driven primarily by developments in the deep learning community. Deep learning has achieved remarkable breakthroughs in natural language and image processing, largely due to three factors: bigger computers, labeled data, and insights into how to connect the networks. The RF community can exploit these advances to address similar problems in ES.

Embracing deep learning, however, does not mean that classical ML techniques or expert knowledge of the problem should be dropped. Use the approach most suited for the task at hand that meets system goals and requirements.

References

[1] Howland, P., S. Farquhar, and B. Madahar, "Spectrum Situational Awareness Capability: The Military Need and Potential Implementation Issues," Defence Science and Technology Lab Malvern (United Kingdom), Tech. Rep., 2006.

[2] Aghababaee, H., J. Amini, and Y. Tzeng, "Improving Change Detection Methods of SAR Images Using Fractals," *Scientia Iranica,* Vol. 20, No. 1, 2013.

[3] Dudczyk, J., "Specific Emitter Identification Based on Fractal Features," in *Fractal Analysis–Applications in Physics, Engineering and Technology,* IntechOpen Limited, 2017.

[4] L. Shen, Y.-S. Han, and S. Wang, "Research on Fractal Feature Extraction of Radar Signal Based On Wavelet Transform," in *Advances in Intelligent Systems and Interactive Applications,* 2017.

[5] Brandfass, M., et al., "Towards Cognitive Radar via Knowledge Aided Processing for Airborne and Ground Based Radar Applications," in *International Radar Symposium,* 2019.

[6] Katsilieris, F., and A. Charlish, "Knowledge Based Anomaly Detection for Ground Moving Targets," in *Radar Conference,* IEEE, 2018.

[7] Zhang, Y., et al., "Path Loss Prediction Based on Machine Learning: Principle, Method, and Data Expansion," *Applied Sciences,* 2019.

[8] Krishnamurthy, V., *Adversarial Radar Inference. From Inverse Tracking to Inverse Reinforcement Learning of Cognitive Radar,* 2020. Online: https://arxiv.org/abs/2002.10910.

[9] Haigh, K. Z., et al., "Modeling RF Interference Performance," in *Collaborative Electronic Warfare Symposium,* 2014.

[10] Topal, O., S. Gecgel, and E. Eksioglu, *Identification of Smart Jammers: Learning Based Approaches Using Wavelet Representation,* 2019. Online: https://arxiv.org/abs/1901.09424.

[11] Zhang, G., W. Jin, and L. Hu, "Radar Emitter Signal Recognition Based on Support Vector Machines," in *Control, Automation, Robotics and Vision Conference,* IEEE, Vol. 2, 2004.

[12] Zhang, G., H. Rong, and W. Jin, "Application of Support Vector Machine to Radar Emitter Signal Recognition," *Journal of Southwest Jiaotong University,* Vol. 41, No. 1, 2006.

[13] Barshan, B., and B. Eravci, "Automatic Radar Antenna Scan Type Recognition in Electronic Warfare," *Transactions on Aerospace and Electronic Systems,* Vol. 48, No. 4, 2012.

[14] Mustafa, H., and M. Doroslovacki, "Digital Modulation Recognition Using Support Vector Machine Classifier," in *Asilomar Conference on Signals, Systems and Computers,* IEEE, Vol. 2, 2004.

[15] Park, C.-S., et al., "Automatic Modulation Recognition Using Support Vector Machine in Software Radio Applications," in *International Conference on Advanced Communication Technology,* IEEE, Vol. 1, 2007.

[16] Li, X., et al., "Deep Learning Driven Wireless Communications and Mobile Computing," *Wireless Communications and Mobile Computing,* 2019.

[17] Majumder, U., E. Blasch, and D. Garren, *Deep Learning for Radar and Communications Automatic Target Recognition,* Norwood, MA: Artech House, 2020.

[18] O'Shea, T., J. Corgan, and T. Clancy, "Convolutional Radio Modulation Recognition Networks," in *International Conference on Engineering Applications of Neural Networks,* Springer, 2016.

[19] O'Shea, T., T. Roy, and T. Clancy, "Over-the-Air Deep Learning Based Radio Signal Classification," *IEEE Journal of Selected Topics in Signal Processing,* Vol. 12, No. 1, 2018.

[20] Shi, Y., et al., "Deep Learning for RF Signal Classification in Unknown and Dynamic Spectrum Environments," in *International Symposium on Dynamic Spectrum Access Networks*, IEEE, 2019.

[21] Sun, J., et al., "Radar Emitter Classification Based on Unidimensional Convolutional Neural Network," *IET Radar, Sonar and Navigation,* Vol. 12, No. 8, 2018.

[22] Notaro, P., et al., *Radar Emitter Classification with Attribute-Specific Recurrent Neural Networks,* 2019. Online: https://arxiv.org/abs/1911.07683.

[23] Li, M., et al., "Generative Adversarial Networks-Based Semi-supervised Automatic Modulation Recognition for Cognitive Radio Networks," *Sensors*, 2018.

[24] Shi, Y., K. Davaslioglu, and Y. Sagduyu, *Generative Adversarial Network for Wireless Signal Spoofing,* 2019. Online: https://arxiv. org/abs/1905.01008.

[25] Urrego, B., "Army Signal Classification Challenge," in *GNURadio Conference,* 2018.

[26] Logue, K., et al., "Expert RF Feature Extraction to Win the Army RCO AI Signal Classification Challenge," in *Python in Science,* 2019.

[27] Vila, A., et al., "Deep and Ensemble Learning to Win the Army RCO AI Signal Classification Challenge," in *Python in Science,* 2019.

[28] Xu, Q., et al., "Device Fingerprinting in Wireless Networks: Challenges and Opportunities," *IEEE Communications Surveys & Tutorials,* Vol. 18, No. 1, 2016.

[29] Chen, F., et al., *On Passive Wireless Device Fingerprinting Using Infinite Hidden Markov Random Field,* 2012. Online: https://tinyurl.com/chen2012fingerprint.

[30] Nguyen, N., *et al.*, "Device Fingerprinting to Enhance Wireless Security Using Nonparametric Bayesian Method," in *INFOCOM,* 2011.

[31] Cain, L., et al., "Convolutional Neural Networks For Radar Emitter Classification," in *Annual Computing and Communication Workshop and Conference,* IEEE, 2018.

[32] Sankhe, K., et al., "ORACLE: Optimized Radio clAssification Through Convolutional neuraL nEtworks," in *INFOCOM,* Dataset available at https://genesys-lab.org/oracle, 2019.

[33] Youssef, K., et al., *Machine Learning Approach to RF Transmitter Identification*, 2017. Online: https://arxiv.org/abs/1711.01559.

[34] Wong, L., et al., "Clustering Learned CNN Features from Raw I/Q data for Emitter Identification," in *MILCOM,* 2018.

[35] Tong, J., et al., "Deep Learning for RF Fingerprinting: A Massive Experimental Study," *Internet of Things (IoT) Magazine,* 2020.

[36] Restuccia, F., et al., *DeepRadioID: Real-Time Channel-Resilient Optimization of Deep Learning-Based Radio Fingerprinting Algorithms,* 2019. Online: https://tinyurl.com/deepRadioID.

[37] Haigh, K. Z., et al., "Parallel Learning and Decision Making for a Smart Embedded Communications Platform," BBN Technologies, Tech. Rep. BBN-REPORT-8579, 2015.

[38] Baldo, N., et al., "A Neural Network Based Cognitive Controller for Dynamic Channel Selection," in *ICC,* IEEE, 2009.

[39] Haigh, K. Z., S. Varadarajan, and C. Y. Tang, "Automatic Learning-Based MANET Cross-Layer Parameter Configuration," in *Workshop on Wireless Ad hoc and Sensor Networks,* IEEE, 2006.

[40] Katidiotis, A., K. Tsagkaris, and P. Demestichas, "Performance Evaluation of Artificial Neural Network-Based Learning Schemes for Cognitive Radio Systems," *Computers and Electric Engineering,* Vol. 36, No. 3, 2010.

[41] Troxel, G., et al., "Adaptive Dynamic Radio Open-Source Intelligent Team (ADROIT): Cognitively-Controlled Collaboration Among SDR Nodes," in *Workshop on Networking Technologies for Software Defined Radio (SDR) Networks*, IEEE, 2006.

[42] Haviluddin, A., et al., "Modelling of Network Traffic Usage Using Self-Organizing Maps Techniques," in *International Conference on Science in Information Technology,* 2016.

[43] Qu, X., et al., "A Survey on the Development of Self-Organizing Maps for Unsupervised Intrusion Detection," *Mobile Networks and Applications,* 2019.

[44] Demestichas, P., et al., "Enhancing Channel Estimation in Cognitive Radio Systems by Means of Bayesian Networks," *Wireless Personal Communications,* Vol. 49, 2009.

[45] Selvi, E., et al., "On the Use of Markov Decision Processes in Cognitive Radar: An Application to Target Tracking," in *Radar Conference,* IEEE, 2018.

[46] Thornton, C., et al., *Experimental Analysis of Reinforcement Learning Techniques for Spectrum Sharing Radar,* 2020. Online: https://arxiv.org/abs/2001.01799.

[47] Lee, G.-H., J. Jo, and C. Park, "Jamming Prediction for Radar Signals Using Machine Learning Methods," *Security and Communication Networks,* 2020.

[48] English, D., *C4ISRNET the Compass/Net Defense Blogs: How multiINT Enables Deciphering the Indecipherable*, Accessed 2020-05-31, 2015. Online: https://tinyurl.com/c4isrnet.

[49] Hall, D., and J. Llinas, "An Introduction to Multisensor Data Fusion," *Proceedings of the IEEE,* Vol. 85, No. 1, 1997.

[50] F. White, "JDL, Data Fusion Lexicon," *Technical Panel for C*, vol. 3, 1991.

[51] Khaleghi, B., *et al.*, "Multisensor Data Fusion: A Review of the State-of-the-Art," *Information Fusion,* Vol. 14, No. 1, 2013.

[52] Castanedo, F., "A Review of Data Fusion Techniques," *The Scientific World Journal,* Vol. 2013, 2013.

[53] Dong, X., and F. Naumann, "Data Fusion: Resolving Data Conflicts for Integration," *Proceedings of the VLDB Endowmen*t, Vol. 2, No. 2, 2009.

[54] Blasch, E., and D. Lambert, *High-level Information Fusion Management and Systems Design,* Norwood, MA: Artech House, 2012.

[55] Liggins, M., II, D. Hall, and J. Llinas, *Handbook of Multisensor Data Fusion: Theory and Practice* (Second Edition), CRC Press, 2008.

[56] Blasch, E., et al., "Revisiting the JDL Model for Information Exploitation," in *FUSION,* IEEE, 2013.

[57] ISIF.org. (2020). "International society of information fusion." Accessed 2020-05-31, Online: http://isif.org/.

[58] Milisavljevic, N. (Ed.), *Sensor and Data Fusion*, In-Teh, 2009.

[59] Walts, E., *Data Fusion for C3I: A Tutorial,* Argus Business, 1986.

[60] Fabeck, G., and R. Mathar, "Kernel-Based Learning of Decision Fusion in Wireless Sensor Networks," in *FUSION*, IEEE*,* 2008.

[61] Ansari, N., et al., "Adaptive Fusion by Reinforcement Learning for Distributed Detection Systems," *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 32, No. 2, 1996.

[62] Hossain, M., P. Atrey, and A. El Saddik, "Learning Multisensor Confidence Using a Reward-And-Punishment Mechanism," *IEEE Transactions on Instrumentation and Measurement,* Vol. 58, No. 5, 2009.

[63] Stankovic, J., and T. He, "Energy Management in Sensor Networks," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences,* Vol. 370, No. 1958, 2012.

[64] Levashova, T., et al., "Situation Detection Based on Knowledge Fusion Patterns," in *International Workshop on Ontologies and Information Systems*, 2014.

[65] 3GPP, "Study on NR Positioning Support (Release 16)," Standard 3GPP TSG RAN TR 38.855, 2019.

[66] Bartoletti, S., et al., "5G Localization and Context-Awareness," in *5G Italy White eBook: from Research to Market,* 2018.

[67] Conti, A., et al., "Soft Information for Localization-of-Things," *Proceedings of the IEEE,* Vol. 107, No. 11, 2019.

[68] Lau, B., et al., "A Survey of Data Fusion in Smart City Applications," *Information Fusion*, Vol. 52, 2019.

[69] Witrisal, K., et al., "High-Accuracy Localization for Assisted Living: 5G Systems Will Turn Multipath Channels from Foe to Friend," *IEEE Signal Processing Magazine,* Vol. 33, No. 2, 2016.

[70] Guerra, A., F. Guidi, and D. Dardari, "Single-Anchor Localization and Orientation Performance Limits Using Massive Arrays: MIMO vs. Beamforming," *IEEE Transactions on Wireless Communications,* Vol. 17, No. 8, 2018.

[71] Wymeersch, H., et al., "5G mmWave Positioning for Vehicular Networks," *IEEE Wireless Communications,* Vol. 24, No. 6, 2017.

[72] Win, M. Z., *et al.,* "Network Operation Strategies for Efficient Localization and Navigation," *Proceedings of the IEEE,* Vol. 106, No. 7, 2018.

[73] Campbell, M., and N. Ahmed, "Distributed Data Fusion: Neighbors, Rumors, and the Art of Collective Knowledge," *IEEE Control Systems Magazine,* Vol. 36, No. 4, 2016.

[74] Lang, J., "Collective Decision Making Under Incomplete Knowledge: Possible and Necessary Solutions," in *IJCAI,* 2020.

[75] Makarenko, A., et al., "Decentralised Data Fusion: A Graphical Model Approach," in *FUSION,* 2009.

[76] Thompson, P., and H. Durrant-Whyte, "Decentralised Data Fusion in 2-Tree Sensor Networks," in *FUSION,* 2010.

[77] Hall, D., et al. (Eds.), *Distributed Data Fusion for Network-Centric Operations*, CRC Press, 2012.

[78] Chandola, V., A. Banerjee, and V. Kumar, "Anomaly Detection: A Survey," *ACM Computing Surveys,* Vol. 41, No. 3, 2009.

[79] Edgeworth, F., "On Observations Relating to Several Quantities," *Hermathena,* Vol. 6, No. 13, 1887.

[80] Song, X., *et al.*, "Conditional Anomaly Detection," *IEEE Transactions on Knowledge and Data Engineering,* Vol. 19, No. 5, 2007.

[81] Agyemang, M., K. Barker, and R. Alhajj, "A Comprehensive Survey of Numeric and Symbolic Outlier Mining Techniques," *Intelligent Data Analysis,* Vol. 10, No. 6, 2006.

[82] Ahmed, M., A. N. Mahmood, and J. Hu, "A Survey of Network Anomaly Detection Techniques," *Journal of Network and Computer Applications,* Vol. 60, 2016.

[83] Bakar, Z., et al., "A Comparative Study for Outlier Detection Techniques in Data Mining," in *Conference on Cybernetics and Intelligent Systems,* IEEE, 2006.

[84] Beckman, R., and R. Cook, "Outlier.......... s," *Technometrics,* Vol. 25, No. 2, 1983.

[85] Hodge, V., and J. Austin, "A Survey of Outlier Detection Methodologies," *Artificial Intelligence Review,* Vol. 22, No. 2, 2004.

[86] Markou, M., and S. Singh, "Novelty Detection: A Review—Part 1: Statistical Approaches," *Signal Processing,* Vol. 83, No. 12, 2003.

[87] Markou, M., and S. Singh, "Novelty Detection: A Review—Part 2: Neural Network Based Approaches," *Signal Processing,* Vol. 83, No. 12, 2003.

[88] Patcha, A., and J.-M. Park, "An Overview of Anomaly Detection Techniques: Existing Solutions and Latest Technological Trends," *Computer Networks,* Vol. 51, No. 12, 2007.

[89] Chawla, S., "Deep Learning Based Intrusion Detection System for Internet of Things," Ph.D. dissertation, University of Washington, 2017.

[90] Yin, C., et al., "A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks," *IEEE Access,* Vol. 5, 2017.

[91] Bowman, B., "Anomaly Detection Using a Variational Autoencoder Neural Network with a Novel Objective Function And Gaussian Mixture Model Selection Technique," M.S. thesis, Naval Postgraduate School, 2019.

[92] Rajasegarar, S., et al., "Centered Hyperspherical And Hyperellipsoidal One-Class Support Vector Machines for Anomaly Detection in Sensor Networks," *IEEE Transactions on Information Forensics and Security,* Vol. 5, No. 3, 2010.

[93] Zhang, M., B. Xu, and J. Gong, "An Anomaly Detection Model Based on One-Class SVM to Detect Network Intrusions," in *International Conference on Mobile Ad-hoc and Sensor Networks,* IEEE, 2015.

[94] Salman, T., et al., "Machine Learning for Anomaly Detection And Categorization in Multi-Cloud Environments," in *International Conference on Cyber Security and Cloud Computing,* IEEE, 2017.

[95] Fiore, U., et al., "Network Anomaly Detection with the Restricted Boltzmann Machine," *Neurocomputing,* Vol. 122, 2013.

[96] Pearl, J., *Causality* (Second Edition), Cambridge University Press, 2009.

[97] Pearl, J. "An Introduction to Causal Inference," *International Journal of Biostatistics,* 2010.

[98] Tople, S., A. Sharma, and A. Nori, *Alleviating Privacy Attacks Via Causal Learning,* 2020. Online: https://arxiv.org/abs/1909.12732.

[99] Ali, B., et al., "A Causal Factors Analysis of Aircraft Incidents Due to Radar Limitations: The Norway Case Study," *Air Transport Management,* Vol. 44-45, 2015.

[100] Martin, T., and K. Chang, "A Causal Reasoning Approach to DSA Situational Awareness and Decision-Making," in *FUSION,* 2013.

[101] Martin, T., and K. Chang, "Situational Awareness Uncertainty Impacts on Dynamic Spectrum Access Performance," in *FUSION,* 2014.

[102] Martin, T., and K. Chang, "Development and Analysis of a Probabilistic Reasoning Methodology for Spectrum Situational Awareness and Parameter Estimation in Uncertain Environments," in *FUSION,* 2015.

[103] Cousins, D., et al., "Understanding Encrypted Networks Through Signal and Systems Analysis of Traffic Timing," in *IEEE Aerospace,* Vol. 6, 2003.

[104] Tilghman, P., and D. Rosenbluth, "Inferring Wireless Communications Links and Network Topology from Externals Using Granger Causality," in *MILCOM,* 2013.

[105] Chen, Y., et al., "Analyzing Multiple Nonlinear Time Series with Extended Granger Causality," *Physics Letters A,* Vol. 324, No. 1, 2004.

[106] Morris, K., "What Is Hysteresis?" *Applied Mechanics Reviews,* Vol. 64, 2011.

[107] Deshmukh, R., et al., "Reactive Temporal Logic-Based Precursor Detection Algorithm for Terminal Airspace Operations," Vol. 28, No. 4, 2020.

[108] Keren, S., R. Mirsky, and C. Geib, *Plan, Activity, Behavior, and Intent Recognition Website,* Accessed 2020-03-14. Online: http://www.planrec.org/.

[109] Sukthankar, G., et al., *Plan, Activity, and Intent Recognition,* Elsevier, 2014.

[110] Schmidt, C., N. Sridharan, and J. Goodson, "The Plan Recognition Problem: An Intersection of Psychology and Artificial Intelligence," *Artificial Intelligence,* Vol. 11, No. 1, 1978.

[111] Keren, S.,R. Mirsky, and C. Geib, *Plan Activity And Intent Recognition Tutorial,* 2019. Online: http://www.planrec.org/Tutorial/Resources_files/pair-tutorial.pdf.

[112] Geib, C., and R. Goldman, "A Probabilistic Plan Recognition Algorithm Based on Plan Tree Grammars," *Artificial Intelligence,* Vol. 173, No. 11, 2009.

[113] Geib, C., and R. Goldman, "Plan Recognition in Intrusion Detection Systems," in *DARPA Information Survivability Conference and Exposition,* Vol. 1, 2001.

[114] Geib, C., "Delaying Commitment in Plan Recognition Using Combinatory Categorial Grammars," in *IJCAI,* 2009.

[115] Fu, M., "AlphaGo and Monte Carlo Tree Search: The Simulation Optimization Perspective," in *Winter Simulation Conference,* 2016.

[116] Latombe, G., E. Granger, and F. Dilkes, "Graphical EM for Online Learning of Grammatical Probabilities in Radar Electronic Support," *Applied Soft Computing,* Vol. 12, No. 8, 2012.

[117] Wang, A., and V. Krishnamurthy, "Threat Estimation of Multifunction Radars: Modeling and Statistical Signal Processing of Stochastic Context Free Grammars," in *International Conference on Acoustics, Speech and Signal Processing,* IEEE, 2007.

[118] Lisý, V., et al., "Game-Theoretic Approach to Adversarial Plan Recognition," in *Frontiers in Artificial Intelligence and Applications,* Vol. 242, 2012.

[119] Le Guillarme, N., et al., "A Generative Game-Theoretic Framework for Adversarial Plan Recognition," in *Workshop on Distributed and Multi-Agent Planning,* 2015.

[120] Braynov, S., "Adversarial Planning and Plan Recognition: Two Sides of the Same Coin," in *Secure Knowledge Management Workshop,* 2006.

[121] Kong, C., C. Hadzer, and M. Mashor, "Radar Tracking System Using Neural Networks," *International Journal of the Computer, the Internet and Management,* 1998.

[122] Ming, J., and B. Bhanu, "A Multistrategy Learning Approach for Target Model Recognition, Acquisition, and Refinement," in *DARPA Image Understanding Workshop,* 1990.

[123] Rogers, S., et al., "Artificial Neural Networks for Automatic Target Recognition," in *Society of Photo-Optical Instrumentation Engineers: Applications of Artificial Neural Networks,* 1990.

[124] Schmidlin, V., et al., "Multitarget Radar Tracking with Neural Networks," in *IFAC Symposium on System Identification,* Vol. 27, 1994.

[125] Xiang, Y., et al., "Target Tracking via Recursive Bayesian State Estimation in Cognitive Radar Networks," *Signal Processing,* Vol. 155, 2018.

[126] Gunturkun, U., "Toward the Development of Radar Scene Analyzer for Cognitive Radar," *IEEE Journal of Oceanic Engineering,* Vol. 35, No. 2, 2010.

[127] Bilik, I., J. Tabrikian, and A. Cohen, "GMM-Based Target Classification for Ground Surveillance Doppler Radar," *IEEE Transactions on Aerospace and Electronic Systems,* Vol. 42, 2006.

[128] Davis, B., and D. Blair, "Gaussian Mixture Measurement Modeling for Long-Range Radars," *Defense Systems Information Analysis Center Journal,* Vol. 4, No. 3, 2017.

[129] Espindle. L., and M. Kochenderfer, "Classification of Primary Radar Tracks Using Gaussian Mixture Models," *IET Radar, Sonar and Navigation,* Vol. 3, No. 6, 2009.

[130] Kristan, M., et al., "The Seventh Visual Object Tracking VOT2019 Challenge Results," in *ICCCV workshops,* 2019.

[131] Ma, M., et al., "Ship Classification and Detection Based on CNN Using GF-3 SAR Images," *Remote Sensing,* Vol. 10, 2018.

[132] Wagner, S., "SAR ATR by a Combination of Convolutional Neural Network and Support Vector Machines," *IEEE Transactions on Aerospace and Electronic Systems,* Vol. 52, No. 6, 2016.

[133] Gao, F., et al., "Combining Deep Convolutional Neural Network and SVM to SAR Image Target Recognition," in *International Conference on Internet of Things,* IEEE, 2017.

[134] Choi, J., et al., “Context-Aware Deep Feature Compression for High-Speed Visual Tracking,” in *CVPR,* IEEE, 2018.

[135] Sanna, A., and F. Lamberti, “Advances in Target Detection and Tracking in Forward-Looking Infrared (FLIR) Imagery,” *Sensors,* Vol. 14, No. 11, 2014.

[136] Yoon, S., T. Song, and T. Kim, “Automatic Target Recognition and Tracking in Forward-Looking Infrared Image Sequences with a Complex Background,” *International Journal of Control, Automation and Systems,* Vol. 11, 2013.

[137] Auslander, B., K. Gupta, and D. Aha, “Maritime Threat Detection Using Plan Recognition,” in *Conference on Technologies for Homeland Security,* 2012.

[138] Kozy, M., “Creation of a Cognitive Radar with Machine Learning: Simulation and Implementation,” M.S. thesis, Virginia Polytechnic Institute, 2019.

5

Electronic Protect and Electronic Attack

The majority of EW literature separates the discussion of EP from EA, arguing that they are fundamentally different purposes. From an AI standpoint, however, they use the same underlying algorithmic solutions.

> *The only difference between EP and EA is the mission objective: EP defines objectives with respect to oneself, while EA defines objectives with respect to others.*

EP, for example, might want to minimize BER, while EA aims at maximizing the POI; BER can be directly measured, while POI cannot. Moreover, missions often need to accomplish objectives of both types, so it is unwise to decouple the concepts. Because the goal of this book is not to describe EP and EA, but rather to highlight what challenges AI and ML can help address, we therefore present solutions from the perspective of AI DM.

The EW system must choose actions to accomplish mission objectives, given whatever context it knows about the environment and the tasks: The platform(s) have a set of capabilities, and the cognitive decision-maker composes these into strategies to achieve the desired performance. It is through these knobs, or degrees of freedom, that the EW system can accomplish its goals. Figure 5.1 outlines a spectrum of the kinds of strategies that can accomplish both EP and EA objectives.

Two key reasons for AI-based DM are *time* and *complexity*. DM time requirements are faster than humans are capable of handling. Moreover, the domain has too many inputs for a human to understand quickly, and too many choices for a human to analyze.

Figure 5.2 illustrates a notional flow. The decision-maker takes input from the ES system, consisting of the observables o and recent control feedback on the metrics m, and chooses a new strategy s to implement. If there is an online learning loop (Chapter 7), the system can update models on the fly. As a simple example,

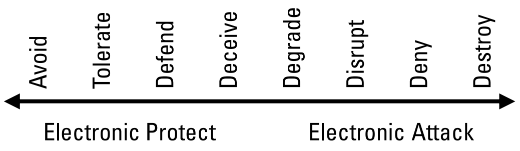


Figure 5.1 EP and EA lie on a spectrum, where a chosen strategy can be effective for multiple mission objectives.

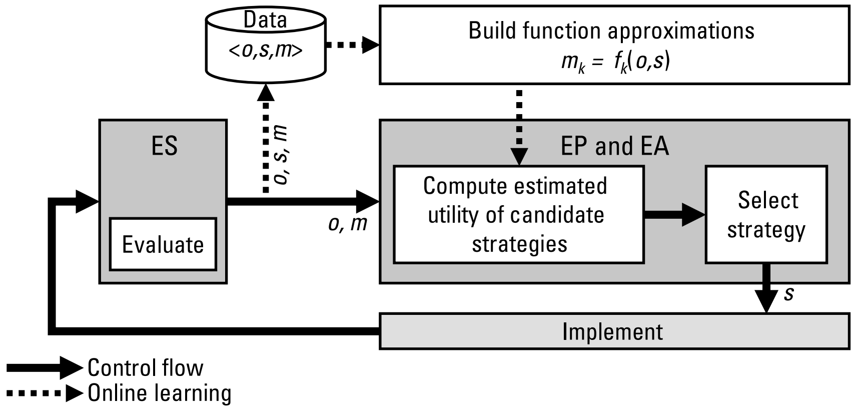


Figure 5.2 The decision-maker chooses a strategy *s* based on recent observables *o* and performance feedback of the metrics *m*. An optional online learning step updates performance models based on empirical data. (Based on Example 7.1.)

imagine choosing commute routes to work. Observables are day of week, time of day, and weather conditions. Strategies involve the route and vehicle. Metrics are time and distance. The optimizer chooses the best strategy for current conditions and updates the model as needed (e.g., if a road goes under construction).

Choosing a strategy is best-solved using *decision theory* approaches, because it follows a rigorous approach to reasoning about action choices. Decision theory is the science of choosing actions based on their expected outcomes, given constraints and assumptions about the problem, as illustrated in Figure 5.3.

Decision theory has a long history; renowned economist Kenneth Arrow describes early work [1]. Russell and Norvig [2] provide a good introductory discussion of decision theory, discussing rational DM, how to handle uncertainty

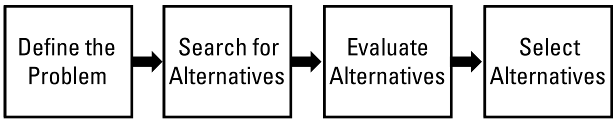


Figure 5.3 Decision theory gives a structural framework for thinking about the problem and making choices.

and partially observable environments, how to compute utility over time, and how to handle adversaries. In a complete cognitive EW system, decision theory appears in the context of three highly connected concepts: planning, scheduling, and optimization.

- *Planning* synthesizes a sequence of actions that results in a desired goal state. Planning is *what to do,* and *in what order*, as a partially ordered graph. Planning is more strategic and more global. An EBM system plans how many platforms to deploy, which resources each gets, and where they will go. Chapter 6 describes the EW planning problem, which is at a higher level than scheduling and optimization.
- *Optimization* evaluates multiple plans to choose the "best" plan. Optimization is more tactical and more local. An EW system optimizes EP and EA metrics like power usage, probability of detection, and EW BDA. Section 5.1 describes optimization, including multiple objectives.
- *Scheduling* maps a partially ordered plan to specific resources and timeslots. Scheduling worries about *when* and *how to do things*. Scheduling drives down into the specifics of when to transmit and when to receive. Section 5.2 describes scheduling.

These activities are not clearly delineated. Traditionally, humans create the EW plan, delegating detailed scheduling and automation to the system. When planning is also automated, these activities are closely tied together; in fact, solution feasibility can only be guaranteed by dynamically harmonizing the layers [3, 4].

Chapter 2 presented the objective function, and Chapter 4 (ES) explained how observables describe the environment. This chapter describes *how to choose a strategy,* with a focus on optimization and scheduling. Section 5.1 describes optimization approaches, and Section 5.2 discusses scheduling. Section 5.3 describes a desirable attributed for DM: that implementations need to be interruptible. Section 5.4 considers methods to optimize across a distributed network of nodes.

5.1 Optimization

An optimization problem consists of choosing a set of values that maximize (or minimize) a utility function. Optimization has been a subject of mathematics since at least 1646 [5], with modern cross-disciplinary work falling in many fields including operations research, economics, and AI. AI confers *heuristics* and *randomized search,* allowing systems to solve previously impossible problems. AI tackles the exponential nature of how to search the performance landscape. Section 5.1.1 describes the challenges of multi-objective optimization. Section 5.1.2

highlights some of the approximation approaches developed by the AI community. Section 5.1.3 presents a few metalearning techniques being used to improve optimization.

5.1.1 Multi-Objective Optimization

It is rarely possible to optimize all objectives simultaneously. Usually, the system must trade off the various objectives, making adequate performance toward all of them, without being optimal in any. For example, there is a rate/range/power trade-off, and a trade between robustness and efficiency.

Ramzan et al. [6] survey multi-objective optimization approaches for spectrum sharing. A classic example in radar is Pd versus Pfa; the receiver operating characteristic curve defines the trade-off [7–9]. Joint optimization of radar and communications performance is another common multi-objective problem [10], as is resource management on the platform [11, 12].

When the desired balance between objectives is not known, the *Pareto-optimal* frontier is the set of noninferior solutions for each objective, as illustrated in Figure 5.4. The "best" decision becomes a choice along the frontier, using one of these common approaches:

1. Balancing mission parameters that define the desired balance in the form of an objective function (e.g., through weights on the metrics) (Section 2.4);
2. Solving a constraint model, where some metrics must meet minimum criteria while others are optimized (Figure 5.5);
3. Computing a probability distribution over the options (e.g., game theory) (Section 6.2);
4. If there's enough time, showing feasible options to a human operator (Section 6.3).

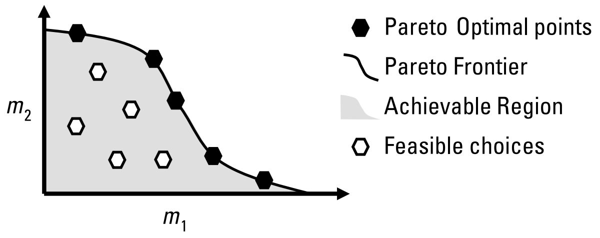
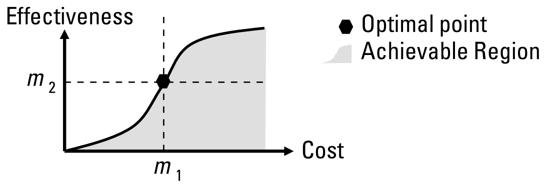


Figure 5.4 Pareto-optimal points occur when no one metric can be improved without degrading other metrics: A higher value for m_1 reduces the value for m_2.



When the system operates under minimal operating requirements, it can be beneficial to formulate the problem using one of two methods:

(a) Maximize Effectiveness subject to constraints on Cost, or
(b) Minimize Cost subject to constraints on Effectiveness

A goal of Cost $\leq m_1$ and Effectiveness $\geq m_2$ determines an optimal operating point. While mathematically equivalent, these two formulations lead to different practical issues [13], [14].

Figure 5.5 Constraint models support DM along a Pareto frontier. In contrast, a single objective function would combine all metrics, using negative weights for costs.

As an example, consider a tone jammer. There are many different techniques that can mitigate the interference it causes; each technique has a different expected benefit and cost (Table 5.1). Even in this simple case, the choice is not always straightforward. First, specific details of the jammer may reduce the expected benefit of the technique, and therefore the system should learn to estimate the performance based on these observables (Section 4.2). Second, different costs may be more or less tolerable to the platform and mission. Finally, different stages

Table 5.1
Different Techniques to Mitigate a Tone Jammer with Different Expected Effectiveness and Costs in a Communications EP Setting

Strategy	**Effectiveness**	**Cost**
Notch filter	Good, but as power goes up, an adaptive notch may remove too much signal	Low
Beamforming	Usually good; poor if interference is distributed	Computation and multiple antennas
DSA	Good, but jammer may follow	Sensing; coordination time
Narrowband-hopping	Depends on bandwidth of tone	Coordination time
Redundant packets	Good if tone is intermittent; poor otherwise	Power (and half the throughput)
Spreading	Poor	More gain in receive path increases vulnerability to saturation
Routing around	Medium	Power (multiple nodes), Latency

of the mission may have different priorities and thus mandate a different choice, even for techniques that have equal expected effectiveness.

Traditional look-up tables and one-to-one mapping of environment to technique are inferior in every realistic case with many performance metrics, capabilities, platforms, missions, and complex jammers.

5.1.2 Searching Through the Performance Landscape

AI tackles the problem of how to search the exponential performance landscape. Traditional mathematics approaches focused on complete, optimal solutions, but computing a solution for most real-world problems (including EW) is impossible. Per our definition of the objective function, EW has an exponential number of strategies ($\Pi_{\forall c} v_c$, where v_c is the number of possible values for the control parameter c), quickly approaching infinity. When the number of strategies is small, the system can estimate the utility of every strategy, as outlined in Algorithm 5.1.

As the number of strategies grows, alternative approaches become necessary. If the objective function is smooth and has relatively few local maxima, gradient-based searches are efficient and effective methods to finding a solution.

EW objectives, however, are rarely smooth, not necessarily continuous, and frequently have many local maxima, making randomized search methods appropriate [16]. Also known as *metaheuristic methods*, *Monte Carlo methods*, or *stochastic algorithms*, they statistically guarantee an optimal solution; there is typically no way to know whether they have found an optimal solution.

Optimality, however, is rarely critical; usually "good" is good enough.

In EW, the domain is partially observable, and changes so fast that an approximate solution is sufficient. In other words, one can overthink the problem. Nobel Prize–winning economist and father of AI, Herbert A. Simon, developed the

Algorithm 5.1 When the number of candidate strategies is small, node *n* can exhaustively compute the estimated utility for each strategy. This approach effectively chooses EP interference-mitigation approaches for a fielded communications network (Example 7.1, [15]).

```
For each candidate strategy $s_i$:                  //All candidate strategies for node n
    For each metric $m_k$:
        $m_k = f_k(o, s_i)$                          //Estimated metrics
    Compute $\tilde{U}(s_i)$ from all metrics $m_k$   //Estimated utility
Select $s = \mathrm{argmax}_{s_i} \tilde{U}(s_i)$   //Best strategy
```

concept of *bounded rationality* [17], which says that the rationality of individuals is limited by the information they have, the limitations of their minds, and *the finite amount of time they have to make a decision.* Section 5.3 continues this line of thought.

Moreover, many EW systems are specified to have certain performance against the most stressing threats. For example, a radar employing adaptive beamforming may be specified to create null depth of, say 50 dB, to counter a noise jammer. The 50 dB is typically chosen to counter the most powerful jammer at the closest range—creating the highest effective isotropic radiated power (EIRP)—at the closest angle to the desired target. Therefore, if the learning system chooses adaptive beamforming as the correct EP, it will likely still be effective against a very large population of jammers because of the extra margin built into the system for this mode, even if some of the parameters for it aren't optimal (e.g., sample support for secondary data estimation).

Randomized search methods compute the quality of individuals in a population over several iterations before selecting a final choice (Algorithm 5.2). In our case, the population P is a subset of all candidate strategies available at the node, and the set is much smaller than the total number of available candidates. Randomized algorithms differ in the metaheuristic they use to compute each successive iteration.

Our discussion of randomized optimization algorithms is *far* from being comprehensive; we list ones that have shown promise to the EW space and mention a few relevant examples. Beheshti and Shamsuddin [18] review population-based meta-heuristics, including mathematical formulations and trade-offs. Jamil and Yang [19] present 175 different benchmark problems that validate the

Algorithm 5.2 Metaheuristic methods explore the performance landscape using randomized methods that search through a small population of individuals. The utility function $\tilde{U}_n$ evaluates the quality of the individuals.

```
P = Compute initial population of candidate strategies for node n
For each iteration:                         // Convergence or max iterations
    For each candidate strategy $s_i \in P$:
        For each metric $m_k$:
            $m_k = f_k(o, s_i)$                   // Estimated metrics
    Compute $\tilde{U}_n(s_i)$                      // Estimated utility
    P = Update population (location or individuals)
Select $s = \mathrm{argmax}_{s_i \in P}(\tilde{U}_n(s_i))$     // Best strategy in last population
```

performance of optimization algorithms and (importantly) the characteristics of each. Ramzan et al. [6] present a survey of optimization approaches in the context of CR networks. The NATO *Cognitive Radar* report presents many approaches and applications [20]. These techniques are merely a starting point, and can be combined and augmented with other approaches. Several randomized optimization algorithms are described as follows.

- *Ant colony optimization (ACO)* is a robust, versatile metaheuristic for solving combinatorial optimization problems, wherein a population of artificial "ants" randomize the search to a solution [21, 22]. As the ants move through the landscape, they leave "pheromones"; higher probability paths have denser pheromones. Updating the population in Algorithm 5.2 for ACO means "moving" the ant to a different strategy.
  - Pros: ACO is inherently parallelizable, guarantees convergence, and can be used in dynamic applications.
  - Cons: Because ACO is a randomized search, time to convergence is uncertain, it is hard to analyze theoretically, and it doesn't guarantee an optimal solution. Good solutions in the previous iteration increase the probability that solutions in subsequent iterations will follow a similar path, reducing the diversity of the population.

  ACO is used in cognitive radio networks to build routing trees [23, 24], assign channels [25], and tune transmission parameters [26]. Karaboga et al. [27] uses ACO to steer nulls in linear antenna arrays. Ye et al. [28] combines ACO with Tabu search to make collaborative cooperative jamming decisions.
- *Particle swarm optimization (PSO)* uses a population (swarm) of candidate solutions (particles) to move through the search space [29]. Particle movements are guided by their own estimate of their position, and the swarm's best-known position.
  - Pros: PSO has a simple structure that is easy to encode. PSO is more efficient than genetic algorithms, especially in problems with continuous variables. PSO can operate in dynamic applications.
  - Cons: The choice of PSO parameters can have a large impact on optimization performance and can converge prematurely to local optima, especially with complex problems. Stochastic variability can be very high.

  PSOs are used to compute multiparameter solutions for transmission and resource allocation in radio networks [6]. PSO works well for radar pulse compression codes [30, 31], and designing discrete frequency waveforms [32, 33].

- *Genetic algorithms (GAs)* are another population-based randomized search method in which solutions are encoded as "chromosomes" [34]. This metaheuristic uses mutation, crossover, and selection to generate high-quality solutions to complex problems.
  - Pros: GAs are naturally suited for discrete-valued problems, and are easily parallelized.
  - Cons: GAs are prone to premature convergence; maintaining diversity in the population is critical [35–37]. GA problem-encoding is problem-specific, and these encodings can be difficult to create. Also, GAs are less computationally efficient than PSO [38].

  GAs are used to compute multiparameter solutions for transmission, with various goals including power management, transmission performance, and shared spectrum usage [6, 39]. GAs have been used for radar design [40], waveform selection [41], waveform design [42, 43], target identification [44, 45], and jammer suppression [46].
- *Simulated annealing* is another search metaheuristic that probabilistically decides whether to move from one solution to another [47]. Unlike GAs and PSO, there is only one search instance. The probability function is usually chosen so that the probability of deciding to move decreases when the difference in solution quality increases; that is, small uphill moves are more likely than large uphill moves. It can be used within GAs and PSO to improve search time.
  - Pros: Simulated annealing is relatively easy to code and generally gives a good solution. (It statistically guarantees finding an optimal solution.) Because simulated annealing has only one individual, it requires significantly less memory than population-based approaches.
  - Cons: For the same reason, it is more likely to end up in the same valley it started in; it doesn't know whether it has found an optimal solution.

  Simulated annealing has been used for topics as varied as detecting faults in antennas [48], meeting the QoS for a radio network [49], estimating radar cross-sections [50], localizing jammers [51], and suppressing jamming [52]. Simulated annealing and GAs have been combined for power allocation in CR networks [53], radar imaging [54], and designing a MIMO sensor array [55]. Simulated annealing has been combined with other AI techniques (e.g., Case-based reasoning [56]) to increase the convergence rate.
- *Cross-entropy method (CEM)* is a randomized search method for importance sampling; it draws a sample from a probability distribution, and then minimizes the cross-entropy between this distribution and the target dis-

tribution to produce a better sample in the next iteration [57, 58]. Amos and Yarats [59] describe a method to differentiate CEM so that it can be used with gradient descent methods.

- Pros: CEM defines a precise mathematical framework, in some sense optimal learning rules. It is robust to parameter settings. CEM is valuable for rare event simulation, where very small probabilities need to be accurately estimated.
- Cons: Time to convergence is uncertain; the stopping condition is a heuristic. The original CEM method has high storage and computational requirements [60].

CEM is used to estimate and optimize network reliability [57, 61], find network paths [62], and place nulls [63]. Naeem et al. [64] use CEM to optimize cooperative communication and greenhouse gas emissions. CEM can also be used in the antenna design problem [65] and in RL to learn more quickly [58, 60]. Like simulated annealing, CEM can be combined with the population-based approaches to parallelize solutions [66, 67].

Despite optimization being a mature field, new approximation techniques are developed regularly. Moreover, individual optimization approaches can be combined to get better performance (speed or solution quality). Many other approaches can likewise help, including hybrid approaches (Section 3.3) and metalearning (Section 5.1.3). For example, while the performance surface is in theory infinitely large, EW engagements are constrained by physics and progression of engagement state by collection or denial of information. Heuristics can leverage this connected state space and use the randomized methods to explore the remaining unknown regions.

5.1.3 Optimization Metalearning

Metalearning is the process of learning to learn [68, 69]. Metalearning is an awareness and understanding of the phenomenon of learning, rather than learning the subject knowledge. A metalearner changes the operational parameters of an algorithm so that its expected performance increases with experience. There is some terminology awkwardness because *learning* involves creating rules to assess or decide more effectively, while *metalearning* is learning-to-learn. Many learning algorithms are optimizers; therefore metalearning can learn to optimize a learner that improves an optimizer.

Early research in metalearning looked at bias-management to select between different algorithms [70], meta-genetic programming to evolve a better genetic program [71], and learning to optimize learning rules in neural networks [72].

Metalearning learns how to exploit structure in the problems of interest in an automatic way, so that they no longer need to be tuned by hand [73]. Metalearning offers a variety of benefits:

1. *Speed:* Learn or optimize faster;
2. *Ease of use:* Automating the process of tuning algorithms;
3. *Adaptability:* agile to environmental changes;
4. *Reduced data:* transfers knowledge from one context to another.

Metalearning approaches can be combined, as illustrated in Figure 5.6, with the overall purpose of improving the generalization of ML models and the efficiency and accuracy of DM engines.

Metalearning can learn a low-dimensional representation of the task [74, 75] to make it easier to search. Here, the metalearner changes the number of parameters in $\tilde{U}_n$. Other dimensionality-reduction approaches include principal components analysis (PCA) and ANNs (Kohonen networks and auto-encoders per Section 3.1.2).

Metalearning can help prioritize the search for strategies, starting with the most important controllables (i.e., those that contribute most to solution quality). The algorithm searches over options for a small handful of important controllables, while using default values or the values selected in the previous timestep, for the less important ones. The algorithm searches for less important controllables only after fixing these important values. This approach is akin to learning how to plan (Section 7.3). An example in communication jamming would be the use of an appropriate duty cycle for causing interference as a major controllable,

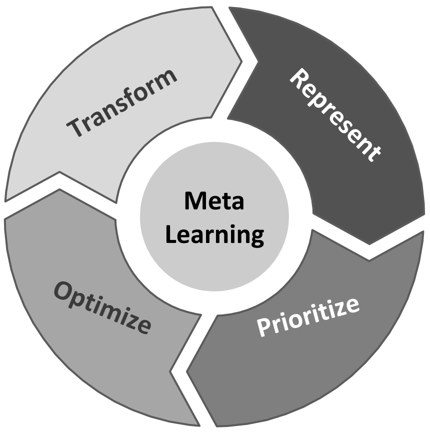


Figure 5.6 Metalearning changes the representation of the data, prioritizing search, adjusting hyperparameters, and transforming the utility function to improve solution quality.

with minor controllables being the details of a jamming waveform like frequency offsets and particular modulation techniques.

Metalearning can analyze the search process to optimize hyperparameters and search more efficiently [76–78]. A metalearner optimizes a set of hyperparameters θ that control the optimization of $\tilde{U}_n$. The metalearner uses a metaUtility function ($\tilde{U}_n$, $\mathcal{J}$), where $\mathcal{J}$ evaluates qualities of the optimizer, such as the speed to convergence, as illustrated in Figure 5.7.

Orthogonally, approximating the derivative creates a differentiable version of $\tilde{U}_n$ that is appropriate for gradient-based methods [59].

Section 7.3 presents additional learning methods to improve DM, within the context of RL and direct interaction with the environment. All the methods presented here can also be viewed as RL methods.

5.2 Scheduling

Scheduling maps a partially-ordered plan to specific resources and timeslots. Scheduling is concerned with when and how to do things. In EW, scheduling commonly furnishes the specifics of when to transmit and when to receive.

A resource scheduler on a single EW node may decide when to activate sensors, when and how to transmit, when and how to avoid specific electromagnetic signatures, and/or when and how to receive. For distributed EW systems, such particulars (of how and when) only grow in complexity. An EW resource scheduler needs (1) a sequence of actions and their resource requirements, (2) a set of resources to use, (3) constraints, and (4) an objective function.

Critical path methods build a directed graph relating actions; the critical path is the one with the total longest duration; it determines the maximum duration of the entire EW mission. Any task that lays on the critical path is called a *critical task*. Critical tasks and their interdependencies can be represented with a Gantt chart or similar approaches. Critical tasks are spectrum sensing, spectrum analysis, and spectrum decision (Figure 5.8); an EW mission encompasses numerous cognition cycle iterations.

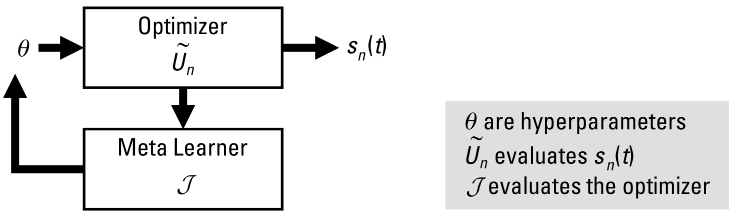

θ
Optimizer
$\tilde{U}_n$
$s_n(t)$
Meta Learner
$\mathcal{J}$
θ are hyperparameters
$\tilde{U}_n$ evaluates $s_n(t)$
$\mathcal{J}$ evaluates the optimizer

Figure 5.7 Metalearning optimizes the hyperparameters θ of the optimizer using a metaUtility function $\mathcal{J}$.

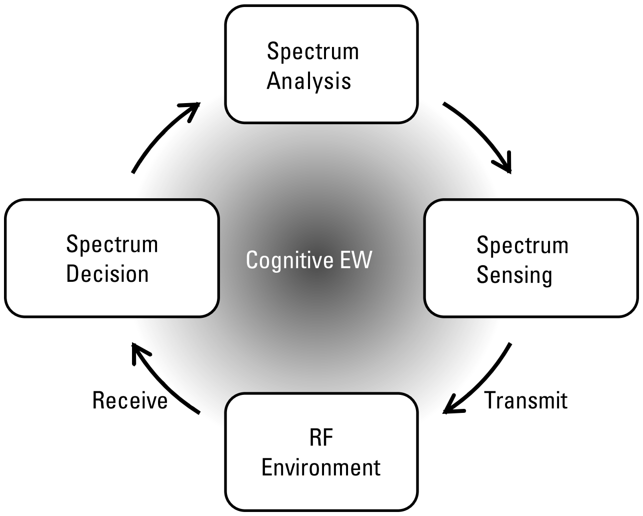


Figure 5.8 Each of the high-level tasks—sensing, analysis, and DM—drive the transmit/receive schedule.

In computing, the first scheduling approaches were taken from the field of operations management. Assembly lines and many other human aspects of life require scheduling, sharing the same concerns with computing, including a desire for efficiency. The development of scheduling policies is concerned with determining what the key assumptions should be and what metrics are important. In computer science, metrics include *workload, jobs, turnaround time,* and *fairness* [79]. Today, many scheduling disciplines exist and are found in solutions for network routers, operating systems, disk drives, embedded systems, and packet-switched wireless networks.

In EW systems, the scheduler chooses actions that will ultimately enable the EW system to transmit and to receive. The scheduler can perform actions in a cyclical or linear manner. Linear schedules are finite, whereas cyclical ones naturally repeat. Ideally, an awareness of all possible states is necessary for proper task scheduling; however, partial observability of some states could make it difficult to accomplish a given task. To address numerous constraints such as output power, energy efficiency, and time requirements, the automatic scheduler may assign weights or costs to these various constraints thus eliminating the need to preplan manually, an arduous task that is often impossible given the timing constraints of the system.

In automatic scheduling, all tasks may have to undergo an automatic prioritization of execution order, but the nature of tasks themselves might have to be predetermined by the cognitive EW system designer. Furthermore, a single node of distributed EW system may be aware of all of its states within its world's purview yet it may not be aware of other nodes' or actors' states. Partial accomplishment of tasks can be achieved in time, locally, on each node, but coordination among nodes is crucial as nodes might not be striving toward the same goal and might end up being in conflict with each other's objectives. For example, the scheduler has to be cognizant of limited node connectivity and visibility, validate that tasks have been successfully accomplished, and manage accountability for wrongly executed tasks (reward and punishment).

The designer's goal is to create a scheduler that achieves all of its scheduling objectives within the mission and operating constraints. In distributed systems, the decision should be based upon nodes' responses (or lack thereof) to determine if some tasks need to be abandoned altogether or whether another node should be allowed to carry out the task. Lack of true broad spectrum "visibility" in spacetime will also determine the boundaries of one's scheduling algorithm. Awareness of many states, past, current, or preplanned/predicted future, allows the scheduler to adjust its parameters when executing the tasks (i.e., adjusting its overall master schedule). For real-time cognitive EW systems, time is of the essence. Thus, near real-time decisions must determine the appropriate order of tasks to execute, based on many internal and external factors such as sensor data, weather, antenna array availability or, generally, resource availability. A scheduler should adjust the schedule based on resource availability.

Modern commercial communications systems, for example, have schedulers that allow them to prioritize when to transmit and receive and in what frequencies and time slots, e.g., frequency-division multiple access (FDMA) or time-division multiple access (TDMA) schemes. DSA is also an attractive feature for any EW system to possess and should be a part of the system's scheduling algorithm.

The scheduler decides in which order to execute system tasks, while assessing resource availability and determining how to utilize these resources. The scheduler must rely on SA information and reorder tasking based on the information received. While a simple scheduler can be entirely deterministic (e.g., rule-based or hard-coded with predetermined inputs and outputs), the chosen approach must support scheduling to account for unexpected, unknown or partially observable states.

Lastly, overall awareness of the mission environment is important for successful execution of system tasks (whatever they may be) in the best possible order. Flexibility is key; prioritization and reprioritization are equally important. Valuable information may sometimes arrive at an inconvenient time, but cannot be ignored and should be recognized by the scheduler. The scheduler may also benefit from having a "fight or flight" capability or an emergency mode, when the ongoing schedule might have to be abandoned in lieu of an emergency to instantiate a new tasks' order to protect the system.

Table 5.2 provides several examples from the literature of various schedulers that are relevant to cognitive EW.

Table 5.2
Examples of Optimization Approaches Used by EW Schedulers

Category	Examples
Comms	Satellite broadcast scheduling using local search, simulated annealing, and ACO [80]; satellite scheduling for low-power terminal operations [81]; minimizing energy to transmit [82]; long-term evolution uplink timing-alignment and power-control [83]; user scheduling for multiple antenna systems [84]; ML-enabled fish-eye state routing (FSR) for MANET optimization [85]; adaptive MANET routing algorithm based on the simulated annealing [86]; MANET transmission scheduling protocol based on recovered minislots [87]; position-based broadcast TDMA scheduling for MANETs [88]; spectrum sensing scheduler for CRNs [89]; GA-based scheduling algorithm for CRNs [90]; real-time heuristic scheduling for CRNs [91]; real-time scheduler for CRNs [92]; polynomial-time, energy-efficient heuristic scheduler for centralized CRNs [93]; scheduling for distributed sensor networks [94]; survey of resource allocation techniques [95], with approaches including cloning GA, estimation of distribution algorithm (EDA) based on weighted-sum method (WSM) for green radio resource allocation, and CEO.
Radar	Phased-array task scheduling using hybrid GAs and heuristics [96]; ML-enabled active electronically scanned array (AESA) antenna scheduling [97]; multifunctional radar network greedy and heuristic scheduling [98]; multifunction phased array radar scheduling heuristics [99]; multifunction radar using Monte Carlo tree search [100].
EW	FSR and power control for EP [85]; partially observable Markov decision process (POMDP) for airborne electronic countermeasures (ECMs) [101]; ML-enabled. periodic sensor scheduling for ES [102]; game-based approach for jamming resources and strategy optimization for EA [103]; risk-based multisensor scheduling using POMDPs and decision trees for target threat assessment [104].
General framework	Scheduling based on knowledge-based engineering [105]; heterogeneity-driven task scheduling and end-to-end synchronized scheduling algorithms for networked embedded systems [106]; utility accrual real-time scheduler based on polynomial-time heuristic algorithm [107]; RL scheduler for heterogeneous distributed systems [108]; generalized ML-based solution for large-scale resource scheduling [109]; online resource scheduling algorithm using deep RL [110]; ML approaches to learning heuristics for combinatorial optimization [111]; scheduling approaches using evolutionary computation [112]; temporal reasoning for planning and scheduling [113].

5.3 Anytime Algorithms

EW DM algorithms operate in rapidly changing environments with hard real-time requirements. New priorities can "pop up" at any time, and resources may be unexpectedly depleted. A desirable trait for a decision-maker is that it can generate solutions quickly, or, when time is available, spend more time deliberating. While it may be a goal to have zero-latency for decisions, in practice, a longer integration time (and waiting a second or two) may yield a better result.

Anytime algorithms find better and better solutions the longer they run. Invented by Dean and Boddy in the 1980s [114], anytime algorithms can be interrupted at any point during computation to return a result whose utility is a function of computation time. The similar concept of *flexible computation* [115] explicitly balances the benefits of additional computation time against the costs of acting with a partial solution. Both ideas have roots in Herb Simon's concept of *satisficing* [116], which involves searching for a solution until an acceptability threshold is met; satisficing is particularly useful when optimal solutions cannot be determined because of computational intractability or lack of information. Longer observation times may generate better quality observations, and longer compute times may yield better optimization. Anytime algorithms recognize that the time required to find the optimal solution typically reduces the overall utility, similar to the law of diminishing returns.

Most metaheuristic algorithms satisfy this condition: Each iteration improves the quality of the best-available solution and therefore can be stopped as needed. Adding to the list of citations from Section 5.1.2, anytime algorithms are effective in domains where approximate solutions need to be generated quickly, from image alignment [117] to adaptive weather-control radars [118].

Zilberstein looks at the problem of metacontrol: compiling, controlling, and managing multiple anytime algorithms [119, 120]. Zilberstein, who used a radar threat analysis and target-assignment problem as one of his motivating scenarios, saying [119]:

> The radar component, designed to track objects, and the planning component, designed to perform target assignment, are clearly interdependent: the quality of the first clearly affects the quality of the second. The use of conditional performance profiles and dynamic scheduling seems essential for solving such problems.

Metalearning (Section 5.1.3) augments this work to learn and utilize performance predictions [121–123].

5.4 Distributed Optimization

An EW system may need to optimize across multiple decision-makers on a single platform, or across platforms. Both radar and comms systems require distributed DM, but comms systems have greater latency and more coordination. Approaches differ according to the nature of the solution:

- *Centralized:* A single decision-maker finds a solution for all components and all nodes;
- *Distributed:* Decision-makers use local communications to coordinate actions;
- *Decentralized:* Decision-makers are fully independent and do not rely on communications for coordination.

On a single platform, a single optimizer will find generally a solution that is closer to optimal than multiple optimizers, because it can search for a single global solution. However, a single optimizer is more challenging to design due to differing timescales, differing types of information, and the amount of information to synchronize.

Across the network of platforms, a single, centralized optimizer is inappropriate. The critical concern is that of a single bottleneck: If the centralized optimizer fails, the entire system fails. Additional concerns include the latency of DM, information privacy, and detectable emissions. It is better to make a locally good decision quickly than it is to attempt to find a globally optimal decision that is irrelevant before it can be implemented. Figure 5.9 illustrates the practical limits of a centralized optimization approach. Many distributed and decentralized optimization solutions are naturally anytime algorithms, making them inherently suitable for generating satisficing solutions in this dynamic domain.

In EW, centralized nodes can capture "slower-time" information, including changes to the broader mission policy, EWO interactions (Section 6.3), and slower information dissemination and task management [124, 125] (Section 6.1.6).

Distributed optimization approaches communicate across local neighborhoods to coordinate actions. Traditional approaches assume that communications is safe and plentiful. This assumption is not true in EW where communications limited or denied due to jamming, mobility disconnects, or decisions to reduce RF emissions. Section 6.1.4 discusses approaches for planning when to communicate—managing the benefit of information sharing against the cost of transmission.

Several surveys describe distributed optimization solutions, noting communications overhead, memory usage, and optimality [126–128].

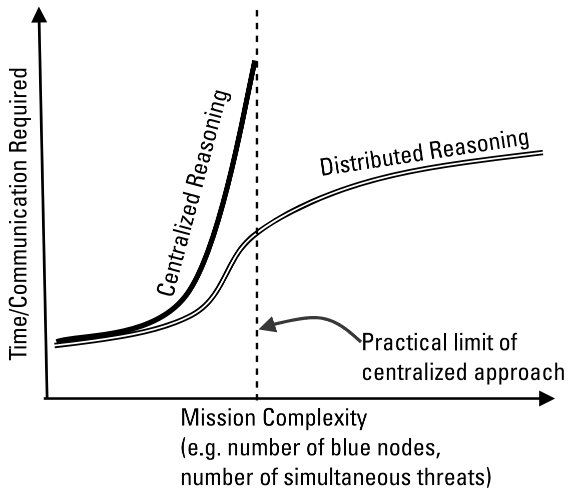


Figure 5.9 Centralized reasoning quickly reaches a practical limit.

Consensus propagation [129] is a lightweight communications approach that allows each node to share its local estimate of the global utility and quickly compute the global average, despite not knowing how many nodes are in the network. ACO algorithms likewise have low communications demands.

Fully decentralized coordination requires each node to independently evaluate its contribution to the global solution. One option is to *formulate the utility function* so that each node's local optimum accurately captures its contribution to the team's global optimum. The formal problem definition of Callout 2.1 uses a fully local approximation of the true utility to choose EP interference-mitigation approaches for a communications network in a online learning environment (Example 7.1, [15]). Molzhan et al. [130] present an approach for decentralized optimization in electric power systems, against the backdrop of classical control theory and real-time feedback control.

Communication-free learning [131] formulates the problem so that local sensing is sufficient to determine whether the solution is valid; the authors describe how to model graph coloring, channel assignment with channel-dependent interference, intersession network coding, and decentralized transmission scheduling. Each node needs to know only its own assignments; it doesn't need to know the choices of the other nodes.

Distributed optimization approaches support information fusion [118], resource allocation [124, 131, 132] and scheduling [133]. A common application for distributed optimization is timing synchronization, which then can be used for data fusion, duty cycling, cooperative localization, and coordinated actions such as distributed beamforming [134–136].

5.5 Conclusion

Just as there is no free lunch for ML and statistical inference [137], there is no free lunch for optimization [138]: If an algorithm performs well on a certain class of problems then it necessarily pays for that with degraded performance on the set of all remaining problems.

This chapter presented some of the optimization approaches shown to be effective for EW problems; these approaches support:

- Support multiple objectives;
- Rapidly search the exponential performance surface;
- Handle a dynamic environment;
- Improve solution quality when given more time;
- Operate well for multiple nodes under limited communication.

Chapter 6 discusses more strategic DM: long-term battle planning and management.

References

[1] Arrow, K., "Decision Theory and Operations Research," *Operations Research*, Vol. 5, No. 6, 1957.

[2] Russell, S., and P. Norvig, *Artificial Intelligence: A Modern Approach,* Pearson Education, 2015.

[3] Chien, S., et al., "Automated Planning and Scheduling for Goal-Based Autonomous Spacecraft," *Intelligent Systems,* Vol. 13, No. 5, 1998.

[4] R-Moreno, M., et al., "RFID Technology and AI Techniques for People Location, Orientation and Guiding," in *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems,* 2009.

[5] Du, D., P. Pardalos, and W. Wu, "History of Optimization," in *Encyclopedia of Optimization* (C. Floudas and P. Pardalos, Eds.), Springer, 2009.

[6] Ramzan, M., et al., "Multi-Objective Optimization for Spectrum Sharing in Cognitive Radio Networks: A Review," *Pervasive and Mobile Computing,* Vol. 41, 2017.

[7] Cassady, P., *Bayesian Analysis of Target Detection with Enhanced Receiver Operating Characteristic,* 2019. Online: https://arxiv.org/abs/1903.08165.

[8] Chalise, B., M. Amin, and B. Himed, "Performance Tradeoff in a Unified Passive Radar and Communications System," *IEEE Signal Processing Letters,* Vol. 24, No. 9, 2017.

[9] Grossi, E., M. Lops, and L. Venturino, "A New Look at the Radar Detection Problem," *Transactions on Signal Processing,* Vol. 64, No. 22, 2016.

[10] Chiriyath, A., et al., "Radar Waveform Optimization for Joint Radar Communications Performance," *Electronics,* Vol. 8, No. 12, 2019.

[11] AlQerm, I., and B. Shihada, "Adaptive Multi-Objective Optimization Scheme for Cognitive Radio Resource Management," in *GLOBECOM,* IEEE, 2014.

[12] Mitchell, A.. et al., "Cost Function Design for the Fully Adaptive Radar Framework," *IET Radar, Sonar and Navigation*, 2018.

[13] Haigh, K. Z., O. Olofinboba, and C. Y. Tang, "Designing an Implementable User-Oriented Objective Function for MANETs," in *International Conference On Networking, Sensing and Control,* IEEE, 2007.

[14] Charlish, A., and F. Hoffman, "Cognitive Radar Management," in *Novel Radar Techniques and Applications* (R. Klemm, et al., Eds.), Scitech Publishing, 2017.

[15] Haigh, K. Z., et al., "Parallel Learning and Decision Making For a Smart Embedded Communications Platform," BBN Technologies, Tech. Rep. BBN-REPORT-8579, 2015.

[16] Zabinsky, Z. B., "Random Search Algorithms," in *Wiley Encyclopedia of Operations Research and Management Science,* 2011.

[17] Simon, H., “A Behavioral Model of Rational Choice,” *Quarterly Journal of Economics,* Vol. 69, No. 1, 1955.

[18] Beheshti, Z., and S. Shamsuddin, “A Review of Population-Based Metaheuristic Algorithm,” *International Journal of Advances in Soft Computing and its Applications,* Vol. 5, No. 1, 2013.

[19] Jamil, M., and X.-S. Yang, “A Literature Survey of Benchmark Functions for Global Optimization Problems,” *International Journal of Mathematical Modelling and Numerical Optimisation,* Vol. 4, No. 2, 2013.

[20] Task Group SET-227, “Cognitive Radar,” NATO Science and Technology, Tech. Rep. TR-SET-227, 2020.

[21] Dorigo, M., “Optimization, Learning and Natural Algorithms,” Ph.D. dissertation, Politecnico di Milano, Milan, Italy, 1992.

[22] Katiyar, S., I. Nasiruddin, and A. Ansari, “Ant Colony Optimization: A Tutorial Review,” in *National Conference on Advances in Power and Control,* 2015.

[23] Almasoud, A., and A. Kama, “Multi-Objective Optimization for Many-to-Many Communication in Cognitive Radio Networks,” in *GLOBECOM,* IEEE, 2015.

[24] Zhang, Q., Q. He, and P. Zhang, “Topology Reconfiguration in Cognitive Radio Networks Using Ant Colony Optimization,” in *Vehicular Technology Conference,* IEEE, 2012.

[25] He, Q., and P. Zhang, “Dynamic Channel Assignment Using Ant Colony Optimization for Cognitive Radio Networks,” in *Vehicular Technology Conference,* IEEE, 2012.

[26] Waheed, M., and A. Cai, “Cognitive Radio Parameter Adaptation in Multicarrier Environment,” in *International Conference on Wireless and Mobile Communications,* IEEE, 2009.

[27] Karaboga, N., K. Güney, and A. Akdagli, “Null Steering of Linear Antenna Arrays with Use of Modified Touring Ant Colony Optimization Algorithm,” *International Journal of RF and Microwave Computer-Aided Engineering,* Vol. 12, No. 4, 2002.

[28] Ye, F., F. Che, and L. Gao, “Multiobjective Cognitive Cooperative Jamming Decision-Making Method Based on Tabu Search-Artificial Bee Colony Algorithm,” *International Journal of Aerospace Engineering,* 2018.

[29] Kennedy, J., and R. Eberhart, “Particle Swarm Optimization,” in *International Conference on Neural Networks,* Vol. 4, 1995.

[30] Hafez, A., and M. El-latif, “New Radar Pulse Compression Codes by Particle Swarm Algorithm,” in *IEEE Aerospace Conference,* 2012.

[31] Li, B., “Particle Swarm Optimization for Radar Binary Phase Code Selection,” in *Radar Sensor Technology,* 2018.

[32] Reddy, B., and U. Kumari, “Performance Analysis of MIMO Radar Waveform Using Accelerated Particle Swarm Optimization Algorithm,” *Signal and Image Processing,* Vol. 3, No. 4, 2012.

[33] Praveena, A., and V. Narasimhulu, “Design of DFC Waveforms for MIMO Radar Using Accelerated Particle Swarm Optimization Algorithm,” *International Journal of Engineering Trends and Technology,* Vol. 33, No. 2, 2016.

[34] Holland, J., "Genetic Algorithms and Adaptation," in *Adaptive Control of Ill-Defined Systems,* Vol. 16, 1984.

[35] Meadows, B., et al., "Evaluating the Seeding Genetic Algorithm," in *Australasian Joint Conference on Artificial Intelligence,* 2013.

[36] Watson, T., and P. Messer, "Increasing Diversity in Genetic Algorithms," in *Developments in Soft Computing,* 2001.

[37] Chuang, C.-Y., and S. Smith, "Diversity Allocation for Dynamic Optimization Using the Extended Compact Genetic Algorithm," in *Congress on Evolutionary Computation,* IEEE, 2013.

[38] Hassan, R., et al., *A Copmarison [sic] of Particle Swarm Optimization and the Genetic Algorithm,* 2004. Online: https://tinyurl.com/pso-vs-ga.

[39] Mehboob, U., et al., "Genetic Algorithms in Wireless Networking: Techniques, Applications, and Issues," *Soft Computing,* 2016.

[40] Bartee, J., "Genetic Algorithms as a Tool for Phased Array Radar Design," M.S. thesis, Naval Postgraduate School, Monterey, California, 2002.

[41] Capraro, C., et al., "Using Genetic Algorithms for Radar Waveform Selection," in *IEEE Radar Conference*, 2008.

[42] Lellouch, G., A. K. Mishra, and M. Inggs, "Design of OFDM Radar Pulses Using Genetic Algorithm Based Techniques," *IEEE Transactions on Aerospace and Electronic Systems,* Vol. 52, No. 4, 2016.

[43] Sen, S., G. Tang, and A. Nehorai, "Multiobjective Optimization of OFDM Radar Waveform for Target Detection," *IEEE Transactions on Signal Processing,* Vol. 59, No. 2, 2011.

[44] Jouny, I., "Radar Target Identification Using Genetic Algorithms," in *Automatic Target Recognition,* 1998.

[45] Smith-Martinez, B., A. Agah, and J. Stiles, "A Genetic Algorithm For Generating Radar Transmit Codes to Minimize the Target Profile Estimation Error," *Journal of Intelligent Systems,* Vol. 22, No. 4, 2013.

[46] Zhou, C., F. Liu, and Q. Liu, "An Adaptive Transmitting Scheme for Interrupted Sampling Repeater Jamming Suppression," *Sensors (Basel),* Vol. 17, No. 11, 2017.

[47] Kirkpatrick, S., C. Gelatt Jr., and M. Vecchi, "Optimization by Simulated Annealing," *Science,* Vol. 220, No. 4598, 1983.

[48] Boopalan, N., A. Ramasamy, and F. Nagi, "Faulty Antenna Detection in a Linear Array Using Simulated Annealing Optimization," *Indonesian Journal of Electrical Engineering and Computer Science,* Vol. 19, No. 3, 2020.

[49] Kaur, K., M. Rattan, and M. Patterh, "Optimization of Cognitive Radio System Using Simulated Annealing," *Wireless Personal Communications,* Vol. 71, 2013.

[50] White, R., "Simulated Annealing Algorithm for Radar Cross-Section Estimation and Segmentation," in *Applications of Artificial Neural Networks V,* International Society for Optics and Photonics, Vol. 2243, 1994.

[51] Liu, Z., et al., "Error Minimizing Jammer Localization Through Smart Estimation of Ambient Noise," in *International Conference on Mobile Ad-Hoc and Sensor Systems,* 2012.

[52] Wang, Y., and S. Zhu, "Main-beam Range Deceptive Jamming Suppression with Simulated Annealing FDA-MIMO Radar," *IEEE Sensors Journal,* Vol. 20, No. 16, 2020.

[53] Zhao, J., X. Guan, and X. Li, "Power Allocation Based on Genetic Simulated Annealing Algorithm in Cognitive Radio Networks," *Chinese Journal of Electronics,* Vol. 22, No. 1, 2012.

[54] Yang, G., et al., "W-Band MIMO Radar Array Optimization and Improved Back-Projection Algorithm for Far-Field Imaging," in *International Conference on Infrared, Millimeter, and Terahertz Waves,* 2019.

[55] Sayin, A., E. G. Hoare, and M. Antoniou, "Design and Verification of Reduced Redundancy Ultrasonic MIMO Arrays Using Simulated Annealing & Genetic Algorithms," *IEEE Sensors Journal,* Vol. 20, No. 9, 2020.

[56] Liu, Y., et al., "A Self-Learning Method for Cognitive Engine Based on CBR and Simulated Annealing," in *Advanced Materials and Engineering Materials,* Vol. 457, 2012.

[57] Rubinstein, R., and D. Kroese, *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning,* Springer, 2004.

[58] de Boer, P.-T., et al., *A Tutorial on the Cross-Entropy Method,* 2003. http://web.mit.edu/6.454/www/www_fall_2003/gew/CEtutorial.pdf.

[59] Amos, B., and D. Yarats, "The Differentiable Cross-Entropy Method," in *ICML,* 2020.

[60] Joseph, A., and S. Bhatnagar, "Revisiting the Cross Entropy Method with Applications in Stochastic Global Optimization and Reinforcement Learning," in *ICAI,* 2016.

[61] Kroese, D., K.-P. Hui, and S. Nariai, "Network Reliability Optimization via the Cross-Entropy Method," *IEEE Transactions on Reliability,* Vol. 56, No. 2, 2007.

[62] Heegaard, P., B. Helvik, and O. Wittner, "The Cross Entropy Ant System for Network Path Management," *Telektronikk,* 2008.

[63] Bian, L., "Parameters Analysis to Pattern Null Placement Based on the Cross Entropy Method," *Physics Procedia,* Vol. 24, No. B, 2012.

[64] Naeem, M., A. S. Khwaja, A. Anpalagan, et al., "Green Cooperative Cognitive Radio: A Multiobjective Optimization Paradigm," *IEEE Systems Journal,* Vol. 10, No. 1, 2016.

[65] Minvielle, P., et al., "Sparse Antenna Array Optimization with the Cross-Entropy Method," *IEEE Transactions on Antennas and Propagation,* Vol. 59, No. 8, 2011.

[66] Heegaard, P., et al., *Distributed Asynchronous Algorithm for Cross-Entropy-Based Combinatorial Optimization,* 2003. Online: https://tinyurl.com/ce-ants-2003.

[67] Helvik, B., and O. Wittner, "Using the Cross-Entropy Method to Guide/Govern Mobile Agent's Path Finding in Networks," in *Workshop on Mobile Agents for Telecommunication Applications,* 2001.

[68] Maudsley, D., "A Theory of Meta-Learning and Principles of Facilitation: An Organismic Perspective," Ph.D. dissertation, University of Toronto, Ontario, Canada, 1979.

[69] Vanschoren, J., "Meta-Learning," in *Automated Machine Learning,* Springer, 2019.

[70] Rendell, L., R. Seshu, and D. Tcheng, "More Robust Concept Learning Using Dynamically-Variable Bias," in *Workshop on Machine Learning,* 1987.

[71] J. Schmidhuber, "Evolutionary Principles in Self-Referential Learning, or on Learning How to Learn: The Meta-Meta-... Hook," M.S. thesis, Technische Universität München, Munich, Germany, 1987.

[72] Bengio, Y., S. Bengio, and J. Cloutier, "Learning a Synaptic Learning Rule," in *IJCNN,* Vol. ii, 1991.

[73] Andrychowicz, M., et al., "Learning to Learn by Gradient Descent By Gradient Descent," in *NeurIPS,* 2016.

[74] Rusu, A., et al., "Metalearning with Latent Embedding Optimization," in *ICLR,* 2019.

[75] Zintgraf, L., et al., "Fast Context Adaptation via Metalearning," in *ICLR,* 2019.

[76] Chen, Y., et al., "Learning to Learn Without Gradient Descent by Gradient Descent," in *ICML,* vol. 70, 2017.

[77] Li, K., and J. Malik, "Learning to Optimize," in *ICLR,* 2017.

[78] Vilalta, R., and Y. Drissi, "A Perspective View and Survey of Metalearning," *Artificial Intelligence Review,* Vol. 18, 2005.

[79] Arpaci-Dusseau, R. H., and A. C. Arpaci-Dusseau, *Operating systems: Three Easy* Pieces, Arpaci-Dusseau Books, LLC, 2018.

[80] Kilic, S., and O. Ozkan, "Modeling and Optimization Approaches for Satellite Broadcast Scheduling Problem," *IEEE Transactions on Engineering Management,* 2019.

[81] Doron Rainish, D., and A. Freedman, *Air Interface for Low Power Operation of a Satellite Terminal,* 2015. Online: https://www.satixfy.com/vlnsr-implementation-for-mobile/.

[82] El Gamal, A., et al., "Energy-Efficient Scheduling of Packet Transmissions Over Wireless Networks," in *Computer and Communications Societies,* IEEE, Vol. 3, 2002.

[83] Dahlman, E., S. Parkvall, and J. Sköld, "Chapter 11: Uplink Physical Layer Processing," in *4G LTE/LTE-Advanced for Mobile Broadband,* Academic Press, 2011.

[84] Pattanayak, P., and P. Kumar, "Computationally Efficient Scheduling Schemes for Multiple Antenna Systems Using Evolutionary Algorithms and Swarm Optimization," in *Evolutionary Computation in Scheduling,* 2020.

[85] Grilo, A., et al., "Electronic Protection and Routing Optimization of MANETs Operating in an Electronic Warfare Environment," *Ad Hoc Networks,* Vol. 5, No. 7, 2007.

[86] Kim, S., "Adaptive MANET Multipath Routing Algorithm Based on the Simulated Annealing Approach," *The Scientific World Journal,* 2014.

[87] Bollapragada Subrahmanya, V., and H. Russell, "RMTS: A Novel Approach to Transmission Scheduling in Ad Hoc Networks by Salvaging Unused Slot Transmission Assignments," *Wireless Communications and Mobile Computing,* 2018.

[88] Amouris, K., "Position-Based Broadcast TDMA Scheduling for Mobile Ad-Hoc Networks (MANETs) with Advantaged Nodes," in *MILCOM,* 2005.

[89] Salih, S., M. Suliman, and A. Mohammed, "A Novel Spectrum Sensing Scheduler Algorithm for Cognitive Radio Networks," in *International Conference on Computing, Electrical and Electronic Engineering,* 2013.

[90] Zhu, L., et al., "The Design of Scheduling Algorithm for Cognitive Radio Networks Based on Genetic Algorithm," in *IEEE International Conference on Computational Intelligence Communication Technology,* 2015.

[91] Liang, J.-C., and J.-C. Chen, "Resource Allocation in Cognitive Radio Relay Networks," *Selected Areas in Communications,* Vol. 31, No. 3, 2013.

[92] Sodagari, S., "Real-Time Scheduling for Cognitive Radio Networks," *Systems Journal,* Vol. 12, No. 3, 2017.

[93] Bayhan, S., and F. Alagoz, "Scheduling in Centralized Cognitive Radio Networks for Energy Efficiency," *Transactions on Vehicular Technology,* Vol. 62, No. 2, 2013.

[94] Zhang, W., et al., "Distributed Stochastic Search and Distributed Breakout: Properties, Comparison and Applications to Constraint Optimization Problems in Sensor Networks," *Artificial Intelligence,* Vol. 161, No. 1-2, 2005.

[95] Ramzan, M., et al., "Multiobjective optimization for Spectrum Sharing in Cognitive Radio Networks: A Review," *Pervasive and Mobile Computing,* Vol. 41, 2017.

[96] Zhang, H., et al., "A Hybrid Adaptively Genetic Algorithm for Task Scheduling Problem in the Phased Array Radar," *European Journal of Operational Research,* 2019.

[97] Sahin, S., and T. Girici, "A Method for Optimal Scheduling of Active Electronically Scanned Array (AESA) Antennas," 2019.

[98] Li, X., et al., "A Scheduling Method of Generalized Tasks for Multifunctional Radar Network," in *International Conference on Control, Automation and Information Sciences,* IEEE, 2019.

[99] Orman, A., et al., "Scheduling for a Multifunction Phased Array Radar System," *European Journal of Operational Research,* Vol. 90, No. 1, 1996.

[100] Shaghaghi, M., R. Adve, and Z. Ding, "Multifunction Cognitive Radar Task Scheduling Using Monte Carlo Tree Search and Policy Networks," *IET Radar, Sonar and Navigation,* Vol. 12, No. 12, 2018.

[101] Song, H., et al., "A POMDP Approach for Scheduling the Usage of Airborne Electronic Countermeasures in Air Operations," *Aerospace Science and Technology,* Vol. 48, 2016.

[102] Vaughan, I., and L. Clarkson, "Optimal Periodic Sensor Scheduling in Electronic Support," in *Defence Applications of Signal Processing,* 2005.

[103] Ren, Y., et al., "A Novel Cognitive Jamming Architecture for Heterogeneous Cognitive Electronic Warfare Networks," in *Information Science and Applications,* Vol. 621, Springer, 2020.

[104] Zhang, Y., and G. Shan, "A Risk-Based Multisensor Optimization Scheduling Method for Target Threat Assessment," *Mathematical Problems in Engineering,* 2019.

[105] Rajpathak, D., "Intelligent Scheduling—A Literature Review," Knowledge Media Institute, The Open University (United Kingdom), Tech. Rep. KMI-TR-119, 2001.

[106] Xie, G., R. Li, and K. Li, "Heterogeneity-Driven End-to-End Synchronized Scheduling for Precedence Constrained Tasks and Messages on Networked Embedded Systems," *Journal of Parallel and Distributed Computing,* Vol. 83, 2015.

[107] Balli, U., et al., "Utility Accrual Real-Time Scheduling Under Variable Cost Functions," *IEEE Transactions on Computers*, Vol. 56, No. 03, 2007.

[108] Orhean, A., F. Pop, and I. Raicu, "New Scheduling Approach Using Reinforcement Learning for Heterogeneous Distributed Systems," *Journal of Parallel and Distributed Computing,* Vol. 117, 2018.

[109] Yang, R., et al., "Intelligent Resource Scheduling at Scale: A Machine Learning Perspective," in *Symposium on Service-Oriented System Engineering,* IEEE, 2018.

[110] Ye, Y., et al., *A New Approach for Resource Scheduling with Deep Reinforcement Learning*, 2018. Online: https://arxiv.org/abs/1806.08122.

[111] Mirshekarian, S., and D. Sormaz, "Machine Learning Approaches to Learning Heuristics for Combinatorial Optimization Problems," *Procedia Manufacturing,* Vol. 17, 2018.

[112] Gandomi, A., et al. (Eds.), *Evolutionary Computation in Scheduling,* Wiley, 2020.

[113] Barták, R., R. A. Morris, and K. B. Venable, "An Introduction to Constraint-Based Temporal Reasoning," *Synthesis Lectures on Artificial Intelligence and Machine Learning,* Vol. 8, No. 1, 2014.

[114] Dean, T., and M. Boddy, "An Analysis of Time-Dependent Planning," in *AAAI,* 1988.

[115] Horvitz, E., "Reasoning About Beliefs and Actions Under Computational Resource Constraints," in *Workshop on Uncertainty in Artificial Intelligence,* 1987.

[116] Simon, H., "Rational Choice and the Structure of the Environment," *Psychological Review,* Vol. 63, No. 2, 1956.

[117] Brooks, R., T. Arbel, and D. Precup, "Fast Image Alignment Using Anytime Algorithms," in *IJCAI,* 2007.

[118] Kim, Y., M. Krainin, and V. Lesser, "Application of Max-Sum Algorithm to Radar Coordination and Scheduling," in *Workshop on Distributed Constraint Reasoning,* 2010.

[119] Zilberstein, S., "Operational Rationality Through Compilation of Anytime Algorithms," Ph.D. dissertation, University of California, Berkeley, CA, 1993.

[120] Zilberstein, S., "Using Anytime Algorithms In Intelligent Systems," *AI Magazin*e, 1996.

[121] Svegliato, J., K. H. Wray, and S. Zilberstein, "Meta-Level Control of Anytime Algorithms with Online Performance Prediction," in *IJCAI,* 2018.

[122] Gagliolo, M., and J. Schmidhuber, "Learning Dynamic Algorithm Portfolios," *Annals of Mathematics and Artificial Intelligence,* Vol. 47, 2006.

[123] López-Ibáñeza, M., and T. Stützlea, "Automatically Improving the Anytime Behaviour of Optimisation Algorithms," *European Journal of Operational Research,* Vol. 235, No. 3, 2014, Extended version at https://core.ac.uk/download/pdf/208141673.pdf.

[124] Smith, S. F., et al., "Robust Allocation of RF Device Capacity for Distributed Spectrum Functions," *Journal of Autonomous Agents and Multiagent Systems,* Vol. 31, No. 3, 2017.

[125] Gillen, M., et al., "Beyond Line-of-Sight Information Dissemination for Force Protection," in *MILCOM,* 2012.

[126] Fioretto, F., E. Pontelli, and W. Yeoh, "Distributed Constraint Optimization Problems and Applications: A Survey," *Journal of Artificial Intelligence Research,* Vol. 61, No. 1, 2018.

[127] Faltings, B., "Distributed Constraint Programming," in *Handbook of Constraint Programming,* Elsevier, 2006.

[128] Shoham, Y., and K. Leyton-Brown, *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*, Cambridge University Press, 2009.

[129] Moallemi, C., and B. Van Roy, "Consensus Propagation," *Transactions on Information Theory,* Vol. 52, No. 11, 2006.

[130] Molzahn, D., et al., "A Survey of Distributed Optimization and Control Algorithms for Electric Power Systems," *IEEE Transactions on Smart Grid,* Vol. 8, No. 6, 2017.

[131] Duffy, K., C. Bordenave, and D. Leith, "Decentralized Constraint Satisfaction," *IEEE/ACM Transactions on Networking,* Vol. 21, No. 4, 2013.

[132] Di Lorenzo, P., and S. Barbarossa, "Distributed Resource Allocation in Cognitive Radio Systems Based on Social Foraging Swarms," in *International Workshop on Signal Processing Advances in Wireless Communications,* IEEE, 2010.

[133] Zhang, W., et al., "Distributed Stochastic Search and Distributed Breakout: Properties, Comparison and Applications to Constraint Optimization Problems in Sensor Networks," *Artificial Intelligence,* Vol. 161, No. 1, 2005.

[134] Stankovic, M., S. Stankovic, and K. Johansson, "Distributed Time Synchronization in Lossy Wireless Sensor Networks," in *IFAC Workshop on Estimation and Control of Networked Systems,* 2012.

[135] Yan, H., et al., "Software Defined Radio Implementation of Carrier and Timing Synchronization for Distributed Arrays," in *Aerospace Conference,* 2019.

[136] Kim, M., H. Kim, and S. Lee, "A Distributed Cooperative Localization Strategy in Vehicular-to-Vehicular Networks," *Sensors,* Vol. 20, No. 5, 2020.

[137] Wolpert, D., "The Lack of a priori Distinctions Between Learning Algorithms," *Neural Computation,* 1996.

[138] Wolpert, D., and W. Macready, "No Free Lunch Theorems for Optimization," *Transactions on Evolutionary Computation,* Vol. 1, No. 67, 1997.

6

Electronic Battle Management

An EBM system plans how many platforms to deploy, which resources each gets, and where they will go. It coordinates activity across EMSO domains, and with broader issues such as logistics. EW plans should identify the desired electromagnetic profile; identify missions, tasks, and resources; evaluate threats; and meet the constraints and goals of relevant policies [1]. Factors affecting the plan include available assets, desired effects, placement limitations (altitude, range, time, or loads), frequency deconfliction, anticipated EW missions from other services, and authentication requirements [2]. Figure 6.1 captures some of the inputs and outputs of an EBM system, including the desired effects, platform limitations, rules of engagement, and anticipated EW missions from other services.

DoD doctrine explains the traditional approach to EW planning, noting that "Joint EW is centrally planned and directed and decentrally executed…" [1], and that "…this planning requires a multidisciplined approach with expertise from operations (ground, airborne, space), intelligence, logistics, weather, and information." [2]

Moving to an *automated* EBM planning system creates a more interactive system that can respond to events as they occur during the mission. As noted in Chapter 5, automated planning activities overlap optimization and scheduling and will become fully interactive integrated systems in the future. Section 6.1 presents AI planning approaches, discussing issues such as uncertainty, resource allocation, and multiple timescales. Section 6.2 touches on game theory approaches for team coordination and reasoning about adversarial actions. Section 6.3 discusses the human-machine interface (HMI), including how to leverage human expertise within the system, extract human-level goals for optimization, and explain decisions to human users. (Section 7.2 describes how to update plans during a mission.)

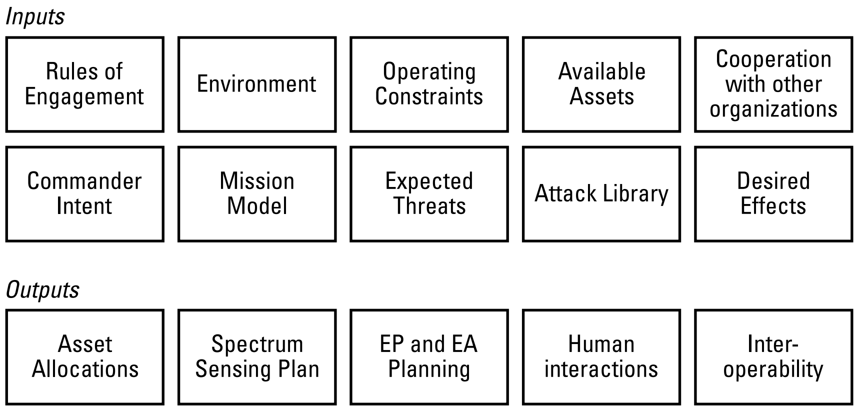


Figure 6.1 To support deliberative EBM planning, inputs include relatively static descriptions of the mission, and outputs include high-level tasking. (As planning becomes more automated, dynamic inputs will also become relevant.)

In a complete EW system, there will be multiple decision-makers. Planners might be task-specific, algorithm-delineated, and geographically or temporally separated. This separation of concerns makes system design significantly easier. Examples of task-specific planners include ROGUE, which had a task planner and a route planner [3], and Tactical Expert Mission Planner (TEMPLAR), which had four planners (Example 6.1).

Example 6.1 TEMPLAR generates air tasking orders.

Tactical Expert Mission Planner (TEMPLAR)

There are four distinct planning modules in TEMPLAR:

- The package planner, which generates mission lines on Target Planning Worksheets.
- The flow planner, which generates mission lines on unit schedules for [Close Air Support] CAS and [Defensive Counter Air] DCA missions.
- The air-air refueling planner, which generates mission lines on Refueling Planning Worksheets.
- The mission line planner, which fills in mission number, call sign, SIF code, and other information on existing mission lines.

—Siska, Press, and Lipinski [4]

6.1 Planning

Planning operates over broader scope and with a slower time envelope than optimization or scheduling. Planning evaluates actions over longer time horizons, and handles broader asset and resource types. The Hubble Space Telescope, which is automatically controlled with a planning and scheduling system built in the early 1990s, highlights the tight coupling between a planner and a scheduler (Example 6.2).

Planning focuses on the causal reasoning for finding a sequence of actions needed to achieve the goal(s). The plan is usually a partially ordered set of actions that does not specify resources or a schedule; scheduling assigns time and resources to the plan. Figure 6.2 shows how different challenges drive selection of a planning approach.

Classical planning takes as input a description of the current situation (an initial state), a set of actions, and a description of the goal(s), and generates as output a sequence of actions that lead from the current situation to the goals. *Probabilistic planning* handles nondeterministic and partially observable environments. *Temporal planning* handles actions with duration, or can be concurrent; temporal planning techniques usually handle resources, because resource use is frequently temporal in nature (e.g., a resource may be used by at most one action at a time). *Conditional planning,* or contingent planning, generates plans with conditional branches: Rather than a sequence of actions, it generates a contingency plan that chooses actions based on perception of the environment, such as the one illustrated in Figure 6.3. The more mission-critical the failure, the more important it is to prepare in advance for contingencies [7].

Several good books exist on planning [8–10]. Benaskeur et al. [7] describe different planning approaches for naval-engagement scenarios. The International

Example 6.2 An AI planner and scheduler controls astronomy tasks on the Hubble Space Telescope.

The Hubble Space Telescope was one of the first fielded, large-scale, automated planning and scheduling systems [5, 6]. The *Heuristic Scheduling Testbed System* (HSTS) constructs observation schedules based on goal expansion (planning) and resource allocation (scheduling). Astronomers formulate observation programs with temporal constraints specifying the collection of light from a celestial object with one of the six scientific instruments. The resulting plan must accomplish tasks for multiple astronomers. Multiple layers of abstraction allow the planner to manage the combinatorics of the problem: The abstract level manages telescope availability, reconfiguration time, and target visibility, while the detail level manages specific exposures.

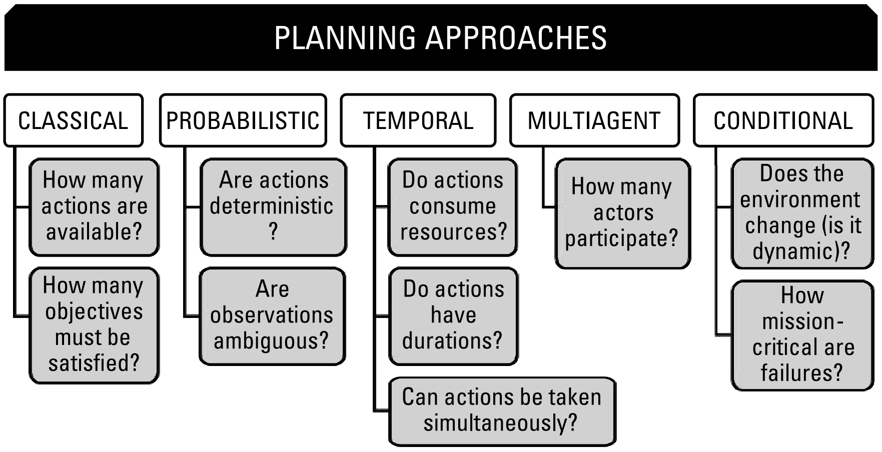

PLANNING APPROACHES
CLASSICAL
How many actions are available?
How many objectives must be satisfied?
PROBABILISTIC
Are actions deterministic?
Are observations ambiguous?
TEMPORAL
Do actions consume resources?
Do actions have durations?
Can actions be taken simultaneously?
MULTIAGENT
How many actors participate?
CONDITIONAL
Does the environment change (is it dynamic)?
How mission-critical are failures?

Figure 6.2 Different problem domain challenges require different considerations in the planning approach (not mutually exclusive).

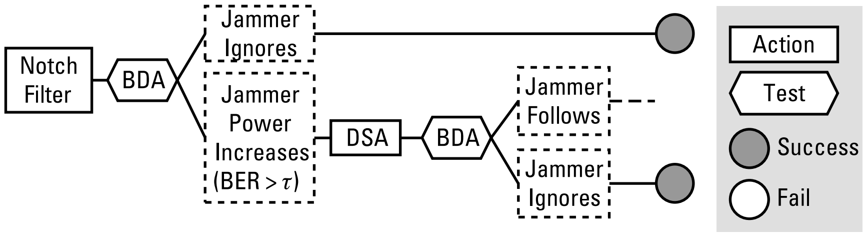


Figure 6.3 A contingent plan generates actions with conditional branches based on expected outcomes. In simple cases such as this plan for a tone jammer (Table 5.1), they look like decision trees or rules systems.

Planning Competition (IPC) [11], held roughly biennially since 1998, runs tracks for different types of planners on realistic (and sometimes real) real-world problems, such as satellite-based Earth observation, wildlife poaching, and data-network task-scheduling. Notably, the source code from these planners is public. Yuen [12] presents a graphic showing the evolution of IPC competition tracks and the important trade-offs to consider for different types of problems. Yuen then presents a detailed cyber red-teaming use case.

Search and planning algorithms are judged on the basis of *completeness*, *optimality*, *time complexity*, and *space complexity* [8]. This chapter presents search algorithms that search through the solution space in a *deliberative* manner. Section 5.1.2 presents *randomized* search engines and uses this same criteria to evaluate their appropriateness to EW.

6.1.1 Planning Basics: Problem Definition, and Search

A problem can be defined formally with the following components:

- An initial state, $s_0 \in S$;
- A set of possible actions, A;
- A description of what each action does, as a function that returns the state $s' \in S$ that results from doing action a in state s, $\rho : (s,a) \rightarrow s'$,
- A goal state, $s_g \in S$;
- A cost function that assigns a cost to each step between states, $c(s,a,s')$.

A solution to the planning problem P is to find a sequence of actions $\pi = (a_1,a_2,...,a_n)$ from s_0 that produces a sequence of states $(s_1,s_2,...,s_g)$. An optimal solution has the lowest cost path from s_0 to s_g.

Classical planning domains are commonly represented in the Planning Domain Definition Language (PDDL) [13]. PDDL syntax encodes the domain: states, actions, and compound actions. States are represented as a conjunction of positive literals *predicate(arg1,...,argN)*, e.g., s = *in(pilot,airplane) AND has fuel(airplane).* Actions have *preconditions* and *effects;* An action a is applicable in a state s only if its precondition literals $p_i \in s$ are true. Several variants of PDDL capture planning problems of different natures and complexities, including constraints, conditional effects, numeric and temporal features, and action costs.

Search algorithms look for the sequence of actions, by creating a search tree whose root is the initial state. Each *branching step* expands the current state by applying each legal action to the state, thereby generating new states. As an example, a simple robot moving in a grid-world can move to any adjacent grid square, resulting in four possible future states. (A robot moving on a flat surface has a 2D infinite space of future states. An aerial drone has a 3D infinite space.) Continuous values can be discretized or presented with statistics such as mean and variance. *Pruning steps* eliminate illegal or repeated states to reduce the search effort.

Uninformed algorithms build this search tree without guidance from the domain, expanding nodes until they reach the goal. Common strategies include breadth-first search, depth-first search, iterative deepening, and bidirectional search. Given the complexity of EW, we do not present these in more detail here, because domain knowledge is critical to finding solutions.

Informed search strategies use *heuristics* to guide the search, thus making search more efficient or more effective [14]. Heuristic rules can be generated by experts, or created by ML from empirical data. One of the best-known heuristic algorithms is A*, a best-first search that expands the node n with the lowest estimated path cost [i.e., $g(n) + h(n)$, where $g(n)$ is the cost from the initial state s_0 to n, and $h(n)$ is the estimated cost from n to the goal state s_g].

Nearly every successful planning system for real-world domains uses domain-specific heuristics.

The randomized metaheuristic search algorithms of Section 5.1.2 are one approach to finding solutions in a complex space, and many of the schedulers in Section 5.2 also use heuristics. Typically, each possible plan is encoded as a sequence of actions that can be manipulated by the randomized search engine. For example, plans become chromosomes for GAs [15, 16], and graphs capture hazards in fire-escape routes for ACO [17].

6.1.2 Hierarchical Task Networks

Hierarchical task networks (HTN) planning embodies a different view of planning [10, 18–20]: Instead of searching for goals by changing the state of the world model, the HTNs are a set of abstract tasks to be done, and a set of methods for each task that represent different ways in which they can be carried out. The dependency among actions is represented with hierarchically structured networks.

HTN planners develop plans at increasing levels of abstraction. HTNs are the most widely used form of planning in practical applications for two reasons. First, they effectively manage computational complexity. Second, they naturally correspond to how humans think about problems. This characteristic means that humans can describe relationships among tasks more easily, and the user interface inherently explains task progress.

An HTN planner reasons over a *task network* representing the problem to be solved. The network is a collection of *tasks* to carry out and associated *constraints,* such as their order and what preconditions must be true, as illustrated in Figure 6.4. There are three types of tasks:

- *Goal tasks,* capturing the desired end state;
- *Primitive tasks,* which are actions that can be directly executed in the state, with associated preconditions and expected effects;
- *Compound tasks*, showing how to accomplish goals with primitive tasks. A *method* shows how to decompose a compound task into partially ordered subtasks (primitive or compound).

A solution to an HTN problem is thus an executable sequence of primitive tasks that can be obtained from the initial task network by decomposing compound tasks into their set of simpler tasks, and by inserting ordering constraints.

HTN planners are used in many problem domains [20], including medicine, production planning, logistics, crisis management, air campaign planning, naval command and control [22], aircraft carrier scheduling [23], and unmanned

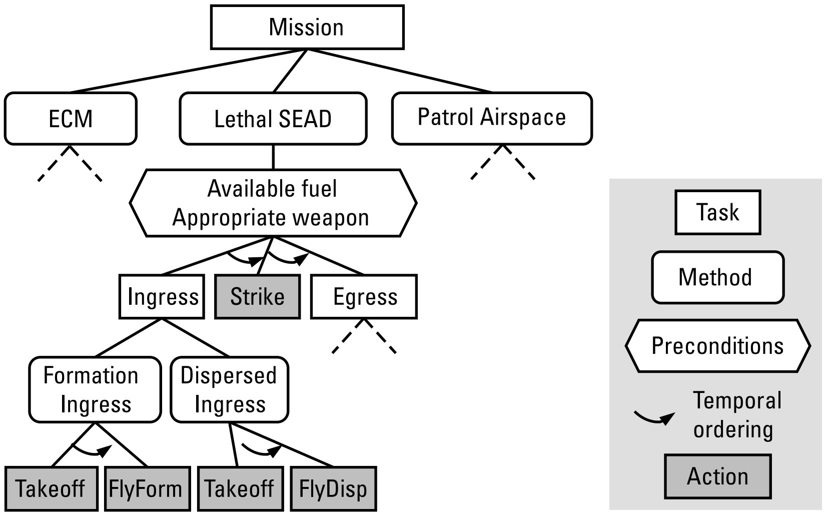


Figure 6.4 HTNs contain tasks, their preconditions, and associated constraints such as ordering. The mission goal has three possibly independent tasks: ECM, suppression of enemy air defense (SEAD), or patrol airspace. The SEAD method decomposes into three ordered tasks, ingress, strike, and egress. (Derived from [21].)

combat robot teams [21]. Example 6.3 presents a more detailed example for coalition battle planning. Because HTN planners encode domain knowledge within the task networks, they can solve much larger planning problems, and solve them more quickly, than domain-independent planners.

HTNs can be translated to other planning representations to improve their performance, for example PDDL [27], MDPs [28], and temporal planners [29]. In this way, the domain-independent planners can leverage domain knowledge encoded in the HTN. The ability to use domain knowledge is essential for solving large problems in complex domains.

6.1.3 Action Uncertainty

There are several approaches for planning in domains where actions have nondeterministic outcomes, including conditional planning [30, 31], graph planning [32], stochastic satisfiability [33, 34], first-order decision diagrams [35] and MDPs. Because MDPs are the most widely used of these, they are the focus of this section. (Section 6.1.4 discusses the orthogonal issue of *information* uncertainty.)

MDPs are a framework for modeling and guiding sequential DM under uncertainty [8, 36, 37]. Markov models describe systems that transition probabilistically between a finite set of states. When the domain is only partially observable, planning corresponds to a POMDP.

MDPs augment the problem definition of Section 6.1.1 with *transition probabilities* $P(s'|s,a)$ that denote the probability of reaching state $s' \in S$ if action

Example 6.3 The Course-of-Action Display and Evaluation Tool (CADET) integrates HTN planning with scheduling to create coalition battle plans.

CADET assists military planners in coalition environments [24–26]. CADET is a knowledge-based tool capable of producing automatically (or with human guidance) battle plans with a realistic degree of detail and complexity. CADET has been integrated with several battle management systems for DARPA and the U.S. Army [25].

The human planner defines the key goals for a tactical course of action (COA), and CADET expands them into a detailed plan/schedule of the operation. CADET includes technology for interleaved planning, scheduling, routing, attrition, and resource consumption. CADET models assets and tasks, handles the adversarial environment, coordinates team efforts, and supports autonomous action (through commander's intent).

The authors state, "The integration of planning and scheduling is achieved via an algorithm for tightly interleaved incremental planning and scheduling. The HTN-like planning step produces an incremental group of tasks by applying domain-specific 'expansion' rules to those activities in the current state of the plan that require hierarchical decomposition. The scheduling step performs temporal constraint propagation (both lateral and vertical within the hierarchy) and schedules the newly added activities to the available resources and time periods." [25]

$a \in A$ is done in state $s \in S$. A *reward R* captures the expected reward when action a is done in state s. If models of adversary behavior are available from ES, they can be incorporated into the transition function to reflect the fact that the state of threats (which is part of each $s_i \in S$) is affected by both friendly and adversarial actions.

The policy π specifies what agents should do in every state; $\pi(s)$ is the action recommended by π for state s. In a *stationary policy,* once the policy is found, the action is fixed for each state; in a *nonstationary policy,* the policy changes over time (i.e., actions depend on history).

Because the environment is stochastic, the quality of a policy is measured by its *expected utility* from executing it many times. An optimal policy π^* is one that yields the highest expected utility. The planning problem is to find a good (or optimal) policy. Two standard algorithms that return a stationary optimal policy after a finite number of iterations are *value iteration* and *policy iteration* [38].

The most common utility function for MDPs is that of *discounted rewards,* where the utility of a state sequence is discounted over time, for the discount factor $\gamma \in [0,1]$. The expected utility of a policy π starting in state s_0 is:

$$U^{\pi}(s_0) = E\left[\sum_{i-0}^{\infty} \gamma^i R(s_i, a_i)\right]$$

where $a_i = \pi(s_i)$. $\pi^*(s_i)$ is then the action that maximizes the expected utility of the subsequent state s_{i+1}.

A variety of action-choice functions that may be appropriate for the mission include:

- *Maximax:* Choose the action with the largest expected reward, even if it is unlikely (optimistic; risky).
- *Maximum likelihood:* Pick the action with the largest expected value by combining probability of outcome with expected reward.
- *Maximin:* Choose the action with the maximum of the minimum values (pessimistic; risk-averse).
- *Minimax regret:* Choose the action that minimizes worst-case regret, which is the opportunity loss for making the wrong decision (risk-averse).

A *myopic best response* selects the reward-maximizing action in each state, ignoring any possible future reward (i.e., $\gamma = 0$).

MDP planning is very computationally inefficient [38–40], but MDPs are the most widely used framework to formulate probabilistic plans. For this reason, much research focuses on how to improve MDP planning performance [28, 41–47].

Decentralized POMDPs are a common solution for a team of coordinating nodes [48] but are NEXP-complete [49] and may require doubly exponential time to solve in the worst case. Moreover, states and actions of node n include knowledge and actions from nodes n'. In practice, nodes reduce problem size by using heuristics or exploiting problem structure. *Locality of interaction,* for example, assumes that nodes only need to coordinate with a limited subset of other nodes (i.e., that nodes do not need to know every detail of every other node) [48, 50, 51]. Section 5.4 presents additional distributed optimization approaches.

POMDPs are also the most common representation used in RL settings (Chapter 7), where RL techniques learn the probabilities and/or rewards of the model. There are three key reasons why RL is not synonymous with MDPs: (1) MDPs are models that specify a policy, while RL takes actions to update that policy, (2) in MDP planning, the planner must decide when to terminate planning, while RL terminates only when the mission is over [42], and (3) other ML models can be used for RL.

6.1.4 Information Uncertainty

In complex environments, information is rarely known perfectly. DM engines must incorporate and reason about the uncertainty in the information leading to decisions. With this information, the planner can use active sensing and communication to improve the information quality or certainty. Approaches to managing information uncertainty include:

- *Dempster-Shafer theory,* which measures *belief* in a fact and computes the probability that the evidence supports the proposition [52, 53];
- *Fuzzy logic,* which represents the truth value of facts [54];
- *Argumentation,* which explicitly constructs the relationships connecting evidence and conclusions, making it possible to reason about these relationships directly [55], for example, to resolve conflicting information or to allow planners to reason about causality [56].

These approaches support evaluating the utility of information.

Information utility enables being deliberative about whether, when, with whom, and what to communicate or sense.

Table 6.1 summarizes these active sensing/communication actions. The concepts of provenance and credibility are closely connected (Section 8.1.3), as is data fusion (Section 4.3). Table 6.2 suggests some possible active sensing actions that the system might take depending on the desired information. After performing an active sensing action, the execution monitor must determine whether the effect was simply an inherent function of the environment, or whether the sensing caused the observed behavior.

Note also that communication can be interplatform, and also intraplatform (i.e., among multiple decision-makers on a single platform). This section focuses

Table 6.1
Active Sensing/Communication Actions

Factor	Active Sensing	Deliberate Communication
Why	Improve info quality	Improve info quality
Who	Collaborative sensing	With whom to communicate
What	What to gather	What to transmit; abstraction level
Where	Spatial and spectral	Team geography
When	Sensing schedule	Criticality of information
How	What sensors to use	Implicit or explicit interactions

Table 6.2
Sensing Actions Elicit Information About Unknown Emitters in the Environment

Domain	Desired Info	Active Sensing Action
Radar	Number of radar(s) Location of radar(s)	Generate ghosts Directional probes for blind spots
Radar	Max detection range sensitivity G-force limits	Move toward target Signal amplification, PRI and Doppler Increase replay rate

Communications	Node response time;
	network recovery time;
	network recovery methods
		Surgical packet jamming
		Self-collapse trigger attacks
		Piecewise attack during recovery

Communications	Network critical nodes	Isolate nodes
Communications and radar	ES recognition ability EP methods EA agility and precision	Honeypot sequences Camouflage signatures

on interplatform communication, but similar reasoning applies to the simpler problem of intraplatform communication.

When time and bandwidth are available, nodes can exchange observations and decisions to improve or recalibrate their understanding of the situation. Data fusion resolves ambiguities, reduces uncertainty in the models, and can accelerate the rate of identifying previously unknown threats. For example, a team of nodes can combine estimates of threats that can only be seen with fleeting glimpses. Figure 6.5 illustrates the potential impact of communications on planning for an integrated air defense system (IADS) task. As communications become more available, more of the lower-priority threats can be neutralized.

In the broad context of performing distributed optimization under limited communications, traditional approaches assume that communications is safe and plentiful. In EW, this assumption is false: Communications are limited or denied due to jamming, mobility disconnects, or deliberate choices to reduce

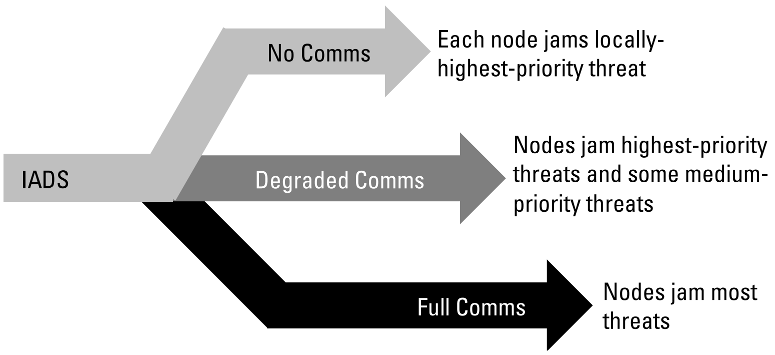

IADS
No Comms
Each node jams locally-highest-priority threat
Degraded Comms
Nodes jam highest-priority threats and some medium-priority threats
Full Comms
Nodes jam most threats

Figure 6.5 Task accomplishment depends on available communications for coordination.

RF emissions. Reasoning about when to communicate can achieve near-optimal results [57–61]. The system can minimize the operation and computational costs of communication based on a quantitative estimate of the benefit of communicating or sensing. Some of the DM's choices and tradeoffs include:

- *Types of information:* Surprising outcomes; high-penalty outcomes; critical coordination; ambiguous or conflicting observations; uncertain variables, predictions, or inferences; unfamiliar concepts; assumptions (testing hypotheses).
- *With whom:* Nodes in coordinated actions; nodes with temporal dependencies; nearby nodes.
- *Level of abstraction:* Different state and action abstractions depending on task.
- *Costs:* Energy; exposure; congestion; coordination effort; cost of being wrong.

The level of detail at which nodes communicate determines the level of coordination they can achieve if/when needed and the kind of coordinated actions they can execute. It also impacts the computational and communication cost of solving the problem. (Even in settings where instantaneous and cost-free communication is available, broadcasting all information to all nodes doesn't scale [48].) A richer communication language is informative (e.g., incorporating detailed current state [62]), leading to better coordinated decisions, but it results in a larger problem size. In a simpler language, the nodes can exchange commitments over actions with nonlocal effects [63]. In the simplest language, a node can signal the completion of an action with nonlocal effects, but for this information to be informative, the recipient needs to have some knowledge of what the completion of this action implies regarding the sender's future intentions.

Nodes can also use *implicit communication* to send information. This approach assumes that other nodes can observe actions or changes to the state and infer what happened without needing to explicitly interact. This method dovetails with the decentralized coordination approach of Section 5.4.

Ontologies are a way to represent information (Section 8.1.2) and can be used jointly with the planner communication languages.

6.1.5 Temporal Planning and Resource Management

Protracted engagements require awareness and coordination over shared resources (including communications). Nodes are heterogeneous, resources are constrained (and possibly nonrenewable), and distributed tasks must be coordinated. Augmenting the planning with resource management ensures a unified approach to resource optimization and EW strategy selection. (Ignoring resources creates plans that may not be achievable, and/or need to be updated based on execution monitoring per Section 7.1.) The planner must make the best use of resources to handle current and anticipated threats according to the mission model, for the time scale over which the resources are needed. Temporal planning and resource planning are tightly coupled, because resource use is frequently temporal in nature. Temporal overlap between actions is limited if they need the same resource in conflicting ways.

> The critical feature of a cognitive radar is that various constraints have to be fulfilled resulting from the hardware, the platform or the environment. In reality, the constraining requirements are often contradicting. The art-of-radar-system engineering consists of handling these constraints by matching the different hardware and software components of the system to fulfill the requirements in a good compromise.
>
> —*R. Klemm et al. [64]*

Consider a heterogeneous team of nodes. Airborne assets must coexist and share tasks with ground-based EW systems. The appropriate assignment of tasks to assets depends on available resources (such as transmit power, frequency band, range, look angle, and time-to-target) and friendly considerations (such as fratricide minimization). For example, while ground-based systems may have longer time-on-station, they also have more limited range and greater vulnerability. The planner must ensure that these trade-offs are effectively managed when tasks are assigned.

Resource management ensures availability and appropriate usage of resources, both locally at a single node, and as needed by coordinating groups of nodes. The planner does this by (1) tracking current and expected resource usage, (2) determining what resource usage information should be shared with other nodes, and (3) sharing this information in an efficient and timely manner.

Resources have associated *capacity,* which can be a categorical value (e.g., available/unavailable) or a numerical value that tracks consumption. Consumable resources may be renewable or disposable and are either shareable by multiple tasks or exclusive for one task.

By incorporating resource consumption considerations into the planner, we ensure effective decisions that address threat conditions while respecting long-term distributed-resource constraints. Resource constraints have been successfully incorporated into decision-theoretic models for both single [65] and multiple [66] decision-makers.

Game-theoretic approaches are also relevant (Section 6.2) (e.g., for finding fair resource allocations). Various optimization and scheduling approaches can also accomplish the task; see surveys [7, 67–69], or Chapter 5. Multi-armed bandits (Section 7.3.4) are also useful.

Different approaches to resource modeling have different impacts on resource reasoning. Extending the definitions of Section 6.1.3, we augment the rewards *R* or state descriptions *s*. Table 6.3 outlines the options:

- Managing local resource consumption through costs. The simplest approach is to incorporate local resource consumption directly into the reward function, enabling simultaneous optimization of domain rewards and resource usage.
- Incorporating levels of local resources in state. A richer expression incorporates resource profiles into the state, allowing the planner to reason about the long-term cumulative resource effects of actions. Consumable resources, such as fuel or the number of available decoys, must be managed so that they are used when they are most effective for the mission. An augmented state $\overline{s_n}$ for node *n* is $\overline{s_n} = \langle s_n, q_1,\ldots,q_K\rangle$, where each q_k is the available remaining amount of resource *k,* and s_n is the original state. The transition function *P* operates on the augmented state. The resulting func tion $\overline{P_n}\left(\overline{s_n}'\middle|\overline{s_n},a_n\right)$ ensures that the only possible transitions are the ones consistent with resource consumption of the actions:

$$\overline{P_n}\left(\overline{s_n}'\middle|\overline{s_n},a_n\right)=\begin{cases}0 & \textit{if } \exists k \ \ \overline{s_n}.q_k - need(a_n,k) \neq \overline{s_n}'.q_k \\ P_n\left(\overline{s_n}'\middle|s_n,a_n\right) & \textit{otherwise}\end{cases}$$

- Incorporating levels of shared resources in state. Shared resources are not entirely under the node's control, thus requiring coordination. When

Table 6.3
Different Approaches To Resource Modeling

Network Scope	Temporal Scope	Approach
Local	Immediate	Consider resource utilization of actions: $R_n(s_n,a_n)=R_n(s_n,a_n)-\sum_{k=1}^{K}c_k \times needs(a_n,k)$
Local	Long-term	Capture local resources in state: $\overline{s_n}=\langle s_n,q_1,\ldots,q_r\rangle\overline{P_n}\left(\overline{s_n}'\middle|\overline{s_n},a_n\right)$
Global	Long-term	Capture shared resources in state: $\overline{P_n}\left(\overline{s_n}'\middle|\overline{s_n},\overrightarrow{a_n}\right)$

s_n = state of node *n*; $\overline{s_n}$ = augmented state of node *n*; a_n = action of node *n*; c_k = cost of resource *k*, needs(a_n,*k*) = action an requires resource *k*, *K* = number of resources nodes coordinate, they can exchange abstract information that includes probabilities using a shared resource. $\overline{P_n}$ thus accounts for another node taking an action that affects a shared resource. For example, consider two nodes that adjust their power level during transmission to have their combined SNR meet a desired level at the target. As the nodes (friendly or adversarial) move, the nodes need to update their power levels.

This unified approach to DM and resource management allows the planner to simultaneously consider resource-impact and attack effectiveness. The planner considers the impact of actions on immediate, near-term, and long-term mission goals.

6.1.6 Multiple Timescales

One of the system design questions to answer is how to handle DM at different timescales. Decisions may have different lead times; actions may have different durations; and metrics may have different latencies before they can be measured. Figure 6.6 illustrates the temporal groups. Priority tasks are often in conflict, both within platforms and across platforms, and architectural choices at lower levels constrain higher-level capabilities.

The most effective approach is to completely decouple the planners, for example, to have one planner that operates over units of days, a second that operates over minutes, and a third that operates over milliseconds. Each layer of temporal abstraction places constraints on the layer(s) below. This approach is simple to design, implement, and test, and it supports specialized reasoning at different scales.

Within a single planner, it may be beneficial to reason over abstractions, increasing the resolution in each iteration. This approach, while being harder to design than decoupled planners can lead to significantly better solutions because all dependencies are explicit.

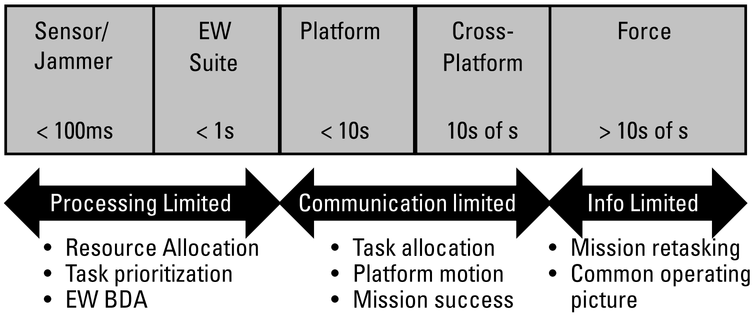


Figure 6.6 Different temporal requirements drive different architectural choices.

6.2 Game Theory

Game theory is a set of analytical tools used to model strategic, complex interactions among multiple entities. Game theory supports DM in uncertain environments when the uncertainty in the outcome of an action depends on how others react [70, 71].

Game theory characterizes both cooperation and competition. It is effective for cooperative teams, for resource allocation, and for adversarial settings. *Cooperative games* allow individuals to form agreements before choosing actions, while *noncooperative games* model selfish-but-rational individuals.

In the teamwork setting, game theory addresses self-organizing, decentralized networks where individuals have self-interest, but must cooperate to maximize performance [72–74]. Cooperative self-interest extends to distributed resource use. Game theory has been applied to the RF domain in power control, admission control, and network management [68, 73–76].

Game theory is an effective method to handle adversarial (competitive) situations. It is a popular approach in security settings, modeling and attacker and defender, such as cybersecurity [77–79], and EW for countering jammers [80–87]. Sundaram et al. [88] examined the case where defense EP actions have costs, while Blum [89] examined electronic countermeasures including bluffing to create uncertainty. The mixed-strategy game-theoretic decision can be viewed as rolling biased dice: Traditionally, the decision-maker chooses the single optimal action, while in an adversarial game, the decision-maker chooses an semirandom action based on a probability that is a function of the direct utility and the adversary's possible responses. The randomness makes it harder for the adversary to anticipate actions.

To incorporate game theory in an EW decision-maker, two significant challenges must be addressed: (1) confidence about adversary goals, and (2) computational cost.

In adversarial settings, the approach relies on an estimate of the adversary's utility. *Zero-sum games* assume that the loss and gain for each participant is exactly balanced by the losses or gains of other participants (i.e., that the total losses and gains for all players sum to zero). Zero-sum games can be solved with linear programming. EW usually assumes that the adversary wins when friendly forces lose, but this simplification may not be true. Moreover, games that are zero-sum in one dimension may not be zero-sum in another, increasing the complexity of the DM problem.

Algorithmic game theory studies the computational aspects of game theory, focusing on finding solutions with reasonable limits on computational complexity. Generally, game-theoretic solutions are computationally difficult, although approximate or problem-specific solutions can be constructed.

6.3 Human-Machine Interface

A successful design of the HMI needs to be intuitive, flexible, extensible, and highly responsive. The main objectives of the HMI system should be to (1) improve operational performance, (2) increase effectiveness, and (3) alleviate tasking overload on pilots/drivers, EWOs, and mission planners. The system should provide a set of wide-ranging and intuitive displays and controls for the operator's timely interaction with EW system functions and data. A well-considered HMI design is critical for maintaining situational awareness.

Customarily, HMI transforms human-operator actions to command actions for the machine and then transforms the machine's sensory data back to the human operator as illustrated in Figure 6.7 [90]. In traditional HMIs, these transformations are static and do not change based on the state of the human operator, the machine, or the environment.

Human-machine teaming (HMT) addresses this gap: Transformation processes are no longer static and may employ intelligent agents and ML. HMT is a relationship between the human operator and machine that goes beyond a human operating or supervising machine(s) [90, 91]. This evolving discipline aims to develop operative teams where human operator and automation are complementary team members [92, 93]. The HMT framework should be based around the following four themes to design an ethical and trustworthy AI system [94]:

1. Accountability to humans;
2. Cognizance of speculative risks and benefits;
3. Respectfulness and security;
4. Honesty and usability.

In addition, the interrelationship between humans and machines will have the following five dominant aspects or technological challenges to contend with [95]:

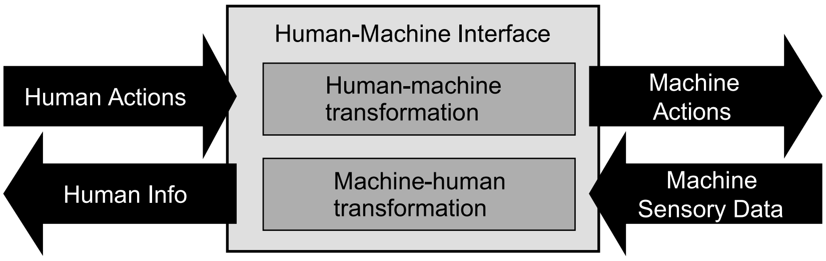


Figure 6.7 An HMI translates between human and machine (adapted from [90]).

- Human state sensing and assessment: Performance, behavior, and physiological factors;
- Human-machine interaction: Communication and information-sharing between human and machine;
- Task and cognitive modeling: Establishing workload and DM balance by task and function allocation;
- Human and ML: Adaptive learning and extended mutual training between human and machine;
- Data fusion and understanding: Integration of human and machine data to generate a shared world model.

An EBM system plans how many platforms to deploy, which resources each gets, and where they will go. To achieve all of these objectives, the EBM system can benefit from multiple HMT-enabled HMIs and may have to establish interaction points with every notional block of the EBM system depicted in Figure 1.4. It may also be beneficial to have an overarching, global HMI providing the end-to-end capability for the entire EBM system. Trust must be established at every HMT-based HMI. Figure 6.8 depicts the potential HMI interaction points for the EBM components of the cognitive EW system.

These interaction points vary among the cognitive EW system participants (i.e., designers, commanders, mission planners, EWOs, and the cognitive system). Below we highlight some of the concepts/interaction points that each stakeholder type may be concerned with.

System designers have the greatest impact on how the system works, and are often forgotten as key members of the human-machine team. Leveraging human expertise improves the performance of the AI components, thus supporting

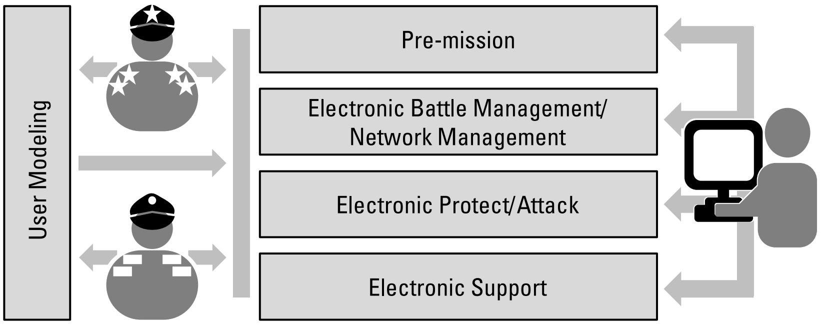


Figure 6.8 Premission planners, EWOs, and system designers interact with the system at all points; the system must accept their commands and inputs, track their tasks, learn their preferences, and explain its own decisions.

EWOs and the EW mission. Planning systems have leveraged human expertise since their initial inception, for example, showing the power of domain-guidance in HTN planners (Section 6.1.2). Haigh et al. [96] discusses hybrid ML models in RF as one example of a neurosymbolic AI approach that seeks to blend numerical approaches (e.g., neural nets) with symbolic approaches (i.e., concepts that a human can understand) [97, 98]. Designers can provide:

- Assumptions about types of missions, users, and platforms.
- Concepts to represent [e.g., via ontologies (see Section 6.1.4)].
- Choice of system observables, controllables, and metrics;
- Abstract feature construction for ES (e.g., cross-layer interactions or effect of topography on signal propagation). For example, Haigh et al. [96] use distance-to-neighbors to bootstrap a rough estimate of MANET throughput.
- Statespace reduction and guidance for optimization (e.g., relevant or critical parameters, current operating space, and constraints).
- Type of learned models (e.g., decision trees versus DeepNets as a function of how explainable the results need to be [99]).
- Within learned models to match data types, inference requirements, speed, or accuracy (e.g., changing the form of the model).
- Search heuristics to reduce search effort. Experts and offline simulations can create guidelines and rules, expressed in a language appropriate to the decision-maker. For example, Mitola [100] suggested radio knowledge representation language (RKRL), and a policy description language manages spectrum sharing in radio networks [101].

Commanders control and guide all participants in the mission. The system must capture their objectives and vision. *Commander's intent* (CI), a critical concept for the entire EW mission team, describes concepts such as acceptable risk, possible methods, and success conditions. Human participants and machine-generated decision support need to have the ability to communicate and interpret a shared CI [102, 103]. For example, Gustavsson et al. [102] propose a model that relates CI to effects, and supports both traditional military planning and effects-based operations.

Utility captures objectives. Roth et al. [104] break down high-level utilities to the lower-level specifications that the system needs to run. Haigh et al. [105] describe policy-based methods for a human expert and an approach to turning them into an implementable, operationally meaningful objective function. A satisfactory objective function must be (1) operationally meaningful, (2) sufficiently flexible, (3) easy to modify, and (4) implementable and optimizable in practice.

Mission planners and *EWOs* interact with the EW system both in preplanning and in real time. They must be able to guide and control the automation, and understand all feedback. EW resources for mission planners include databases for intelligence and operational data, (automated) planners, spatial and propagation modeling tools, and reach-back support if required information is not available immediately [1] (Figure 6.9).

Mixed-initiative planning is an approach to developing plans interactively; for example the automated planner presents trade-offs, and the human planner selects from amongst the choices [4, 22, 106, 107]. *Argumentation* is a natural way to present conflicting concepts to humans, showing the reasons why an observation might be true or a decision might be good, and the reasons against (including deception) [56, 108, 109]. *Playbook interfaces* allow a user to call a "play" and then tailor it to the situation [110, 111]. Influenced by sports playbooks, the playbook assumes each player knows the broad outline of a play, and each play has been well-practiced by the team. The shared-task model provides a means of human-automation communication about plans, goals, methods, and resource usage. Raytheon's EW Planning and Management Tool (EWPMT) uses a playbook interface (Example 6.4). Playbook interfaces have an additional benefit: They map naturally to HTN planning (Section 6.1.2). For example, Goldman et al. [21] propose a mixed-initiative planning system with a playbook GUI to generate, check, and modify plans for teams of heterogeneous UAVs. Finally, the concept of *adjustable autonomy* allows a human user to grant the platform different levels of autonomy [115].

In addition to these planning and control approaches, humans can support ES, for example by labeling unknown signals, or providing supporting Human-INT evidence such environmental conditions (e.g., topography, weather, or threats).

Another underrecognized concept in a HMT is that of the machine understanding the human users. *Preference learning* captures a user's preferences in cases where they are hard to state explicitly [116]. Preference learning approaches can learn *utility functions* (learning a function that captures why *a* is better than *b*), or learn *preference relations* (the partially ordered graph between options). Preference learning can take explicit or implicit feedback [117]. Bantouna et al. [118] use Bayesian statistics to learn user preferences for QoS. *Intent Recognition* (Section 4.6) attempts to understand what goals the user is trying to achieve; this information can be used to anticipate user needs and customize the display.

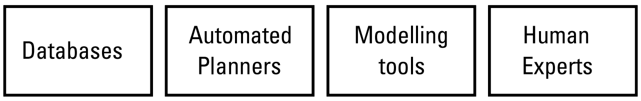


Figure 6.9 A variety of decision aids can help human EW planners.

Example 6.4 Raytheon's EWPMT uses plays to interact with the EW personnel.

Raytheon's EWPMT software assists the maneuver commander's ability to plan, coordinate, and synchronize EW, spectrum management, and Cyber operations [112–114]. EWPMT has been a U.S. Army program of record since 2014.

EWPMT provides EW mission planning, EW targeting, and simulation capabilities to support COA development. It develops the electromagnetic order of battle, deconflicts EW and communications assets, and displays the electromagnetic operational environment [113]. EWPMT takes in data from sensors in real time and provides automation and analytics. It allows users to easily identify a new threat, geolocate that threat, and share that information [112].

Automated "plays" help relieve the cognitive burden on EW in the field, enabling automated actions when certain conditions are met, such as turning off sensors on a specific frequency if they're detected by the sensing system [112].

A cognitive EW system designer/developer may wish to implement an *agile* software development approach whose tenets are also applicable to hardware. In the agile approach, one creates *user stories* to capture each user's requirements and desired functionality at the highest level. The story's purpose is to communicate how a feature will provide value to various stakeholders [119]. Within the context of HMI, these user stories should encapsulate all the necessary interfaces/interaction points for each end user. A user story is an informal, general description of a feature written from the perspective of the end user, and can be as simple as [120]:

As a *type of user,* I want *goal* so that *reason.*

User stories are not system requirements, but a way to ensure that these various requirements are "not lost in translation" down the line.

> A key component of agile software development is putting people first, and a user story puts end users at the center of the conversation. These stories use non-technical language to provide context for the development team and their efforts. After reading a user story, the team knows why they are building, what they're building, and what value it creates. [119]

At the highest level, the cognitive EW system's users are the stakeholders: designers, commanders, mission planners, EWOs, and machines or intelligent agents. The agile approach supports good human factors design, wherein stakeholders are regularly consulted from the beginning. MITRE's HMT engineering guide suggests approaches for this process [91].

6.4 Conclusion

Automated EBM planning systems must support human planners at all stages of EW planning—long-range planning, midterm planning, and real-time mission management. Multiple tightly coupled decision-makers can handle separate concerns.

HTN planners work for complex domains, and naturally support the transparency, explainability, and control required for HMT. In-mission replanning (Section 7.2) ensures that the plan remains feasible despite surprises and changes in mission goals.

Planners must address both action and information uncertainty, manage temporal and resource constraints, and choose what and when to communicate with team members. Domain-specific heuristics and metalearning (Section 5.1.3) help improve the search process. Game-theoretic approaches assist with cooperative team management and competitive adversaries.

References

[1] Chairman of the Joint Chiefs of Staff, *Joint publication 3-13.1: Electronic warfare,* 2012. Online: https://fas.org/irp/doddir/dod/jp3-13-1.pdf.

[2] US Air Force, *Air Force Operational Doctrine: Electronic Warfare, Document 2-5.1*, 2002. Online: https://tinyurl.com/af-ew-2002-pdf.

[3] Haigh, K. Z., and M. Veloso, "Planning, Execution and Learning in a Robotic Agent," in *AIPS,* Summary of Haigh's Ph.D. thesis, 1998.

[4] Siska, C., B. Press, and P. Lipinski, "Tactical Expert Mission Planner (TEMPLAR)," TRW Defense Systems, Tech. Rep. RADC-TR-89-328.

[5] Miller, G., "Planning and Scheduling for the Hubble Space Telescope," in *Robotic Telescopes,* Vol. 79, 1995.

[6] Adler, D., D. Taylor, and A. Patterson, "Twelve Years of Planning and Scheduling the Hubble Space Telescope: Process Improvements and the Related Observing Efficiency Gains," in *Observatory Operations to Optimize Scientific Return III,* International Society for Optics and Photonics, Vol. 4844, SPIE, 2002.

[7] Benaskeur, A., E. Bossé, and D. Blodgett, "Combat Resource Allocation Planning in Naval Engagements," Defence R&D Canada, Tech. Rep. DRDC Valcartier TR 2005–486, 2007.

[8] Russell, S., and P. Norvig, *Artificial Intelligence: A Modern Approach,* Pearson Education, 2015.

[9] Helmert, M., *Understanding Planning Tasks: Domain Complexity and Heuristic Decomposition,* Springer, 2008. (Revised version of Ph.D. thesis.)

[10] Wilkins, D., *Practical Planning,* Morgan Kaufmann, 1998.

[11] *International Conference on Automated Planning and Scheduling Competitions.* Accessed 2020-09-12. Online: https://www.icapsconference.org/competitions/.

[12] Yuen, J., "Automated Cyber Red Teaming," Defence Science and Technology Organisation, Australia, Tech. Rep. DSTO-TN-1420, 2015.

[13] Haslum, P., et al., *An Introduction to the Planning Domain Definition Language,* Morgan & Claypool, 2019.

[14] Korf, R., "Heuristic Evaluation Functions in Artificial Intelligence Search Algorithms," *Minds and Machines,* Vol. 5, 1995.

[15] Liang, Y.-q., X.-s. Zhu, and M. Xie, "Improved Chromosome Encoding for Genetic-Algorithm-Based Assembly Sequence Planning," in *International Conf. on Intelligent Computing and Integrated Systems,* 2010.

[16] Brie, A., and P. Morignot, "Genetic Planning Using Variable Length Chromosomes," In *ICAPS,* 2005.

[17] Goodwin, M., O. Granmo, and J. Radianti, "Escape Planning in Realistic Fire Scenarios with Ant Colony Optimisation," *Applied Intelligence,* Vol. 42, 2015.

[18] Erol, K., J. Hendler, and D. Nau, "HTN Planning: Complexity and Expressivity," in *AAAI,* 1994.

[19] Erol, K., J. Hendler, and D. Nau, "Complexity Results for HTN Planning," Institute for Systems Research, Tech. Rep., 2003.

[20] Georgievski, I., and M. Aiello, "HTN Planning: Overview, Comparison, and Beyond," *Artificial Intelligence,* Vol. 222, 2015.

[21] Goldman, R. P., et al., "MACBeth: A Multiagent Constraint-Based Planner," in *Digital Avionics Systems Conference,* 2000.

[22] Mitchell, S., "A Hybrid Architecture for Real-Time Mixed-Initiative Planning and Control," in *AAAI,* 1997.

[23] Qi, C., and D. Wang, "Dynamic Aircraft Carrier Flight Deck Task Planning Based on HTN," *IFAC-PapersOnLine,* Vol. 49, No. 12, 2016, Manufacturing Modelling, Management and Control.

[24] Kott, A., et al., *Toward Practical Knowledge-Based Tools for Battle Planning and Scheduling,* AAAI, 2002.

[25] Ground, L., A. Kott, and R. Budd, "Coalition-Based Planning of Military Operations: Adversarial Reasoning Algorithms in an Integrated Decision Aid," *CoRR,* 2016.

[26] Kott, A., et al., *Decision Aids for Adversarial Planning in Military Operations: Algorithms, Tools, and Turing-Test-Like Experimental Validation,* 2016. Online: https://arxiv.org/abs/1601.06108.

[27] Alford, R., U. Kuter, and D. Nau, "Translating HTNs to PDDL: A Small Amount Of Domain Knowledge Can Go a Long Way," in *IJCAI,* 2009.

[28] Kuter, U., and D. Nau, "Using Domain-Configurable Search Control for Probabilistic Planning," *AAAI,* 2005.

[29] Victoria, J., "Automated Hierarchical, Forward-Chaining Temporal Planner for Planetary Robots Exploring Unknown Environments," Ph.D. dissertation, Technical University of Darmstadt, Germany, 2016.

[30] Andersen, M., T. Bolander, and M. Jensen, "Conditional Epistemic Planning," in *Logics in Artificial Intelligence,* 2012.

[31] To, S., T. Son, and E. Pontelli, "Contingent Planning as AND/OR Forward Search with Disjunctive Representation," in *ICAPS,* 2011.

[32] Blum, A., and J. Langford, "Probabilistic Planning in the Graph Plan Framework," in *European Conference on Planning,* 1999.

[33] Majercik, S., and M. Littman, "Contingent Planning Under Uncertainty via Stochastic Satisfiability," in *AAAI,* 1999.

[34] Ferraris, P., and E. Giunchiglia, "Planning as Satisfiability in Nondeterministic Domains," in *AAAI,* 2000.

[35] Joshi, S., and R. Khardon, "Probabilistic Relational Planning with First Order Decision Diagrams," *Artificial Intelligence Research,* Vol. 41, 2011.

[36] Mausam, W., and A. Kolobov, *Planning with Markov Decision Processes: An AI Perspective,* Morgan & Claypool, 2012.

[37] Sutton, R., and A. Barto, *Reinforcement Learning: An Introduction,* Bradford, 2018.

[38] Littman, M., T. L. Dean, and L. P. Kaelbling, "On the Complexity of Solving Markov Decision Problems," in *Uncertainty in AI,* 1995.

[39] Gupta, A., and S. Kalyanakrishnan, "Improved Strong Worst-Case Upper Bounds for MDP Planning," in *IJCAI,* 2017.

[40] Jinnai, Y., et al., "Finding Options That Minimize Planning Time," in *ICML,* 2019.

[41] McMahan, B., and G. Gordon, "Fast Exact Planning in Markov Decision Processes," in *ICAPS,* 2005.

[42] Taleghan, M., et al., "PAC Optimal MDP Planning with Application to Invasive Species Management," Vol. 16, 2015.

[43] Busoniu, L., et al., "Optimistic Planning for Sparsely Stochastic Systems," in *Adaptive Dynamic Programming,* 2011.

[44] van Seijen, H., and R. Sutton, "Efficient Planning in MDPs by Small Backups," in *ICML,* 2013.

[45] Zhang, J., M. Dridi, and A. Moudni, "A Stochastic Shortest-Path MDP Model with Dead Ends for Operating Rooms Planning," in *International Conference on Automation and Computing,* 2017.

[46] Yoon, S., A. Fern, and R. Givan, "FF-Replan: A Baseline for Probabilistic Planning," in *ICAPS,* 2007.

[47] Teichteil-Koenigsbuch, F., G. Infantes, and U. Kuter, "RFF: A Robust, FF-Based MDP Planning Algorithm for Generating Policies with Low Probability of Failure," in *ICAPS,* 2008.

[48] Oliehoek, F., and C. Amato, *A Concise Introduction to Decentralized POMDPs,* Springer, 2016.

[49] Bernstein, D., et al., "The Complexity of Decentralized Control of Markov Decision Processes," *Mathematics of Operations Research,* Vol. 27, No. 4, 2002.

[50] Varakantham, P., et al., "Exploiting Coordination Locales in Distributed POMDPs via Social Model Shaping," in *ICAPS,* 2009.

[51] Nair, R., et al., "Networked Distributed POMDPs: A Synthesis of Distributed Constraint Optimization and POMDPs," in *AAAI,* 2005.

[52] Denoeux, T., "40 Years of Dempster-Shafer Theory," *Approximate Reasoning,* 2016.

[53] Liu, L., and R. Yager, "Classic Works of the Dempster-Shafer Theory of Belief Functions: An Introduction," *Studies in Fuzziness and Soft Computing,* Vol. 219, 2008.

[54] Nickravesh, M, "Evolution of Fuzzy Logic: From Intelligent Systems and Computation to Human Mind," in *Forging New Frontiers: Fuzzy Pioneers I. Studies in Fuzziness and Soft Computing,* Vol. 217, Springer, 2007.

[55] Mogdil, S., and H. Prakken, "The ASPIC* Framework for Structured Argumentation: A Tutorial," *Argument and Computation,* Vol. 5, No. 1, 2014.

[56] Collins, A., D. Magazzeni, and S. Parsons, "Towards an Argumentation-Based Approach to Explainable Planning," in *Explainable Planning,* 2019.

[57] Mostafa, H., and V. Lesser, "Offline Planning For Communication by Exploiting Structured Interactions in Decentralized MDPs," in *International Conference on Web Intelligence and Intelligent Agent Technology,* 2009.

[58] Mogali, J., S. Smith, and Z. Rubinstein, "Distributed Decoupling of Multiagent Simple Temporal Problems," in *IJCAI,* 2016.

[59] Melo, F., and M. Veloso, "Decentralized MDPs with Sparse Interactions," *Artificial Intelligence,* Vol. 175, 2011.

[60] Yordanova, V., "Intelligent Adaptive Underwater Sensor Networks," Ph.D. dissertation, University College London, London, UK, 2018.

[61] Ren, P., et al., *A Survey of Deep Active Learning,* 2020. Online: https://arxiv.org/abs/2009.00236.

[62] Becker, R., V. Lesser, and S. Zilberstein, "Decentralized Markov Decision Processes with Event-Driven Interactions," in *AAMAS,* 2004.

[63] Xuan, P., and V. Lesser, "Incorporating Uncertainty in Agent Commitments," in *Agent Theories, Architectures, and Languages,* 1999.

[64] Klemm, R., et al. (Eds.), *Novel Radar Techniques and Applications,* Scitech Publishing, 2017.

[65] Altman, E., *Constrained Markov Decision Processes,* Chapman Hall, 1999.

[66] Dolgov, D., and E. Durfee, "Stationary Deterministic Policies for Constrained MDPs with Multiple Rewards, Costs, and Discount Factors," in *IJCAI,* 2005.

[67] Muhammad, N., et al., "Resource Allocation Techniques in Cooperative Cognitive Radio Networks," *Communications Surveys & Tutorials,* Vol. 16, No. 2, 2013.

[68] Ramzan, M., et al., "Multiobjective Optimization for Spectrum Sharing in Cognitive Radio Networks: A Review," *Pervasive and Mobile Computing,* Vol. 41, 2017.

[69] Jorswieck, E., and M. Butt, "Resource Allocation for Shared Spectrum Networks,"in *Spectrum Sharing: The Next Frontier in Wireless Networks,* Wiley, 2020.

[70] Shoham, Y., and K. Leyton-Brown, *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations,* Cambridge University Press, 2009.

[71] Nisan, N., et al. (Eds.), *Algorithmic Game Theory,* New York: Cambridge University Press, 2007.

[72] Saad, W., et al., "Coalitional Game Theory for Communication Networks: A Tutorial," *IEEE Signal Processing Magazine*, 2009.

[73] MacKenzie, A., and L. DaSilva, *Game Theory for Wireless Engineers,* Morgan & Claypool, 2006.

[74] Han, Z., et al., *Game Theory in Wireless and Communication Networks: Theory, Models, and Applications,* Cambridge University Press, 2012, The course http://www2.egr.uh.edu/~zhan2/game_theory_course/ contains slides based on the book.

[75] Zayen, B., A. Hayar, and G. Noubir, "Game Theory-Based Resource Management Strategy for Cognitive Radio Networks," *Multimedia Tools and Applications,* Vol. 70, No. 3, 2013.

[76] Mkiramweni, M., et al., "Game-Theoretic Approaches for Wireless Communications with Unmanned Aerial Vehicles," *IEEE Wireless Communications,* 2018.

[77] Snyder, M., R. Sundaram, and M. Thakur, "A Game-Theoretic Framework for Bandwidth Attacks and Statistical Defenses," in *Local Computer Networks,* 2007.

[78] Schramm, H., et al., "A Game Theoretic Model of Strategic Conflict in Cyberspace," *Military Operations Research,* Vol. 19, No. 1, 2014.

[79] Kasmarik, K., et al., "A Survey of Game Theoretic Approaches to Modelling Decision-Making in Information Warfare Scenarios," *Future Internet,* Vol. 8, 2016.

[80] Liu, X., et al., "SPREAD: Foiling Smart Jammers Using Multilayer Agility," in *INFOCOM,* 2007.

[81] Yang, D., et al., "Coping with a Smart Jammer in Wireless Networks: A Stackelberg Game Approach," *IEEE Transactions on Wireless Communications,* Vol. 12, No. 8, 2013.

[82] Firouzbakht, K., G. Noubir, and M. Salehi, "Multicarrier Jamming Mitigation: A Proactive Game Theoretic Approach," in *Proactive and Dynamic Network Defense,* 2019.

[83] Wang, H., et al., "Radar Waveform Strategy Based on Game Theory," *Radio Engineering,* Vol. 28, No. 4, 2019.

[84] Li, K., B. Jiu, and H. Liu, "Game Theoretic Strategies Design for Monostatic Radar And Jammer Based on Mutual Information," *IEEE Access,* Vol. 7, 2019.

[85] Zhang, X., et al., "Game Theory Design for Deceptive Jamming Suppression in Polarization MIMO Radar," *IEEE Access,* Vol. 7, 2019.

[86] Wonderley, D., T. Selee, and V. Chakravarthy, "Game Theoretic Decision Support Framework for Electronic Warfare Applications," in *Radar Conference,* IEEE, 2016.

[87] Mneimneh, S., et al., "A Game-Theoretic and Stochastic Survivability Mechanism Against Induced Attacks in Cognitive Radio Networks," *Pervasive and Mobile Computing,* Vol. 40, 2017.

[88] Sundaram, R., et al., "Countering Smart Jammers: Nash Equilibria for Asymmetric Agility And Switching Costs," in *Classified US Military Communications,* 2013.

[89] Blum, D., "Game-Theoretic Analysis of Electronic Warfare Tactics with Applications to the World War II Era," M.S. thesis, 2001.

[90] Thanoon, M., "A System Development For Enhancement of Human-Machine Teaming," Ph.D. dissertation, Tennessee State University, 2019.

[91] McDermott, P., et al., "Human-Machine Teaming Systems Engineering Guide," MITRE, Tech. Rep. AD1108020, 2018.

[92] Urlings, P., and L. Jain, "Teaming Human and Machine," in *Advances In Intelligent Systems For Defence,* World Scientific, 2002.

[93] Taylor, R., "Towards Intelligent Adaptive Human Autonomy Teaming," Defence Science and Technology Laboratory, UK, Tech. Rep. STOMP-HFM-300, 2018.

[94] Smith, C., *Ethical Artificial Intelligence (AI),* 2020. doi: 10.1184/ R1/c.4953576.v1.

[95] Overholt, J., and K. Kearns, *Air Force Research Laboratory Autonomy Science & Technology Strategy,* 2014. Online: https://tinyurl.com/afrl-autonomy.

[96] Haigh, K. Z., S. Varadarajan, and C. Y. Tang, "Automatic Learning-Based MANET Cross-Layer Parameter Configuration," in *Workshop on Wireless Ad hoc and Sensor Networks,* IEEE, 2006.

[97] Hilario, M., "An Overview of Strategies for Neurosymbolic Integration," in *Connectionist-Symbolic Integration,* Lawrence Erlbaum, 1997.

[98] D'Avila Garcez, A., L. Lamb, and D. Gabbay, *Neural-Symbolic Cognitive Reasoning,* Springer, 2009.

[99] Confalonieri, R., et al., "A Historical Perspective of Explainable Artificial Intelligence," *WIREs Data Mining and Knowledge Discovery,* Vol. 11, No. 1, 2021.

[100] Mitola III, J., "Cognitive Radio: An Integrated Agent Architecture for Software Defined Radio," Ph.D. dissertation, Royal Institute of Technology (KTH), Kista, Sweden, 2000.

[101] Berlemann, L., S. Mangold, and B. H. Walke, "Policy-Based Reasoning for Spectrum Sharing In Radio Networks," in *Symposium on New Frontiers in Dynamic Spectrum Access Networks,* 2005.

[102] Gustavsson, P., et al., "Machine Interpretable Representation of Commander's Intent," in *International Command and Control Research and Technology Symposium: C2 for Complex Endeavors,* 2008.

[103] Gustavsson, P., et al., "Formalizing Operations Intent and Effects for Network-Centric Applications," in *International Conference on System Sciences,* 2009.

[104] Roth, E., et al., "Designing Collaborative Planning Systems: Putting Joint Cognitive Systems Principles to Practice," in *Cognitive Systems Engineering: A Future for a Changing World,* Ashgate Publishing, 2017, Ch. 14.

[105] Haigh, K. Z., O. Olofinboba, and C. Y. Tang, "Designing an Implementable User-Oriented Objective Function for MANETs," in *International Conference On Networking, Sensing and Control,* IEEE, 2007.

[106] Heilemann, F., and A. Schulte, "Interaction Concept for Mixed-Initiative Mission Planning on Multiple Delegation Levels in multi-UCAV Fighter Missions," in *International Conference on Intelligent Human Systems Integration,* Springer, 2019.

[107] Irandoust, H., et al., "A Mixed-Initiative Advisory System for Threat Evaluation," in *International Command and Control Research and Technology Symposium,* 2010.

[108] Parsons, S., et al., "Argument Schemes For Reasoning About Trust," *Argument & Computation,* Vol. 5, No. 2-3, 2014.

[109] Sarkadi, S., et al., "Modelling Deception Using Theory of Mind in Multiagent Systems," *AI Communications,* Vol. 32, No. 4, 2019.

[110] Miller, C., and R. Goldman, "'Tasking Interfaces; Associates That Know Who's the Boss," in *Human/Electronic Crew Conference,* 1997.

[111] Miller, C., et al., "The Playbook," in *Human Factors and Ergonomics Society,* 2005.

[112] Pomerleau, M., "Here's What the Army is Saying about Its New Electronic Warfare Solution," *C4ISRNet,* 2018.

[113] US Army, *Electronic Warfare Planning and Management Tool (EWPMT),* Accessed 2020-09-19. Online: https://tinyurl.com/ewpmt-army.

[114] Raytheon, *Electronic Warfare Planning Management Tool (EWPMT),* Accessed 2020-09-19. Online: https://tinyurl.com/ewpmt-rtn.

[115] Mostafa, S., M. Ahmad, and A. Mustapha, "Adjustable Autonomy: A Systematic Literature Review," *Artificial Intelligence Review,* No. 51, 2019.

[116] Fürnkranz, J., and E. Hüllermeier, *Preference Learning,* Springer Verlag, 2011.

[117] Kostkova, P., G. Jawaheer, and P. Weller, "Modeling User Preferences In Recommender Systems," *ACM Transactions on Interactive Intelligent Systems,* Vol. 4, 2014.

[118] Bantouna, A., et al., "An Overview of Learning Mechanisms for Cognitive Systems," *Wireless Communications and Networking,* No. 22, 2012.

[119] Rehkopf, M., *User Stories with Examples and Template.* Accessed 2020- 9-28. Online: https://www.atlassian.com/agile/project-management/user-stories.

[120] Mountain Goat Software, *User stories.* Accessed 2020-09-28. Online: https://www.mountaingoatsoftware.com/agile/user-stories.

7

Real-Time In-Mission Planning and Learning

SA and DM are key components to all EW systems. Chapters 1–6 presented approaches for these activities without putting them in context of *in-mission* performance.

Traditional approaches to EW replanning assumed lengthy timescales between creating EP/EA plans and updating ES models. In a fully integrated modern EW system the system can, and *must,* respond to unexpected events that happen during the mission and learn from its experiences.

In a natural evolution of control theory, execution monitoring tracks the outcomes of actions, giving a planner the ability to adapt to changes, and a learner the ability to incorporate experience.

7.1 Execution Monitoring

An EW system must monitor execution at all levels: ES, EP/EA, and EBM. Monitoring must occur on each platform, and across the team of multiple nodes. Execution monitoring requires ES capabilities and becomes an introspective component of ES.

ES describes a situation/environment. Execution monitoring compares the observed environment to the expected environment.

Monitoring actions illustrated in Figure 7.1 include the following:

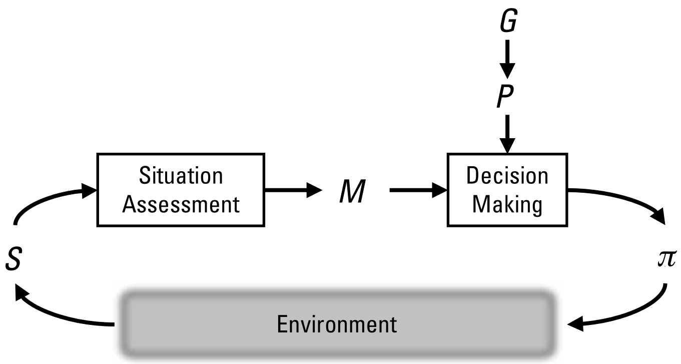


Figure 7.1 In uncertain or dynamic domains, execution monitoring must analyze sensors *S*, models *M*, actions π, plans *P*, and goals *G*.

- *Sensor monitoring:* Is the raw data correct? Are the sensors operating correctly? Is the inferred current situation rare or anomalous? Can the sensor information be trusted? Is there an adversarial attack on the sensors?
- *Model monitoring:* Are the ES models correct? Is the SA correct? Does the model cover the current situation; is the situation within the domain of expertise? Is there an adversarial attack on the situational inferences? In a system using learned models, does the input data follow the distribution over which the model was trained (its *domain of expertise* or *operational space*)? When the system also learns in-mission, is the concept drift acceptable?
- *Action monitoring:* Are the assumptions or preconditions still true for an action? After acting, did one of the expected outcomes happen? How well did the action perform? Were there minor, but possibly critical, variations on the expected outcome(s)?
- *Plan monitoring:* Is the remainder of the plan still feasible? Can the system still accomplish its goals? Do any tasks need to be reallocated? Will the costs or time to accomplish the mission change?
- *Goal monitoring:* Before acting, do the long-term goals still make sense? Has the mission changed in a meaningful way? Are there new opportunities to accomplish additional objectives?

A failure of one of these questions might trigger *replanning* or trigger *learning*. Replanning involves updating the plan to handle the new situation; learning updates and corrects the underlying models.

ES capabilities evaluate expected conditions against the current conditions. When the feedback matches that of the learned models, the feedback enables in-mission learning updates. For example, execution monitoring can compute the model's prediction error to trigger retraining, as in Example 7.1.

By tying actions to observations, execution monitoring can compute the causal relationships between events. This analysis allows the system to determine whether events are an inherent function of the observed environment, or whether actions are true precursors. The system can then use precursor events as "red flags" of imminent danger. Section 4.5 describes how to detect these patterns.

7.1.1 Electronic Warfare Battle Damage Assessment

In EP missions, the performance metric can often be directly *measured* (e.g., BER or throughput). For an EA system, ES must instead *infer* the effectiveness of the attack. Examples of inferred values include jamming effectiveness [1], radar mode, Pfa/Pd, or POI. Using ground truth collected during a test phase, an ML model can be trained to infer these values and then used at runtime.

For example, an EW BDA system can estimate whether a threat's mode or other behavior changed. Different radar modes vary different parameters to accomplish different tasks [2–5], including, for example, pulse width to improve search capability or PRI to change detection range [3]. Changes in these characteristics may indicate EA effectiveness [5–10].

In addition to causal event analysis, historical patterns add to the evidence. For example, when a radar parameter changes several times in a row, the system can increase its confidence in the effectiveness of the EA technique. A hand-engineered rules-based BDA system might set thresholds for alerts, or construct rules. Each rule might contribute a score; scores can be combined arithmetically or used as votes for the overall effectiveness of the action. As emitters become more complex, hand-generated rules quickly become brittle.

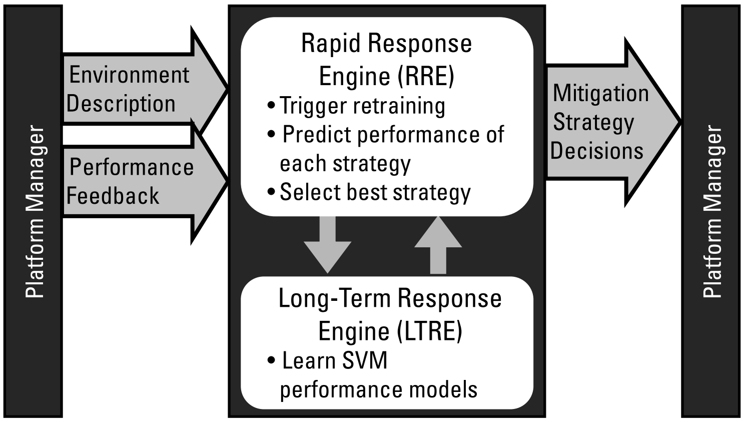
To reduce the brittleness of hand-generated rules, incorporate these features into an ML model. Any ML model, including rule learning, may be appropriate as a BDA scoring engine. Standard classification methods can combine features to estimate BDA effectiveness.

Figure 7.2 illustrates a possible ML architecture. Each feature (e.g., mode or behavior) could be derived by a hand-generated algorithm, or inferred by ML (traditional, DeepNet, or hybrid). Multiple classifiers use these features to compute their local threat-mode estimates, and the ensemble method (Section 3.2) chooses the best threat mode. Finally, change detection compares the current mode to the previous mode(s). Algorithm 7.1 shows a corresponding code listing for an ensemble of different classifiers chosen from scikit-learn: a decision tree, an extra tree, a Gaussian Naïve Bayes, and two *k*NN models using different values of *k*. Figure 7.3 shows the results. The tenfold cross-validation in `evalModel()` yields 10 scores per model, presented in a box-and-whisker plot.

Example 7.1 The BBN SO performs real-time in-mission learning and optimization for communications EP.

The BBN SO learns how to optimize network performance in the presence of previously unknown interference and jamming conditions [35, 51, 52]. The SO automatically recognizes conditions that affect communications quality and learns to select a configuration that improves performance, even in highly dynamic missions. The SO is the first known communications EP system to use ML *in-mission*, at mission-relevant timescales.

The SO comprises a rapid response-engine (RRE) that makes strategy decisions, and a long-term response engine (LTRE) that learns the models of how strategies perform.


Platform Manager
Environment Description
Performance Feedback
Rapid Response Engine (RRE)
• Trigger retraining
• Predict performance of each strategy
• Select best strategy
Long-Term Response Engine (LTRE)
• Learn SVM performance models
Mitigation Strategy Decisions
Platform Manager

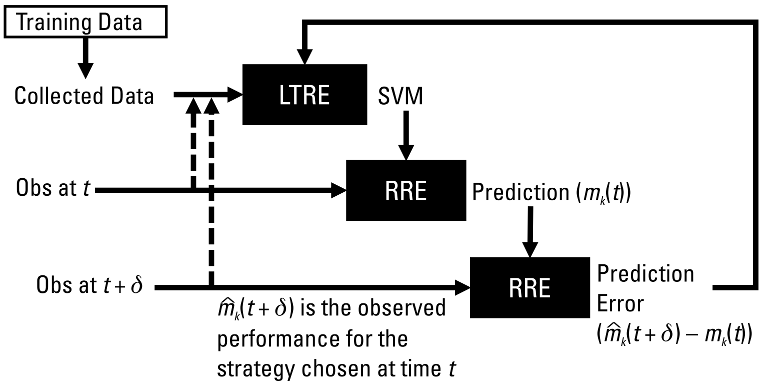
Given a set of training data (possibly empty), the LTRE constructs a model of the performance surface for each metric: $m_k = f_k(o,s)$. The SO uses SVMs (Section 3.1.1) as a regression tool to estimate performance (Section 4.2). The SO uses SVMs because they can learn from small numbers of training examples, and because they can be implemented on limited hardware. Because memory is limited on this device, the LTRE manages data diversity (Section 8.3.2) and forgets old instances (Section 8.3.4).

The RRE then uses these SVM models f_k to make rapid real-time strategy decisions during the mission using Algorithm 5.1.

The SO effectively learns how to mitigate communications interference, in real-time, on limited hardware, from small numbers of examples, *even when given no a priori training data.*

The SO has been tested against many different interference conditions, from distributed interference to complex jammers, in simulation, in lab emulation, and in the field. Results are shown on a PPC440 with a hard real-time operating system [35].

At each time step, the platform provides the observables and the performance feedback for the metrics m. If the estimated performance of $m_k(t)$ differs sufficiently from the observed performance $\hat{m}_k(t+\delta)$,* then the RRE triggers a retraining event, and the LTRE rebuilds the SVM for m_k using the new training data. The SO therefore uses RL to perform in-mission learning at mission-relevant timescales.


Training Data
Collected Data
LTRE
SVM
Obs at t
RRE
Prediction ($m_k(t)$)
Obs at $t+\delta$
RRE
Prediction Error ($\hat{m}_k(t+\delta) - m_k(t)$)
$\hat{m}_k(t+\delta)$ is the observed performance for the strategy chosen at time t

The SO is agnostic to the platform and problem domain. All observables, controllables, and metrics are listed in a configuration XML. Controllables list their valid settings, and metrics list their associated weights and costs. The learning and optimization approach is therefore transferrable to other platforms, problem sets, and utility metrics.

The BBN SO is part of a family of research into ML for RF systems starting in 2006 [35, 43, 51–58]. ADROIT [43] was the first to use ML to control a real-world (not simulated) mobile ad hoc network, demonstrating control of network, MAC, and PHY parameters for an application-level map-quality metric. The BBN SO [35] was the first communications EP system to use ML to learn at real time during the mission, demonstrating control of antenna, PHY, and MAC layer parameters for EP. Early work used ANNs and achieved the accuracy at the expense of training time and large training sets. SVMs solved both of those challenges.

*The BBN SO uses $\delta = 1$ for immediate feedback.

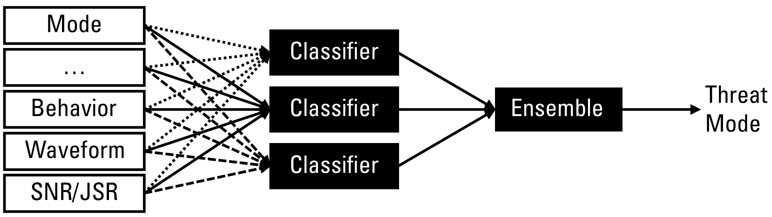


Figure 7.2 Ensemble methods can increase accuracy by selecting from or combining multiple hypotheses. Features can be hand-generated or learned, and any ML method can be used as the intermediate classifier.

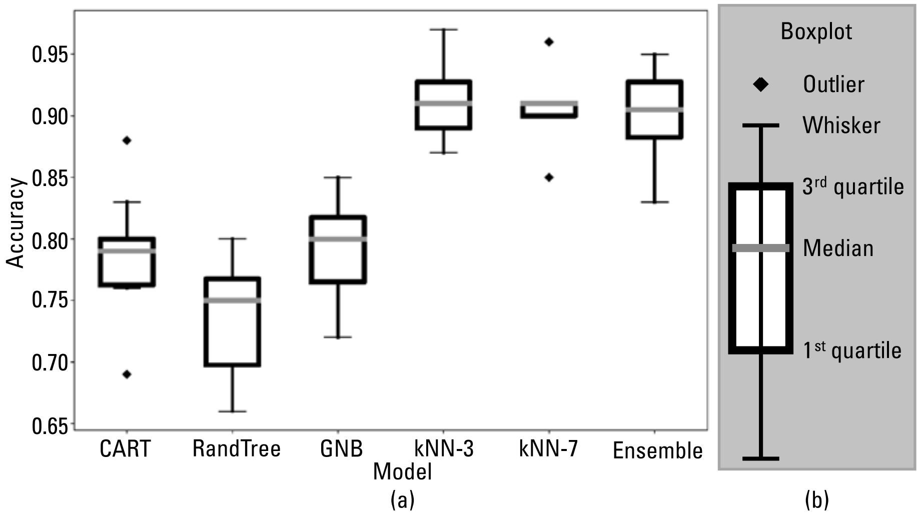


Figure 7.3 Ensemble methods can achieve higher accuracy than independent models. Each boxplot shows the tenfold cross validation scores for the model. (a) A *k*-fold cross-validation trains k models; each model uses a different 1/*k* of the dataset as its test examples. (b) A box-and-whisker plot shows a distribution of values. The central box shows the range of the middle half of the data (first quartile to third quartile), and whiskers correspond to 1.5 times the interquartile range (99.3% of the data if the data is normally distributed). Outliers are shown individually.

7.2 In-Mission Replanning

No plan of operations reaches with any certainty beyond the first encounter with the enemy's main force.

—*Helmuth von Moltke*
Kriegsgechichtliche Einzelschriften (1880)
(Often misquoted as, "No plan survives contact with the enemy.")

In-mission replanning ensures that the plan remains feasible, despite surprises and changes in mission goals. New priorities can emerge, and resources may be

Algorithm 7.1 Ensemble methods can achieve higher accuracy than independent models. scikit-learn contains multiple classifiers and ensemble methods.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import ExtraTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import\
  KNeighborsClassifier as KNN
from sklearn.ensemble import VotingClassifier
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold from
matplotlib import pyplot

# define a fake dataset
def getData():
  X, y = make_classification(n_samples=1000,
            n_features=25, n_informative=15,
            n_redundant=5)
  return (X,y)

def getModels():
  models = []
  models.append(('CART', DecisionTreeClassifier()))
  models.append(('RandTree', ExtraTreeClassifier()))
  models.append(('GNB', GaussianNB()))
  models.append(('kNN-3', KNN(n_neighbors=3)))
  models.append(('kNN-7', KNN(n_neighbors=7)))
  return models

# 10-fold cross validation
def evalModel(name, model, X, y):
  cv = KFold(n_splits=10)
  scores = cross_val_score(model, X, y,
             scoring='accuracy',
             cv=cv, error_score='raise')
  s = '{:10s} mu={:.3f} '.format(
     name,np.mean(scores))
  s = s+'std={:.3f} scores={}'.format(
     np.std(scores),scores)
  print(s)
  return scores
```

```
if __name__ == '__main__':
   names,results=[],[]
   X,y = getData()
   models = getModels()
   for name,model in models:
      scores = evalModel( name, model, X, y )
      results.append(scores)
      names.append(name)
   ensemble = VotingClassifier( estimators=models )
   scores = evalModel( 'Ensemble', ensemble, X, y )
   results.append(scores)
   names.append('Ensemble')

   # plot model performance
   fig = pyplot.figure()
   pyplot.boxplot(results, labels=names)
   fig.savefig('ensemble.png')
```

unexpectedly depleted. Some planning approaches, such as conditional planning and MDPs, incorporate options for *expected* outcomes.

The planner must also always update the plan to handle *unexpected* outcomes (both positive and negative). Figure 7.4 illustrates a possible scenario where replanning adapts to the situation.

Planners can incorporate flexibility to surprise as part of the decision criteria. Planners can also plan to communicate to handle *expected* changes in mission goals [11].

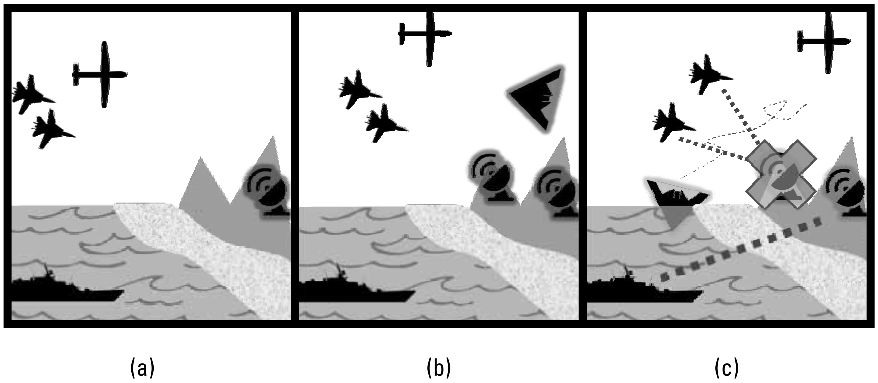


Figure 7.4 In-mission replanning must adjust to unexpected situations: (a) a priori ELINT anticipated one enemy radar; (b) in-mission collaborative ES detected two enemy radars and an aerial drone; and (c) in-mission replanning adjusts, and the nodes neutralize the enemy.

In conditional planning, the planner prepares actions for multiple possible outcomes (usually the most mission-critical), leaving other actions for dynamic replanning because they are less important, less likely, or easy to handle at runtime [12].

In all cases, execution monitoring triggers the replanning process. An action might have failed to achieve expected results, and it must be repeated or replaced [e.g., BDA indicates that the chosen EA method wasn't sufficiently effective and that a different technique(s) must be used]. The existing plan may no longer be achievable, such as when a platform failed and tasks need to be reallocated. The existing plan may no longer be relevant, because the mission goals changed, and a completely new plan must be generated.

Engineers can often estimate the *types* of failures, even if *specific* failures are hard to foresee. For example, one can anticipate platform attrition but not know which platforms (and thus which tasks) will be compromised.

Planners have the option to perform a complete replan from the current state, or repair the existing plan [13, 14]. The decision can depend, for example, on the algorithmic approach, computation time, or desire for plan stability. Anytime approaches manage DM when time is limited (Section 5.3). Plan-repair techniques usually depend on the chosen planning approach, for example:

- Interleaving planning and execution [15, 16];
- Contingent planner with sensing [17, 18];
- HTN planning [19–21];
- Probabilistic planning [22];
- Task allocation [23, 24],
- Graph planning [25, 26];
- Analogical planning [27];
- Plan editing [28, 29].

A common theme of these approaches is that of *plan stability,* wherein the new plan is similar to the original [26–29]. This approach reduces computation, reduces coordination among participants, and supports human operators more effectively: humans do not like significant changes to plans without clear justification.

AFRL's Distribute Operations (DistrO) program addresses this concept directly, looking at plan creation and plan repair [30]. Components assess progress toward goals, identify deviations from expected outcomes, and recommend adjustments to the plan for forward nodes in an anti-access area-denial (A2/AD) environment. A key concept is to minimize adjustments to the plan because nodes have made commitments, and may not be able to communicate any changes.

It may also be useful to factor the cost of changing the plan into the planning process.

7.3 In-Mission Learning

RL, in its broadest definition, is a goal-directed learning approach wherein individuals interact with the environment to improve their performance over time [31]. (In contrast, a *supervised learner* is passive and given the examples; see Chapter 3.) In EW, RL means that the system can take an EP or EA action in the environment, collect feedback, and evaluate its own performance.

Real-world environments are usually too complex to collect data that covers all expected situations. In EW, moreover, systems will encounter novel conditions that cannot be captured in any lab setting. RL is related to direct adaptive control of nonlinear systems [32] and model-predictive control (MPC) [33, 34]. The critical difference is that MPC is model-based, while RL is not. RL systems do not have a differentiable objective function and must take an action in the environment to learn the function. RL allows the system to learn in situ, where learning is most beneficial, and most needed.

RL can update action descriptions to capture what actually happens in the environment—for example updating transition probabilities $P(s'|s,a)$, or action preconditions (p_1,p_2,p_3) to (p_1,p_2,p_4), for $p_i \in$ state s. RL is the only way to accurately learn performance models $m = f(\mathbf{o}_n,\mathbf{c}_n)$, as introduced in Section 4.2.

Figure 7.5 presents a simplified example of in-mission incremental learning. At time $t - \delta$, the optimizer uses the previously learned model f to estimate the performance m of each candidate strategy s, given the known observables o: $m = f(\mathbf{o}_n,\mathbf{c}_n) = f(o,s)$. After choosing and then executing the best strategy $s = \text{argmax}_{s_i} \tilde{U}_n(s_i)$, the system observes the actual performance of s_i. The learner adds the result to the dataset, updates the model f, and the optimizer starts using the updated function f. In the sketch, if $\mathbf{o}_n(t) \approx \mathbf{o}_n(t - \delta)$, then the optimizer chooses the indicated best strategy.

The two key actions in every RL system are (1) what actions to take and (2) when to update models. Figure 5.2 illustrates an example control flow that incorporates in-mission learning of performance metrics.

Most RL systems *update models* at every time step, but the choice can be deliberate. For example, the system might trigger retraining in batches if compute cycles are limited, or if there is new information. Essentially, every expectation or prediction failure is an opportunity to learn. New information includes performance metric feedback for new actions in a known environment, known actions in new environments, or unexpected feedback for a known action in a known environment. Edge cases, strange sensor readings, and adversarial attacks are also candidates.

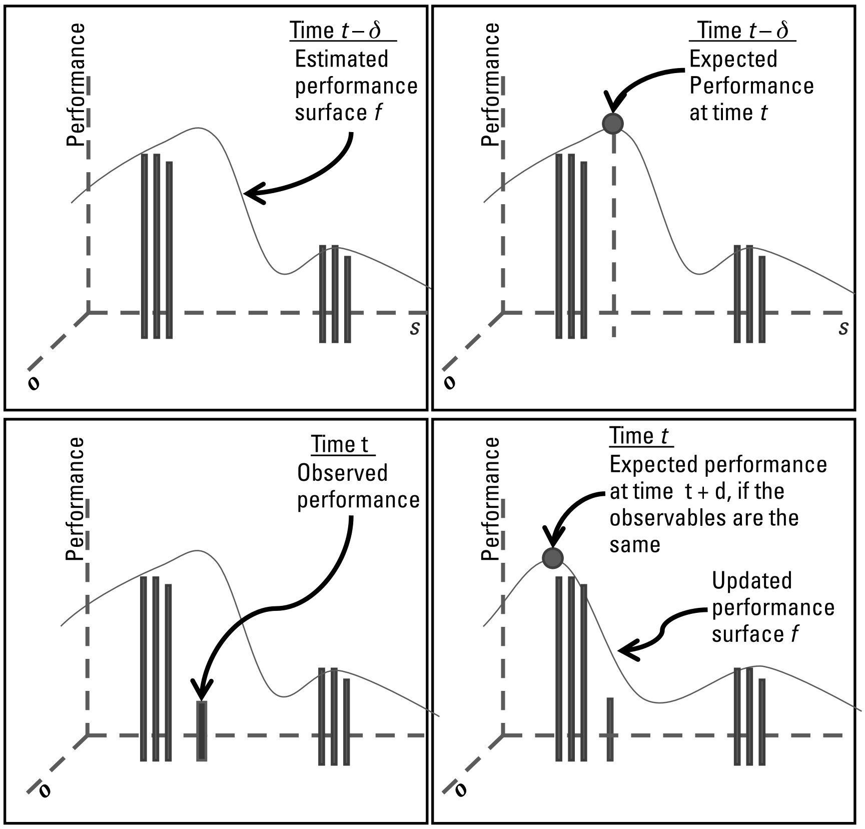

Performance
Time $t-\delta$
Estimated performance surface f
s
o
Performance
Time $t-\delta$
Expected Performance at time t
s
o
Performance
Time t
Observed performance
o
Performance
Time t
Expected performance at time t + d, if the observables are the same
Updated performance surface f
o

Figure 7.5 At time $t - \delta$, the BBN SO (Example 7.1, [35]) uses the model $m_k = f_k(o,s)$ to estimate m_k based on the (known) observables and all (candidate) strategies o,s, and chooses the best strategy. The learner updates f_k for the next iteration. To simplify the illustration, the x- and y-axes are multidimensional: x captures all values of all controllables, and y captures all values of all observables. Note also that the sketch shows f_k only for the given o_t–δ and o_t, but f_k models all possible observable values and all strategies for metric m_k.

In terms of *choosing actions,* the decision-maker makes an explicit *explore/exploit* trade-off. Exploitation means that the decision-maker chooses the strategy with the highest expected utility, while exploration means that it chooses actions that will acquire new knowledge. (The utility function can also incorporate value of information directly, per Section 6.1.4.) The decision balances reward maximization based on previously acquired knowledge with attempting new actions to increase knowledge. Even in human DM, the willingness to fail increases learning rate [36].

Active Learning is the RL concept of deliberately choosing what actions to take (i.e., choosing actions that are most useful to learning). For example, one way to manage bias (Section 8.2) and maintain data diversity (Section 8.3.2) is for the decision-maker to record which strategies it has used against which environments, using a sparse matrix such as Table 7.1. The decision-maker clusters environments

Table 7.1
Examples of Sparse Matrix Used by a Decision-Maker to Track Strategies Used Against RF Environments

Env	None	One Value				Two Values					Three Values	
	000	001	002	010	100	011	012	101	102	110	111	112
Env1	Y	Y	Y	Y	Y	Y	—	—	—	Y	—	—
Env2	Y	Y	Y	Y	Y	—	—	Y	—	Y	—	—
Env3	Y	Y	Y	Y	Y	—	—	Y	Y	—	—	—
Unknown	—	—	—	—	—	—	—	—	—	—	—	—

using observable vectors (Section 2.1.1), adding new classes as needed. In Table 7.1, controllables c_1 and c_2 are binary on/off, and c_3 can take the values 0, 1, and 2, yielding 12 strategies. "Y" indicates when the system has performance results for that strategy in that environment, "–" otherwise.

Much of the common research in RL is based on MDPs to the extent that RL is almost synonymous with MDP. It's not: MDPs may or may not be the underlying model.

RL is not defined by the learning method, but by the learning problem and direct interaction with the environment [31].

In fact, MDP-based RL is generally inappropriate for in-mission EW due to computational complexity and the number of training samples required. Section 3.6 discusses some of the algorithmic trade-offs to consider when choosing a model, and what to learn in-mission.

7.3.1 Cognitive Architectures

Many *cognitive architectures* learn rules of one form or another to help make decisions. Many of these have been used in physical domains, such as robot soccer and fighter pilots [37–41]. Section 5.1.3 describes other learning approaches that improve planning performance. RL formalizes the explore-exploit trade-off; the cognitive architecture literature uses similar concepts of *perceptual attention* and *action selection.*

7.3.2 Neural Networks

When time is plentiful, ANNs (Section 3.1.2) effectively learn a variety of tasks in RF in an RL environment [42]. Many examples in Chapter 4 are based on ANNs; the RL framework trains these models using real environmental feedback. Some examples include learning performance [43, 44], avoiding collisions in spectrum-sharing radar [45], interference control in CRNs [46], recognizing signals [47, 48], selecting tasks [49], and detecting and organizing attack types [50]. Reference [50] suggests techniques to address some of the challenges, including computational efficiency and asymmetric data (class imbalance). ANNs have not been incorporated in the rapid real-time EW literature, because they require a lot of data and computational resources; they cannot learn from a single example. Low-shot learning (Section 3.4) is a promising research area for solving this challenge.

7.3.3 Support Vector Machines

SVMs (Section 3.1.1) are a particularly effective method for RL in EW. They learn from small numbers of training examples (even just a single example) and do not require large compute capability. The BBN SO uses SVMs to perform real-time in-mission learning for communications EP [35] and triggers retraining based on prediction error (Example 7.1).

Combining DeepNets with SVMs is a promising area for research [59– 61]. The DeepNet model can be trained a priori on a large corpus of data to extract latent features of RF signals. One would then replace the output layer of the DeepNet with an SVM that can be updated in mission, at timescales relevant to RF missions, to learn how to classify novel emitters. Figure 7.6 illustrates this concept. SVMs learn rapidly from very little data, and model updates can be computed on FPGAs or CPUs in sub-millisecond timeframes.

7.3.4 Multiarmed Bandit

Multiarmed Bandit (MAB) is a classic RL approach where the rewards have probability distributions rather than fixed amounts [31, 62, 63]. Each action (strategy s) is an "arm" and has a reward distribution R_s with an associated mean value μ_s. MABs model a situation where the environment is static and the same decision

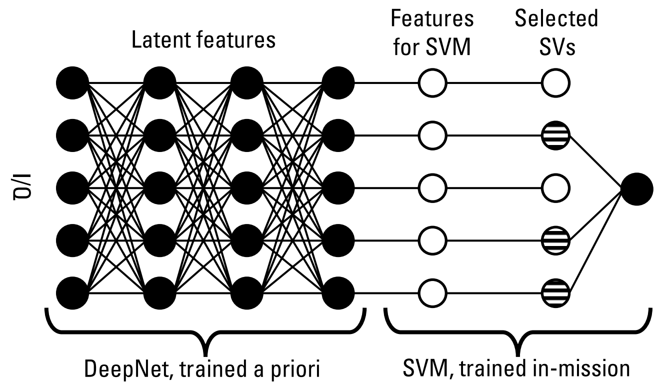


Figure 7.6 Before the mission, a DeepNet learns from a large set of typical environments to extract latent features. During the mission, the system uses the trained DeepNet for inference, and then uses these features to train (and update) an SVM model.

has to be made repeatedly. A bandit can be viewed as an MDP with only one state and multiple available actions; equivalently, an MDP is a set of MAB problems, because the state changes. MAB converges faster than MDPs. A useful variation of MAB adds context to each action, thus learning the reward distribution associated with the context [i.e., for $m = f(o,s)$, the distribution of values for m, when using strategy s under observables o]. MABs have been used for channel selection [64–66], anti-jamming [67], and jamming [68, 69].

Takeuchi et al. [70] develop a simplified MAB algorithm for channel selection that rapidly forgets past parameters and thus reduces computational costs. Several anytime MAB algorithms exist [71–73], correlated to the idea of how much information is enough [74]. Metalearning can also help choose the best MAB heuristic [75], and MAB can become a metalearner to choose effective algorithms [76, 77].

7.3.5 Markov Decision Processes

MDPs are the most common method for representing uncertainty (Section 6.1.3). Q-learning is a model-free RL approach to learn quality (Q-value) of actions; it finds the optimal policy for any finite MDP given infinite exploration time [78]. MDPs have been used within the RL framework to manage spectrum sharing in radar [45], to predict interference [79], for anti-jamming [80], and to understand user behavior [81]. Complementary work includes structuring the utility function to understand how carefully the reward function needs to be maintained [82] and optimizing the utility function while satisfying constraints [83, 84]. Generally Q-learning and other MDP-based methods are inappropriate for EW because they are extremely sample-inefficient and moreover have high computational complexity [45], [85].

Bayesian RL [86, 87] maintains a distribution over key concepts, including model parameters, the policy, or the reward function. The Bayesian posterior naturally captures the full state of knowledge, subject to the chosen parametric representation, and thus, the agent can select actions that maximize the expected gain with respect to this information state.

7.3.6 Deep Q-Learning

Deep Q-Networks (DQNs) use a DeepNet to estimate the Q-values of all possible actions of each MDP state [88] and thus reduce the computational burden (and number of samples required) of conventional Q-learning. The book *Deep RL Hands-on* [89] presents Python examples. DQNs have been used for a variety of RF tasks including signal classification [47], coexistence [90, 91], jamming [92], and anti-jamming [93–99]. Deep active learning approaches use information uncertainty, diversity, and expected model change to select experiments [100]. DQNs have not yet evaluated for sample efficiency in the EW domain where (1) it may be critical to learn from only one sample, and (2) it must meet hard real-time EW requirements.

7.4 Conclusion

No plan survives contact with the enemy, and no model accurately captures every detail of the environment. An EW system must respond to these unexpected, in-mission events and learn from its experiences. This chapter has discussed ways of achieving real-time in-mission planning and learning. Specifically, it focuses on concepts of execution monitoring, in-mission replanning, and in-mission (machine) Learning. Unexpected observations, inferences, and changes must be promptly conveyed to the human users via the HMI (Section 6.3).

Interaction with the real environment mandates constant oversight for planning, and it is the replanning that keeps the mission on track. Moreover, interaction with the real environment enables ML to improve empirical models; in-mission learning improves performance based on real experience. Finally, execution monitoring closes the loop, because it ties (1) the actions to the observations, (2) the DM to the SA, and (3) the EP/EA/EBM to the ES.

References

[1] Lee, G.-H., J. Jo, and C. H. Park, “Jamming Prediction For Radar Signals Using Machine Learning Methods,” *Security and Communication Networks,* 2020.

[2] Melvin, W., and J. Scheer, *Principles of Modern Radar: Radar Applications,* Scitech, 2013.

[3] Aslan, M., “Emitter Identification Techniques in Electronic Warfare,” M.S. thesis, Middle East Technical University, 2006.

[4] Morgan, T., *How Do the Different Radar Modes of a Modern Fighter Aircraft Work?* Accessed 2021-02-02, 2016. Online: https://tinyurl.com/radarmodes.

[5] Avionics Department, “Electronic Warfare and Radar Systems Engineering Handbook,” Naval Air Warfare Center Weapons Division, Tech. Rep. NAWCWD TP 8347, 2013.

[6] Chairman of the Joint Chiefs of Staff, *Joint publication 3-09: Joint fire support,* 2019. Online: https://fas.org/irp/doddir/dod/jp313-1.pdf.

[7] Zhou, D., et al., “Battle Damage Assessment with Change Detection of SAR Images,” in *Chinese Control Conference,* 2015.

[8] Basseville, M., and I. Nikiforov, *Detection of Abrupt Changes: Theory and Application,* Prentice-Hall, 1993.

[9] Choi, S., et al., “A Case Study: BDA Model For Standalone Radar System (Work-In-Progress),” *International Conference on Software Security and Assurance,* 2018.

[10] DoD Army, *Electronic Warfare Battle Damage Assessment.* Accessed 2021-02-02, 2014. Online: https://www.sbir.gov/sbirsearch/detail/872931.

[11] Yordanova, V., “Intelligent Adaptive Underwater Sensor Networks,” Ph.D. dissertation, University College London, London, UK, 2018.

[12] Benaskeur, A., E. Bossé, and D. Blodgett, “Combat Resource Allocation Planning In Naval Engagements,” Defence R&D Canada, Tech. Rep. DRDC Valcartier TR 2005–486, 2007.

[13] Garrido, A., C. Guzman, and E. Onaindia, “Anytime Plan-Adaptation for Continuous Planning,” in *UK Planning and Scheduling Special Interest Group,* 2010.

[14] Cushing, W., and S. Kambhampati, “Replanning: A New Perspective,” in *ICAPS,* 2005.

[15] Haigh, K. Z., and M. Veloso, “Planning, Execution and Learning in a Robotic Agent,” in *AIPS*, Summary of Haigh’s Ph.D. thesis, 1998.

[16] Nourbakhsh, I., “Interleaving Planning and Execution,” in *Interleaving Planning and Execution for Autonomous Robots,* Vol. 385, 1997.

[17] Komarnitsky, R., and G. Shani, “Computing Contingent Plans Using Online Replanning,” in *AAAI,* 2016.

[18] Brafman, R., and G. Shani, “Replanning in Domains with Partial Information and Sensing Actions,” in *Journal of Artificial Intelligence Research,* Vol. 45, 2012.

[19] Ayan, F., et al., “HOTRiDE: Hierarchical Ordered Task Replanning in Dynamic Environments,” in *Planning and Plan Execution for Real-World Systems,* 2007.

[20] Lesire, C., et al., “A Distributed Architecture for Supervision of Autonomous Multirobot Missions,” *Autonomous Robots,* Vol. 40, 2016.

[21] Höller, D., et al., “HTN Plan Repair Via Model Transformation,” in *Künstliche Intelligenz,* 2020.

[22] Chen, C., et al., “RPRS: a Reactive Plan Repair Strategy for Rapid Response to Plan Failures of Deep Space Missions,” *Acta Astronautica,* Vol. 175, 2020.

[23] Beal, J., et al., “Adaptive Task Reallocation for Airborne Sensor Sharing,” in *Foundations and Applications of Self* Systems,* 2016.

[24] Buckman, N., H.-L. Choi, and J. How, “Partial Replanning for Decentralized Dynamic Task Allocation,” in *Guidance, Navigation, and Control,* 2019.

[25] Gerevini, A., A. Saetti, and I. Serina, “Planning Through Stochastic Local Search And Temporal Action Graphs,” *AI Research,* Vol. 20, 2003.

[26] Fox, M., et al., “Plan Stability: Replanning Versus Plan Repair,” in *ICAPS,* 2006.

[27] Veloso, M., *Planning and Learning by Analogical Reasoning,* Springer Verlag, 1994.

[28] van der Krogt, R., and M. de Weerdt, “Plan Repair as an Extension of Planning,” in *ICAPS,* 2005.

[29] Bidot, J., B. Schattenberg, and S. Biundo, “Plan Repair in hybrid planning,” in *Künstliche Intelligenz,* 2008.

[30] Marsh, G., “Distributed Operations,” Air Force Research Labs, Tech. Rep. BAA-AFRL-RIK-2016-0003, 2015.

[31] Sutton, R., and A. Barto, *Reinforcement Learning: An Introduction,* Bradford, 2018.

[32] R. Sutton, A. Barto, and R. Williams, "Reinforcement Learning Is Direct Adaptive Optimal Control," *IEEE Control Systems Magazine,* Vol. 12, No. 2, 1992.

[33] Ernst, D., et al., "Reinforcement Learning Versus Model Predictive Control: A Comparison on a Power System Problem," *IEEE Transactions on Systems, Man, and Cybernetics,* Vol. 39, No. 2, 2009.

[34] Görges, D., "Relations Between Model Predictive Control and Reinforcement Learning," *IFAC-PapersOnLine,* Vol. 50, No. 1, 2017, IFAC World Congress.

[35] Haigh, K. Z., et al., "Parallel Learning and Decision Making for a Smart Embedded Communications Platform," BBN Technologies, Tech. Rep. BBN-REPORT-8579, 2015.

[36] Rzhetsky, A., et al., "Choosing Experiments to Accelerate Collective Discovery," *National Academy of Sciences,* 2015.

[37] Jiménez, S., et al., "A Review of Machine Learning For Automated Planning," *The Knowledge Engineering Review,* Vol. 27, 2012.

[38] Kotseruba, I., and J. Tsotsos, "A Review of 40 years of Cognitive Architecture Research: Focus on Perception, Attention, Learning and Applications," *CoRR,* 2016.

[39] Nason, S., and J. Laird, "Soar-RL: Integrating Reinforcement Learning with SOAR," *Cognitive Systems Research,* Vol. 6, 2005.

[40] T. de la Rosa and S. McIlraith, "Learning Domain Control Knowledge for TLPlan and Beyond," in *ICAPS Workshop on Planning and Learning,* 2011.

[41] Minton, S., et al., "Acquiring Effective Search Control Rules: Explanation-Based Learning in the Prodigy System," in *International Workshop on Machine Learning,* 1987.

[42] Yau, K.-L., et al., "Application of Reinforcement Learning in Cognitive Radio Networks: Models and Algorithms," *The Scientific World Journal,* 2014.

[43] Troxel, G., et al., "Cognitive Adaptation for Teams in ADROIT," in *GLOBECOM,* Invited, IEEE, 2007.

[44] Tsagkaris, K., A. Katidiotis, and P. Demestichas, "Neural Network-Based Learning Schemes for Cognitive Radio Systems," *Computer Communications,* No. 14, 2008.

[45] Thornton, C., et al., *Experimental Analysis of Reinforcement Learning Techniques for Spectrum Sharing Radar,* 2020. Online: https://arxiv.org/abs/2001.01799.

[46] Galindo-Serrano, A., and L. Giupponi, "Distributed Q-Learning for Aggregated Interference Control in Cognitive Radio Networks," *Transactions on Vehicular Technology,* Vol. 59, No. 4, 2010.

[47] Kulin, M., et al., "End-to-End Learning from Spectrum Data," *IEEE Access,* Vol. 6, 2018.

[48] Qu, Z., et al., "Radar Signal Intrapulse Modulation Recognition Based on Convolutional Neural Network and Deep Q-Learning Network," *IEEE Access,* Vol. 8, 2020.

[49] Li, M., Y. Xu, and J. Hu, "A Q-Learning Based Sensing Task Selection Scheme for Cognitive Radio Networks," in *International Conference on Wireless Communications & Signal Processing,* 2009.

[50] Qu, X., et al., "A Survey on the Development of Self-Organizing Maps for Unsupervised Intrusion Detection," *Mobile Networks and Applications,* 2019.

[51] Haigh, K. Z., et al., "Machine Learning for Embedded Systems: A Case Study," BBN Technologies, Tech. Rep. BBN-REPORT-8571, 2015.

[52] Haigh, K. Z., et al., "Optimizing Mitigation Strategies: Learning to Choose Communication Strategies to Mitigate Interference," in *Classified US Military Communication*s, 2013.

[53] Haigh, K. Z., S. Varadarajan, and C. Y. Tang, "Automatic Learning-Based MANET Cross-Layer Parameter Configuration," in *Workshop on Wireless Ad hoc and Sensor Networks,* IEEE, 2006.

[54] Haigh, K. Z., O. Olofinboba, and C. Y. Tang, "Designing an Implementable User-Oriented Objective Function for MANETs," in *International Conference On Networking, Sensing and Control,* IEEE, 2007.

[55] Haigh, K. Z., et al., "Rethinking Networking Architectures for Cognitive Control," in *Microsoft Research Cognitive Wireless Networking Summit,* 2008.

[56] Haigh, K. Z., "AI Technologies for Tactical Edge Networks," in *MobiHoc Workshop on Tactical Mobile Ad Hoc Networking,* Keynote, 2011.

[57] Haigh, K. Z., et al., "Modeling RF Interference Performance," in *Collaborative Electronic Warfare Symposium,* 2014.

[58] Haigh, K. Z., "Learning to Optimize a Network Overlay Router," in *Learning Methods for Control of Communications Networks,* 2017.

[59] Ma, M., et al., "Ship Classification and Detection Based on CNN Using GF-3 SAR Images," *Remote Sensing,* Vol. 10, 2018.

[60] Wagner, S., "SAR ATR by a Combination of Convolutional Neural Network and Support Vector Machines," *IEEE Transactions on Aerospace and Electronic Systems,* Vol. 52, No. 6, 2016.

[61] Gao, F., et al., "Combining Deep Convolutional Neural Network and SVM to SAR Image Target Recognition," in *International Conference on Internet of Things,* IEEE, 2017.

[62] Lattimore, T., and C. Szepesvari, *Bandit Algorithms,* Cambridge University Press, 2019.

[63] Slívkins, A., *Introduction to Multi-Armed Bandits,* 1-2. 2019, Vol. 12.

[64] Jouini, W., C. Moy, and J. Palicot, "On Decision Making for Dynamic Configuration Adaptation Problem in Cognitive Radio Equipments: A Multiarmed Bandit Based Approach," *Karlsruhe Workshop on Software Radios,* 2010.

[65] Liu. K., and Q. Zhao, "Channel Probing for Opportunistic Access with Multichannel Sensing," *Conference on Signals, Systems and Computers,* 2008.

[66] Lu, J., et al., "Dynamic Multi-arm Bandit Game Based Multiagents Spectrum Sharing Strategy Design," in *Digital Avionics Systems Conference,* 2017.

[67] Li, H., J. Luo, and C. Liu, "Selfish Bandit-Based Cognitive Antijamming Strategy for Aeronautic Swarm Network in Presence of Multiple Jammer," *IEEE Access,* 2019.

[68] Amuru, S., et al., "Jamming Bandits: A Novel Learning Method for Optimal Jamming," *IEEE Transactions on Wireless Communications,* Vol. 15, No. 4, 2016.

[69] ZhuanSun, S., J. Yang, and H. Liu, "An Algorithm for Jamming Strategy Using OMP and MAB," *EURASIP Journal on Wireless Communications and Networking,* 2019.

[70] Takeuchi, S., et al., "Dynamic Channel Selection in Wireless Communications via a Multiarmed Bandit Algorithm Using Laser Chaos Time Series," *Scientific Reports,* Vol. 10,

[71] Jun, K.-S., and R. Nowak, "Anytime Exploration For Multiarmed Bandits Using Confidence Information," in *ICML,* 2016.

[72] Kleinberg, R. D., "Anytime Algorithms for Multiarmed Bandit Problems," in *Symposium on Discrete Algorithms,* 2006.

[73] Besson, L., and E. Kaufmann, *What Doubling Tricks Can and Can't Do for Multiarmed Bandits,* 2018. Online: https://tinyurl.com/doubling-mab.

[74] Even-Dar, E., S. Mannor, and Y. Mansour, "Action Elimination and Stopping Conditions for the Multiarmed Bandit and Reinforcement Learning Problems," *Machine Learning Research,* 2006.

[75] Besson, L., E. Kaufmann, and C. Moy, "Aggregation of Multiarmed Bandits Learning Algorithms for Opportunistic Spectrum Access," in *Wireless Communications and Networking Conference,* 2018.

[76] Gagliolo, M., and J. Schmidhuber, "Learning Dynamic Algorithm Portfolios," *Annals of Mathematics and Artificial Intelligence,* Vol. 47, 2006.

[77] Svegliato, J., K. H. Wray, and S. Zilberstein, "Meta-level Control of Anytime Algorithms with Online Performance Prediction," in *IJCAI,* 2018.

[78] Melo, F., *Convergence of Q-Learning: A Simple Proof,* 2007. Online: https://tinyurl.com/qlearn-proof.

[79] Selvi, E., et al., "On the Use of Markov Decision Processes in Cognitive Radar: An Application to Target Tracking," in *Radar Conference,* IEEE, 2018.

[80] Wu, Q., et al., "Reinforcement Learning-Based Antijamming in Networked UAV Radar Systems," *Applied Science,* Vol. 9, No. 23,

[81] Alizadeh, P., "Elicitation and Planning in Markov Decision Processes with Unknown Rewards," Ph.D. dissertation, Sorbonne, Paris, 2016.

[82] Regan, K., and C. Boutilier, "Regret-Based Reward Elicitation for Markov Decision Processes," in *Uncertainty in AI,* 2009.

[83] Efroni, Y., S. Mannor, and M. Pirotta, *Exploration-Exploitation in Constrained MDPs,* 2020. Online: https://arxiv.org/abs/2003.02189.

[84] Taleghan, M., and T. Dietterich, "Efficient Exploration for Constrained MDPs," in *AAAI Spring Symposium,* 2018.

[85] Tarbouriech, J., and A. Lazaric, "Active Exploration in Markov Decision Processes," in *Artificial Intelligence and Statistics,* 2019.

[86] Vlassis, N., et al., "Bayesian Reinforcement Learning," in *Reinforcement Learning. Adaptation, Learning, and Optimization,* Vol. 12, Springer, 2012.

[87] Ghavamzadeh, M., et al., "Bayesian Reinforcement Learning: A Survey," *Foundations and Trendsfiin Machine Learning,* Vol. 8, No. 5-6, 2015.

[88] Mnih, V., et al., "Human-Level Control Through Deep Reinforcement Learning," *Nature,* Vol. 518, 2015.

[89] Lapan, M., *Deep Reinforcement Learning Hands-On,* Packt, 2018.

[90] Maglogiannis, V., et al., "A Q-Learning Scheme for Fair Coexistence Between LTE and Wi-Fi in Unlicensed Spectrum," *IEEE Access,* Vol. 6, 2018.

[91] Kozy, M., "Creation of a Cognitive Radar with Machine Learning: Simulation and Implementation," M.S. thesis, Virginia Polytechnic Institute, 2019.

[92] Zhang, B., And W. Zhu, "DQN Based Decision-Making Method of Cognitive Jamming Against Multifunctional Radar," *Systems Engineering and Electronics,* Vol. 42, No. 4, 2020.

[93] Kang, L., et al., "Reinforcement Learning Based Antijamming Frequency Hopping Strategies Design for Cognitive Radar," in *Signal Processing, Communications and Computing,* 2018.

[94] Bi, Y., Y. Wu, and C. Hua, "Deep Reinforcement Learning Based Multiuser Antijamming Strategy," in *ICC,* 2019.

[95] Ak, S., and S. Bru¨ggenwirth, *Avoiding Jammers: A Reinforcement Learning Approach,* 2019. Online: https://arxiv.org/abs/1911.08874.

[96] Huynh, N., et al., "'Jam Me If You Can': Defeating Jammer with Deep Dueling Neural Network Architecture and Ambient Backscattering Augmented Communications," *IEEE Journal on Selected Areas in* Communications, 2019.

[97] Abuzainab, N., et al., "QoS and Jamming-Aware Wireless Networking Using Deep Reinforcement Learning," in *MILCOM*, 2019.

[98] Erpek, T., Y. Sagduyu, and Y. Shi, "Deep Learning for Launching and Mitigating Wireless Jamming Attacks,"- *IEEE Transactions on Cognitive Communications and Networking,* Vol. 45, No. 1, 2018.

[99] Li, Y., et al., "On the Performance of Deep Reinforcement Learning-Based Anti-Jamming Method Confronting Intelligent Jammer," *Applied Sciences,* Vol. 9, 2019.

[100] Ren, P., et al., *A Survey Of Deep Active Learning,* 2020. Online: https://arxiv.org/abs/2009.00236.

8

Data Management

Data collection and management is perhaps the hardest part of building a system based on AI and ML. A rule of thumb for all ML-enabled systems is that 80% of the work goes into collecting and curating the data, despite the fact that they are only two of the steps in the data lifecycle (Figure 8.1). Chapter 4 discussed the steps related to creating deductions and inferences from the data; this chapter describes how to create high-quality data from the beginning and how to maintain data quality over time.

One of the biggest challenges in EW is data quality: Data is improperly curated, "dropped" because the legacy system doesn't record it, and even gets deliberately destroyed. Data is frequently either not available, not annotated with the right metadata, or not trustworthy. Before creating (or using) a dataset, consider the following questions:

- What is the problem? What is the decision to be informed? What is the task: understand, predict, control, adapt continuously? If you know the question, you can be much selective about what you need. What data is available? Is it sufficient? Is it diverse? If not, can more be collected? What formats does it use? If you build the model, it may help identify where the data is lacking.
- How has the data been curated? Is it inherently biased? Has it been manipulated by an adversary? If you know where the data came from, you know where and how to use it.

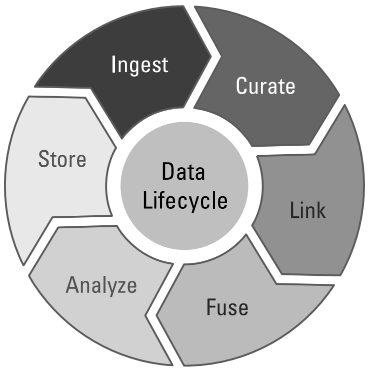


Figure 8.1 The data lifecycle involves collecting and processing the data; iterations increase its utility for multiple purposes.

System requirements rarely capture data needs, such as what data to record, how to tag it, or what properties it must have (e.g., diversity).

This chapter presents both the data management *process* (how to collect and curate EW data), and *practice* (how to use it inside a system to get the desired results). The European Safety Agency presents many of the same ideas from the perspective of safety-critical systems [1].

8.1 Data Management Process

The data "supply chain" is a crucial driver of the quality of an AI/ML system. This supply chain includes the provenance and credibility of the data, its completeness, and its accessibility. If the data (or its metadata) is low-quality, the ML will make poor inferences and, more importantly, be unaware of its own deficiencies. Callout 8.1 outlines the critical actions to take when performing an experiment and collecting data.

Metadata captures the context of the data; semantics establishes the meaning of the concepts in the data; and traceability provides the structure for data integrity. Together, these concepts enable accurate inference and DM, experiment reproducibility, and sharing data across platforms and over time. Moreover, these concepts are key requirements when moving the data through security- or safety-assurance levels: They enable decisions to be made about whether it is appropriate to manipulate the data, whether to grant access to users, or whether the system meets accreditation requirements. Table 8.1 summarizes the DoD's perspective for managing data as an asset. The goals support the purpose of data access, availability, and stewardship. Metadata, semantics, and traceability form the common theme.

Callout 8.1 Establish clear semantics and liberally annotate everything when performing an experiment.

Annotate all data with *metadata* that details how it was collected, including location, type of experiment, and participants.

- Ensure that synthetic and real data have the same structure.
- Version-control *everything*: software, hardware, mission data, and scenario configurations.

Establish clear semantics to support interoperability, across missions, across platforms, and over time.

- Identify all assumptions, both explicit and (if possible) implicit. If the data is synthetic, note all simplifying assumptions.
- Identify unusual cases, such as rare "edge" cases, or adversarial examples purposely constructed to exploit weaknesses.

Table 8.1
Key Goals of the DoD's Data Strategy that Manage Data as a Strategic Asset

Goal	Required Capabilities for Measurable Progress
Visible	Advertised; available; metadata standards; catalogued; common services to publish, search and discover data; governments make decision based on live visualizations
Accessible	Standard APIs; common platforms create and use data; data access is protected
Understandable	Preserves semantics; common syntax; elements aligned; catalogued
Linked	Discoverable and linked; metadata standards
Trustworthy	Protection; lineage; pedigree metadata; data quality
Interoperable	Exchange specifications; metadata and semantics; machine-readable; translation across formats does not lose fidelity; data tagging
Secure	Granular privileges; approved standards; clear markings; authorized users

Summarized from: [2].

From an engineering perspective, modules must provide information that can be directly used by other modules, including data fusion, DM, and the user interface. Figure 8.2 highlights the key concepts that support interoperability and cooperative processes.

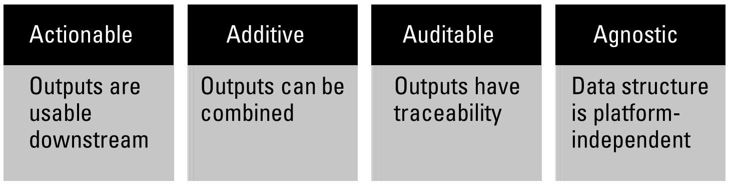

Actionable
Outputs are usable downstream
Additive
Outputs can be combined
Auditable
Outputs have traceability
Agnostic
Data structure is platform-independent

Figure 8.2 To support interoperability, situation assessment and DM modules should produce practical results that are actionable, additive, auditable, and agnostic. (Concepts originally introduced for probability management [3].)

8.1.1 Metadata

Metadata captures all the information about the experiment (and the dataset) that is not inherently part of the dataset. It is the ground truth context of who, what, where, why, and when the data was created. High-quality metadata is a crucial underpinning for cognitive EW systems, supporting a variety of tasks, such as:

- *Accurate root-cause or ground-truth analysis:* Evaluating a cognitive EW system requires knowing what actually happened (e.g., knowing if a packet collision was intentional or accidental. Similarly, for an SEI task, recording the distance from transmitter to receiver enables the experimenter to determine that the model used latent features of the transmitter, and not received power (as a surrogate for distance).
- *Experiment reproducibility:* Evaluating system accuracy requires the ability to rerun experiments, tweak experimental conditions, or evaluate ML models against each other. For example, the system can assess the relative benefit of separating signals by frequency or by cyclostationary processing.
- *Meta-analysis:* Determining data quality involves analyzing datasets for patterns such as distribution, completeness, and systemic bias. Constructing complex scenarios that combine features of (previously) independent datasets involves understanding where and how they interact. Meta-analysis also supports evaluating data integrity: ensuring that incoming or stored data is factual and complete.
- *Future-proofing:* Reducing "bitrot" requires capturing the details of the experiment so it is useful even as equipment or software changes.
- *Interoperability:* Reusing information across platforms or tools requires appropriate tagging of all collected, derived, or inferred knowledge.
- *Task reuse:* Knowing the purpose of a data collection allows developers to evaluate whether a dataset can be reused for a different purpose. For example, a dataset collected for the purpose of waveform classification is unlikely to be useful for SEI or identifying whether nodes are being

jammed. The inverse, meanwhile, may not be true: Specific emitter data may very well be useful for waveform classification. Likewise, knowing which receiver recorded which data for SEI enables developing receiver-independent models.

- *Data sharing:* Sharing data with other organizations for research and operational purposes enables cooperative missions and fosters collaborative research. Metadata helps decide whether, and how, to grant access to sensitive data.
- *Supply-chain evaluation:* Knowing the provenance and credibility of the data allows ES to make correct inferences, EP/EA/EBM to make correct decisions, and system designers to determine whether to use the data at all. Manipulation can occur at any point in the supply chain, via hardware, software, in transit, or in storage.

One of the barriers to developing cognitive EW systems is that data is poorly collected, poorly tagged, and/or proprietary. Good metadata can help eliminate this barrier. Along with the completeness, bias, and appropriateness of the data for new tasks, metadata supports evaluation of the cognitive EW system.

Metadata should annotate the dataset's macro features, such as version, author, and collection date, along with the purpose and goals of the scenario. Synthetic and real data should have the same structure. Examples of these global features include:

- Mission type and sequencing;
- Traffic type and patterns;
- Location (e.g., indoor, outdoor, lab, anechoic chamber, GPS coordinates, and topography);
- Conditions (e.g., weather, mobility, and rural/urban);
- Experiment design (e.g., controlled or "in-the-wild");
- Experiment scenario (e.g., number of emitters and their types/roles);
- Hardware and software components (e.g., versions, capabilities, and measurement error);
- Generation/collection style [e.g., over-the-wire, over-the-air, or synthetic (and the software)];
- Interdependencies, both within and between datasets;
- Collection protocols (e.g., continuous/sporadic experiments, reset/persistent configuration);

- Assumptions, both explicit and (if possible) implicit, particularly simplifying assumptions for synthetic data.

The more detailed the annotations, the more useful the data will be for multiple scenarios. A detail like *this cable connected part A to part B inside radio C* may seem superfluous or overengineered during an initial collection, but having this record allows future analysis and reuse. A detail such as whether data was collected in binary, integer, fixed-point, or *n*-bit floating point allows others to immediately triage whether the data applies to their situation. (Many public datasets are recorded in floating point, which is not compatible with FPGAs.)

Figure 8.3 illustrates a notional (complex) example based on air traffic control radar. A final air traffic control display pulls together information from many different sources, including surface-movement radar (SMR), airport-surveillance radar (ASR), multilateration sensors around the airport, automatic dependent surveillance–broadcast (ADS-B) sensors, and flight plan data.

Each inferred flight track is based on a complex web of connected components. Multiple platforms collect different types of raw data, make intermediate inferences, and communicate them to other platforms. Different data formats, communications media, and communications protocols move the inferences together, leading to final track inferences that support warning systems and air traffic control. *Each piece* of this complex system—hardware, firmware, software, communications, displays—must be certified to meet aviation safety standards because each piece impacts the accuracy of the final inferences and the degree to which stakeholders will trust the system. The degree to which it is recorded impacts how easy it is to analyze the data and identify faults.

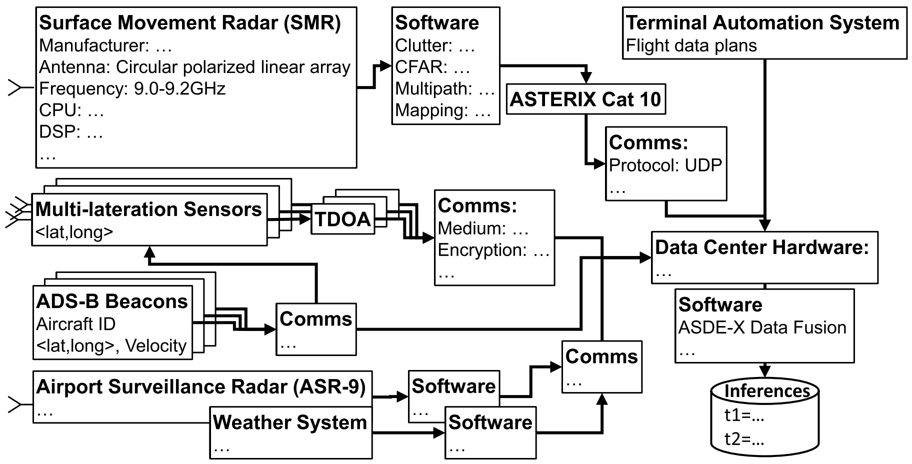


Figure 8.3 A complex web of components lead to high quality tracks and overall air traffic safety.

Annotate everything: all observations and conclusions made by human observers and software components, such as the purpose of sending and the inferences made. For example, an experiment collected at the beginning of the day may have different artifacts than after continuous usage when the hardware is warm. It's particularly useful to identify unusual cases, such as rare "edge" cases, adversarial cases purposely constructed to exploit weaknesses.

Version-control everything: software and hardware configurations, customizations and mission scenarios. It is hard to over-emphasize this requirement.

> A cognitive radar requires a means for evaluating and ensuring the reliability of its knowledge sources: both sources of past knowledge, provided through access of databases, as well as knowledge learned through operational experience. This includes considering not only the possibility for deception, but also whether the validity of data degrades over time.
>
> —*Gürbüz et al., 2019 [4]*

Northeastern University's (NEU) RF fingerprinting project provides a concrete example of metadata for a comprehensive experiment [5]. The metadata captures a description of the original dataset, the preprocessing steps, the model architecture, and methods for training and testing the data, as summarized in Table 8.2. To evaluate RF fingerprinting and ensure repeatable experiments, NEU records detailed features of every experiment [5]. Each component has associated parameters (e.g., slicing chooses random examples from a configurable fraction of data). From this structure, a given experiment can evaluate the effect of channel effects, number of I/Q samples per burst, and whether signal onset impacts results. With a tag indicating that the training set contains transmitters 1,...,*k* and the test set contains transmitters *k* + 1,...,*n*, we can evaluate the impact of diversity on the ability to learn novel emitters [6].

Table 8.2
NEU Records Detailing Features of Every Experiment to Ensure Repeatable Experiments

Preprocessing	Model	Learning Task
Band-filtering Partial equalization Slicing	Full ANN architecture (e.g., layers, activation functions, batch normalization, and dropout) Optimizer (type, metrics, loss) Training platform(s) (CPU, GPUs) Trained weights	Evaluation scenario Training process (batches, epochs, normalization, stopping, etc.) Training population (devices, # examples, etc.) Validation population Test population Results (accuracy, timing, etc.)

Derived from: [5]

The raw data itself should ideally record the details of each transmitter and each receiver, including information such as time-synchronous transmissions. These details enable analysis of the impact of mobility, transmitter versus receiver chains, and simultaneous signals.

Time is one of the most important features to record faithfully.

When time is not recorded accurately, it becomes very challenging for the AI to recover information. For example, a summary of a pulse burst may have an inconsistent TOA compared to the TOA of the constituent pulses.

Hardware layouts, and software versions, should likewise be detailed. A major gap to developing AI systems that understand their own limitations is that key information is lost along the processing chain. For example, while the system integrators know the angle and range measurement error for the radar system, this information needs to be available to the AI. If the metadata captures the uncertainty in raw measurements, software modules (especially data fusion) can estimate the error and confidence of the final inferences.

Figure 8.4 depicts a rudimentary, singular universal software radio peripheral (USRP) radio setup to demonstrate the locations for the USRP hardware driver (UHD) and FPGA-based applications. On the server side, there are several options to talk to UHD, with raw C/C++ encoding having the best performance but requiring one to write all signal-processing blocks from scratch. GNU radio packages (C/C++ or Python) are quicker to adopt, but may not have the same efficiency. On the USRP side, the RF network on chip (RFNoC) utility can be

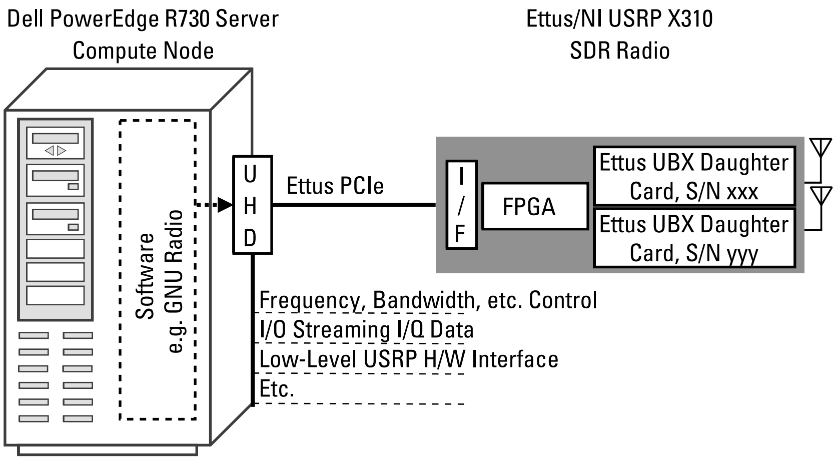

Dell PowerEdge R730 Server
Compute Node
Ettus/NI USRP X310
SDR Radio
Software
e.g. GNU Radio
U
H
D
Ettus PCIe
I
/
F
FPGA
Ettus UBX Daughter Card, S/N xxx
Ettus UBX Daughter Card, S/N yyy
Frequency, Bandwidth, etc. Control
I/O Streaming I/Q Data
Low-Level USRP H/W Interface
Etc.

Figure 8.4 The metadata should contain details of the hardware, firmware, and software in this USRP structure, including versions and serial numbers.

used to create FPGA applications to abstract some of the FPGA-based processing. The UHD software application programming interface (API) supports application development on all USRP software-defined radio (SDR) products [7]. Using common software API improves code portability, allowing applications to transition seamlessly to other USRP SDR platforms when necessary. It reduces development efforts by allowing the users to preserve and reuse their legacy code so that they can focus on the development of new algorithms.

8.1.2 Semantics

Well-established data semantics is a crucial piece of supporting tasks such as interoperability, task reuse, and data sharing. Good semantics allows data to be reused across systems and over time. The data set can grow, even as components (software, hardware, platforms, and missions) evolve.

Two orthogonal tools capture data semantics: ontologies and schemas. Ontologies are dictionaries that capture the meaning of concepts. Schemas are the formats that capture the data.

An *ontology* represents information that captures semantics. An ontology is a formal naming and definition of the terms and categories to describe the concepts, data, and entities of a problem domain and the relationships among them. A *domain* ontology focuses on problem domains, while a *process* ontology captures the steps and constraints involved in a processing chain. Ontologies for CR [8–10] and radar [11] exist. Each has a slightly different focus; for example, the SDR forum's ontology [8] focuses on adaptive modulation, while Cooklev and Stanchev's [10] supports radio topology. The web ontology language (OWL) is commonly used to capture the ontology [12], and the Protégé editor is used to create and manipulate it [13]. The main challenge when developing an ontology is to develop it only for the necessary task. It is easy to overengineer the ontology and develop concepts that will never be used.

A formal ontology may be excessive for the task. The crucial point is to establish a clear semantics that cannot be misinterpreted.

The *Signal Metadata Format* (SigMF) [14, 15] is an open standard schema adopted by the GNU Radio community. It captures general information about the collection of samples (e.g., purpose, author, date), the characteristics of the system that generated the samples (e.g., hardware, software, scenario), and features of each signal (e.g., center frequency and timestamp). A variety of extensions to the base format exist. For example, the *antenna* format captures information such as model, frequency range, and gain patterns, while the WiFi extension adds information for frames and packets. An associated python library sigmf makes it easy to use. Annotations allow comments, such as whether it was deliberately injected, or how it was classified by an ML model. Algorithm 8.1 contains a small example. The GENESYS lab at NEU provides two SigMF-compatible datasets for RF fingerprinting of USRP X310 radios [16].

VITA-49 is a complementary standard used for sending RF samples over a data transport [17, 18]; it is designed for data in motion, not long-term storage. Packet types include context, signal data, and control.

Google's open-source protocol buffers *(protobuf)* serialize and share structured data [19]. Protobuf's compactness emphasizes simplicity and performance (particularly for network communication) and maintains backward compatibility. It is not however, self-describing: The structure of the data is external to the data itself. Simple object access protocol (SOAP) [20, 21] is a self-describing (and therefore less-compact) data format.

Algorithm 8.1 SigMF captures information about the collection, the characteristics of the system, and features of each signal. (Example generated by code of README in [15].)

```
{
   "global": {
      "core:author": "jane.doe@domain.org",
      "core:datatype": "cf32_le",
      "core:description": "All zero example file.",
      "core:sample_rate": 48000,
      "core:sha512":"18790c279e0ca614c2b57a215fec...
      "core:version": "0.0.2"
   },
   "captures": [
      {
         "core:datetime":"2020-08-19T23:58:58.610246Z",
         "core:frequency": 915000000,
         "core:sample_start": 0
      }
   ],
   "annotations": [
      {
         "core:comment": "example annotation",
         "core:freq_lower_edge": 914995000.0,
         "core:freq_upper_edge": 915005000.0,
         "core:sample_count": 200,
         "core:sample_start": 100
      }
   ]
}
```

8.1.3 Traceability

Traceability is a necessary component for determining data integrity, both from a security perspective and a learning/inference perspective. The two key points are:

- The system must ensure that the data is not corrupted at ingest, during analysis, in transit, or in storage.
- The system must track the provenance and credibility of the data as it moves through hardware, firmware, software, communications, and humans.

The *provenance* of the data records where the data came from, while its *credibility* tracks how reliable the source is (i.e., that the data represents what it is supposed to represent). While provenance and credibility are often considered a cybersecurity problem, the same issues apply to poorly collected data. In fact, if the system can determine that the data was deliberately manipulated, it can compensate and infer more accurately than when the data collection was sloppy or incomplete.

Never attribute to malice that which is adequately explained by incompetence.
—*Robert Hanlon, Hanlon's Razor*

Good data collection and curation is the first step in creating credible data; each transformation of the data also transforms its credibility. For example, data fusion can fuse information from multiple low-credibility sources to create a relatively high-credibility conclusion. Another example would be improved radar-emitter signal recognition and classification via amalgamation of classifiers' predictions and drone's surveillance data; surveillance can provide visual target confirmation and its location, further increasing the credibility of the predicted data.

Traceability ensures that ES data fusion forms the correct conclusions (Section 4.3) and that DM algorithms manage information uncertainty correctly (Section 6.1.4). Traceability also enables anomalies and errors to be traced to their origin, components replaced, and better results achieved.

8.2 Curation and Bias

Data curation is the activity of validating, organizing, and integrating data. Curation ensures high-quality data from which the system can draw high-quality conclusions. The goal is to eliminate bias and increase the quality of the data. Almost all datasets are biased, in that they do not accurately represent a model's use case, resulting in skewed outcomes, low accuracy levels, and analytical errors. For example, bias can lead to radar tracking errors and multiple tracks generated for the same target. Common forms of data bias include:

- *Sample or selection bias:* When samples do not represent the underlying distribution of the data. This problem occurs due to faulty experiment design (missing key examples), or environmental factors that cause samples to be missed. For example, signal propagation over land is very different than over sea water due to ducting and surface-wave action; experiments and data collection must compensate for these factors.

  Feedback loops (such as those created by RL) can exacerbate selection bias. For example, each favorable use of an ECM technique can reinforce the desire to use the same (or similar) technique, stepping along the gradient toward a local optimum. Without being deliberate about active learning and maintaining diversity, the system can become trapped in that local optimum and never discover the global optimum.
- *Class bias:* When data skews toward a particular set of classes. To the extent that the data covers all forms of RF signal (e.g., frequencies, modulations, and protocols), the more likely that a learned model will capture all the latent features of RF, and not just the features within the overly narrow dataset. (Specialized models for a given type of RF signal are appropriate and should be tagged as such.)
- *Exclusion bias:* Post-collection exclusion of examples from the data, usually during data preprocessing. For example, if a system only records data of "interesting" emitters or those above/below a specific SNR threshold, critical examples are lost. A common structure in EW systems is that the legacy framework will act on the things it knows, and only forward the data it doesn't know how to handle; this model becomes a significant challenge because the cognitive structures only receive the "scraps," and don't have sufficient history or context to make accurate conclusions.
- *Measurement bias:* Miscalibration, sensor bias or misregistration, faulty measurements, mislabeled examples, missing variables, or a difference between the data collected for training versus inference.

  Spatial misregistration, for example, will lead to false radar tracks.

  Resampling in digital signal processing (DSP) can cause significant measurement bias. If the original data is down-sampled to meet computation or memory requirements for a DSP, then it may not lose important features. Likewise, if decimating the data violates the Nyquist criteria, aliasing can create distortions that cannot be corrected downstream.

- *Recall bias:* A type of mislabeling where similar examples receive different labels; in a dynamic system that evolves over time, this is a common problem. The domain may change, and the concept may drift.

  If components are replaced, the meaning of labels may change. For example, signals separated by cyclostationary processing will have different characteristics than signals separated by frequency. Maintaining the metadata ensures that these meanings can be correctly interpreted.
- *Observer or confirmation bias:* Labels based on expected results, not actual results. A common cause of this problem is when the system fails to recognize novel examples (e.g., a new modulation) and classifies them as the most similar previously known example. Single-source errors likewise create a risk that the data encodes artifacts of that particular source, such as when the data was collected by a single receiver.

Systemic bias is different from *noise,* which is the random variability of the domain. It is easier to *measure* and *correct* noise than it is to measure and correct bias. While it may be tempting to remove noise from the stored data, the particular noise elimination technique may in itself be biased. In general, the dataset should contain the noise to reduce overfitting; data augmentation will, in fact, deliberately add noise (Section 8.3.3). The general expectation is that the distribution of the noise in the attributes of the training data will match the distribution of noise during inference [22, 23] and should be verified [1].

However, noise is also a leading cause of classification errors because the model learns to fit the noise and not the underlying concept. Regularization, diversity, and cross-validation are good techniques to ensure that the noise improves the quality of the model.

8.3 Data Management Practice

Data management, both in-mission and postmission, is the foundation to achieving the desired results in practice. Data management requirements are different between algorithm development and in-mission operation. During development, databases have relatively few space and timing constraints and therefore should be comprehensive and thoroughly annotated.

In an *embedded* system, however, space and timing constraints must be considered. *Data diversity* ensures efficient memory use and rapid computation while achieving desired model quality. *Augmenting data* is a good way to ensure data diversity and is particularly effective at handling adversarial settings. *Forgetting* irrelevant data helps ensure model relevance to the current situation. Finally, *data security* ensures that the data remains private.

8.3.1 Data in an Embedded System

Data management in an embedded system has very different requirements compared to huge data compute systems in a server farm. Moreover, many EW systems operate in the tactical edge, where communications is limited or nonexistent, so storage and computation must be strictly local.

Controlling the size of the dataset solves both problems: memory usage and computation time. The BBN SO [24] (Example 7.1) uses a fixed-length circular buffer, maintaining the diversity of instances in memory and forgetting old instances. It records all examples in persistent storage for use in later missions, but these are not accessed in real time. The BBN SO stores k instances, each having $\overline{o_n} + \overline{c_n} + \overline{m_n}$ features (each feature is an eight-bit integer). It computes the dot products of the pairwise distances between instances, and stores the triangular similarity matrix, size $\frac{1}{2}k^2$. To improve efficiency for both memory and compute, the triangle matrix is a fixed-size, one-dimensional array that uses index arithmetic [25]; the index of $x[i;j]$ in the one-dimensional array is $\frac{1}{2}i(i+1)+j$. This similarity matrix then becomes a basis for data diversity, model computation, and managing experiments for active learning. Matrix sizes are controlled and known a priori and can be adjusted to suit the available memory on the platform.

Controlling the size of the dataset ensures meeting memory constraints and improves computation time: Smaller datasets are less likely to trigger cache miss penalties, and it is faster to train the models with smaller datasets.

8.3.2 Data Diversity

In embedded systems with limited memory, it is crucial to use the available memory as efficiently as possible. One of the keys to producing a model with good generalizability is to ensure data *diversity*. Diversity of the training data ensures that the data contains enough discriminative information to identify the important features (i.e., to maximize the amount of information contained in the data). The data should include edge cases and adversarial examples. The goal is to have the model build high-quality abstract representations. The better the generalization, the more likely that the model will perform well against novel environments. Moreover, diversity helps protect against adversarial attacks, as demonstrated by the data augmentation literature (Section 8.3.3).

The dataset does not need to be large, but it must be diverse.

Three examples of three different strategies in three different RF environments (27 examples) is far more useful than a million examples of one strategy in one RF environment, even though each observation of that environment will be slightly different. Figure 8.5 illustrates the 27 points; despite so few examples, it is clear that these examples cover the performance surface better than many

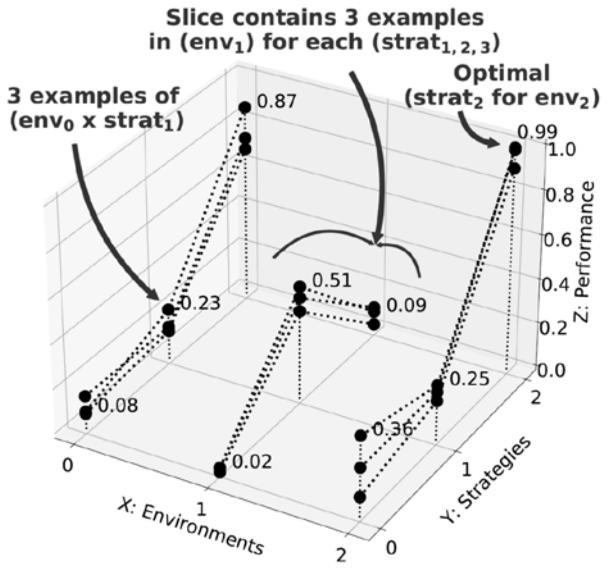


Figure 8.5 Three observations of each of three environments and three strategies yield 27 diverse training points (artificial data). An environment corresponds to a cluster of observable vectors *o*, and a strategy corresponds to a setting of controllables *c*. Figure 4.5 shows several 2D environment "slices" from real environments, showing some environments with no observations where the model generalized from similar characteristics.

examples for a single strategy or a single environment. By maintaining a diverse dataset, the learned models can achieve good performance even with limited examples.

A simple and effective method for computing diversity is simply the pairwise dot products between examples. When a new observation arrives, compute its distance to examples already in memory. If that distance is zero, then it is completely redundant with existing knowledge. Note that the dot product may compute *observables* only, *observables and strategy,* or *observables, strategy, and metrics,* depending on intended use. For example, computing the RF environments (Section 2.1.1) uses only the observables, while managing a diverse training dataset requires also using the strategy and metrics.

Mirzasoleiman et al. [26] develop a rigorous method for choosing a small subset S of the full training data V. The *coreset* of data is a weighted subset S that best approximates the full gradient of V and can be efficiently found using a fast greedy algorithm. At every step, the greedy algorithm selects the element that reduces the upper bound on the estimation error the most. The size of the smallest subset S that estimates the full gradient by an error of at most ε depends on the structural properties of the data. The algorithm obtains significant speed-up in training time while maintaining generalization accuracy.

In fact, diversity is so important that *simulated* diversity yields better results than a nondiverse real-world dataset. Essentially, while the simulated values do not capture the complexity or natural noise of the real environment, they do increase the breadth of knowledge to draw inferences from. Real-world datasets are naturally imbalanced, and hidden war reserve modes exacerbate this imbalance. Simulated data can help "bootstrap" the learning process. Section 11.2.3 describes several RF data-generation tools. Work demonstrating the impact of diversity includes:

- Ablation trials show that training the model using the most diverse dataset yields the best generalization to new environments (Section 10.2).
- "Pseudo classes" cause classes to repulse from each other so that the learned model chooses more discriminative features [27].
- Diversified minibatch stochastic gradient descent (DM-SGD) builds similarity matrices to suppress the co-occurrence of similar data points in the same training minibatch [28]. This approach generates more interpretable and diverse features for unsupervised problems and better classification accuracies for supervised problems.
- Convex transductive experimental design (CTED) [29, 30] incorporates a diversity regularizer to ensure that selected samples are diverse. The approach excludes highly similar examples with redundant information, consistently outperforming other active learning methods.
- Informativeness analysis uses pairwise gradient length [29] as a measure of informativeness, and a diversity analysis forces a constraint on the proposed diverse gradient angle. Empirical evaluations demonstrate the effectiveness and efficiency of the approach.
- Entropy measurement and management [31] controls out-of-sample error; solutions with higher entropy have lower error and handle novel situations better.

Diversity and uncertainty are effective methods to choose experiments in active learning [32, 33].

8.3.3 Data Augmentation

One of the most effective ways of creating data diversity is through data augmentation. Data augmentation is a set of techniques that enables ML practitioners to increase the diversity of the training data, without actually having to collect new data. Augmentation techniques include adding (to the existing dataset) slightly modified replicas of already existing data, or newly synthesized data from existing data.

Data augmentation is closely related to the concept of *oversampling* in data analysis and can reduce *overfitting* during the training phase of the learning model [34]. For example, in image processing, some of the commonly used data augmentation techniques are padding, cropping, flipping, and changing lighting schemes.

Data augmentation techniques are necessary to capture data invariance, the property of remaining unchanged regardless of changes in the conditions of measurement. For example, the surface area of an airplane remains unchanged if the airplane changes its pitch, yaw, and/or roll; thus the surface area of an airplane exhibits rotational invariance. Data augmentation ensures that the model is trained against this data invariance.

Chung et al. [35] present an approach to detecting sample, class, and recall bias using correlation analysis, dynamic bucketization, unknown count estimation, and unknown value estimation. They then use the results to augment the data.

A common theme to many data-augmentation approaches is *adding noise* to the dataset. There can be noise in the *observables* and in the *metrics*. Model accuracy improves by injecting structured noise into the features [36–41]. Gaussian noise is the most common type of noise to add. Noise has a regularization effect that improves the robustness of the model. Adding noise means that the network is less able to memorize training samples, resulting in smaller network weights and a more robust network that has lower generalization error. Consider the following guidelines:

- Normalize the data before adding noise.
- Add noise only during the training phase.
- Noise can be added to the input data features.
- Noise can be added to outputs before training, especially in regression tasks over continuous target values. However, in the cases where it is possible to remove noise from outputs, better results can be obtained [42].
- Noise can be added to the activations, weights, and/or gradients [37].

Soltani et al. [43] propose a data-augmentation technique for the training phase that exposes the RF fingerprinting DeepNet to many simulated channel and noise variations not present in the original dataset. The authors propose two approaches for data augmentation, one for the *transmitter* data (using pure signals without data distortion) and the other for the *receiver* data (using passive, available over-the-air transmitted signals). The authors saw a 75% improvement with transmitter data augmentation and 32%–51% improvement for receiver augmentation.

Sheeny et al. [44] present a radar data-augmentation technique based on the measured properties of the radar signal. Their model achieved only 39% accuracy with nonaugmented data, 82% accuracy with translations and image mirroring, and 99.8% accuracy using data augmented with radar properties of signal attenuation and resolution change over range, speckle noise, and background shift.

Data augmentation doesn't require knowledge of RF spectrum. Huang et al. [45] use rotation, flip, and Gaussian noise to improve classification accuracy for radio-modulation recognition. Modulations can be successfully classified using shorter radio samples, leading to a simplified deep-learning model and shorter inference time. They also show that augmentation improves accuracy at lower SNR.

Data augmentation also helps protect against adversarial attacks. Well-constructed examples train a model to defend against *white-box* attacks (where the adversary knows everything about the model), *black-box* attacks (where the adversary knows nothing), and intermediate *gray-box* attacks [46, 47].

GANs [48] are becoming a de facto method for generating adversarial examples and training models. A key benefit of GANs is that they do not require any information about RF properties. Adversarial training techniques set a *generative network* against a *discriminative network* in a competition: The generative network's goal is to fool the discriminative network. The generative network creates artificial data designed to exploit weaknesses in the discriminative model. One of the most famous GANs generates photos of nonexistent celebrities [49]; humans are unable to distinguish these "photos" from reality. GANs improve accuracy for classification tasks such as modulation recognition [50, 51] and radar target recognition [52]. The generative component has been used for signal spoofing and synthesizing new modulations [53, 54].

8.3.4 Forgetting

The counterpoint to data augmentation is forgetting. There are multiple reasons to deliberately drop data from a training dataset, including limited memory, inaccurate information, and class imbalance. The key issue is to prevent total information loss within a category, either RF environment or strategy (i.e., *maintain the dataset's diversity*).

During a mission, when memory and compute time are limited, simple strategies for choosing which examples to drop from the training dataset in memory include:

- *Distance:* Drop instances that are very similar to other items (observables, strategy, and metrics);
- *Class imbalance:* Drop instances that are well-represented (i.e., have many more examples than other classes);

- *Fidelity:* Drop instances used to bootstrap the learner (e.g., generated from simulated data or collected on similar-but-not-identical platforms);
- *Age:* Drop-the-oldest instances, an approach that is responsive to changing mission conditions.

Note that all examples collected in-mission can (and should) be stored in persistent storage for postmission analysis. Postmission analysis can analyze data for missing data (e.g., performance that can't be explained usually means missing observables), patterns across systems and missions (e.g., something seen only fleetingly at single nodes, but regularly across the group), and best practices (e.g., different methods of curating data).

It is hard to justify permanently forgetting data. Even though data may degrade over time, older knowledge can help bootstrap a system when nothing is known about a set of conditions. This kind of knowledge transfer accelerates learning in novel conditions. Good metadata records the history and context of the data, which premission analysis uses to decide which data to use for training in future missions. We need to overcome the barrier of deliberately destroying data for the sake of security and instead find ways to appropriately protect it.

8.3.5 Data Security

A common issue in data storage is that of data sensitivity, particularly for classified datasets. The system must protect both the *data* (so that adversaries cannot reconstruct the original data) and the *models* so that adversaries cannot determine system vulnerabilities.

Differential privacy (DP) is a mathematical definition of privacy used to prove guarantees of protection against privacy attacks [55]. DP has valuable properties to analyze protection [56], including (1) quantification of privacy loss, (2) group privacy, (3) immunity to postprocessing, and (4) composition and control over multiple computations.

There are many different ways to achieve differential privacy; most involve modifications to the original dataset. Anonymizing the dataset provides privacy protection and facilitates sharing data to other forums. Some approaches include:

- *Label anonymization:* Give features (observables, controllables, and metrics) anonymous names (e.g., "tone jammer" becomes "environment#1," and "notch filter" becomes "technique#1"). Anonymous labeling has the advantage that the decision-maker cannot exploit the semantic meaning of the labels themselves. (In other words, the AI stays agnostic to the particular dataset in question, allowing the same code to transfer to different capabilities, platforms, and tasks.)

- *Normalization:* Normalize the data over a fixed range, say `int8` representation -128 to 127. Moreover, most learning models expect data to be normalized so that large-valued features do not dwarf small-valued features; model generalization is better when trained on normalized data.
- *Generalization:* Round and reduce the precision of data.
- *Discretization:* Replace ranges of values with discrete labels.
- *Perturbation:* Replace values with a uniform function modifying the values. Add noise if appropriate.

There are also methods to learn models that preserve privacy. The goal is to build a model that does not reveal features of the original training data, while still ensuring that any inference based on a specific example does not change. Consider the following:

- The system learns models on fully encrypted data [57–62], and outputs an encrypted result;
- The system partitions the sensitive data and trains an ensemble of "teacher" models; then, using unlabeled nonsensitive data, the system trains a "student" model to mimic the ensemble [63].
- A federated learning system splits training data among many nodes, and then combines the model parameters in a central location before redistributing the blended model back to the nodes [64]. Each node learns only on its own observations, and the central model never sees the original data.

A good security approach requires multiple layers, including standard cybersecurity approaches. One layer of a comprehensive solution is to protect the model and the data to maintain information privacy, even if the model itself is compromised.

8.4 Conclusion

Despite myths to the contrary, the system does *not* need a large dataset to learn effectively. Managing the dataset to maintain diversity is the most effective way to create high-quality generalized models that handle novel situations (out-of-sample) and that are resistant to adversarial attacks. Smaller datasets also have the benefit of improving computation time.

A good data engineer will ensure that the data is high-quality and can be reused on different platforms, for different tasks, and over time. Detailed interactions and use-case development ensures that the EW community and AI community use the same language and have the same goals [65]; we must also address the social gap between the two communities. Detailed metadata ensures high-quality DM and root-cause analysis, allowing the system to propagate errors from measurements to final inferences.

> Garbage-in, Garbage-out.
>
> —*Thomas McRae, 1964, The Impact of Computers on Accounting*

> Life is like a sewer: what you get out of it depends on what you put into it.
>
> —*Tom Lehrer, 1959, We will all go together when we go*

References

[1] Cluzeau, J., et al., *Concepts of Design Assurance For Neural Networks (CoDANN),* European Union Aviation Safety Agency. Online: https://tinyurl.com/CoDANN-2020.

[2] US DoD. (2020). “DoD data strategy.” Accessed 2020-10-08, Online: https://tinyurl.com/dod-data-strategy-2020.

[3] Savage, S., and J. Thibault, “Towards a Simulation Network,” in *Winter Simulation Conference,* 2015.

[4] Gürbüz, S., et al., “An Overview of Cognitive Radar: Past, Present, and Future,” *IEEE Aerospace and Electronic Systems Magazine,* Vol. 34, 2019.

[5] Tong, J., et al., “Deep Learning for RF Fingerprinting: A Massive Experimental Study,” *Internet of Things (IoT) Magazine,* 2020.

[6] Youssef, K., et al., *Machine Learning Approach to RF Transmitter Identification,* 2017. Online: https://arxiv.org/abs/1711.01559.

[7] Ettus Research. (2020). “UHD (USRP Hardware Driver),” Online: https://www.ettus.com/sdr-software/uhd-usrp-hardwaredriver/.

[8] Li, S., et al., *Now Radios Can Understand Each Other: Modeling Language for Mobility,* Wireless Innovation Forum. Ontology from May 08, 2014 on https://www.wirelessinnovation.org/reference-implementations/.

[9] Li, S., et al., “An Implementation of Collaborative Adaptation of Cognitive Radio Parameters Using an Ontology and Policy Based Approach,” *Analog Integrated Circuits and Signal Processing,* Vol. 69, No. 2-3, 2011.

[10] Cooklev, T., and L. Stanchev, “A Comprehensive and Hierarchical Ontology for Wireless Systems,” *Wireless World Research Forum Meeting,* Vol. 32, 2014.

[11] Horne, C., M. Ritchie, and H. Griffiths, “Proposed Ontology for Cognitive Radar Systems,” *IET Radar, Sonar and Navigation,* Vol. 12, 12 2018.

[12] World Wide Web Consortium (W3C). (2020). “Web Ontology Language (OWL).” Accessed 2020-11-07, Online: https://www.w3.org/OWL/.

[13] Musen, M., “The Protégé Project: A Look Back and a Look Forward,” *ACM SIG in AI,* Vol. 1, No. 4, 2015.

[14] Hilburn, B., et al., "SigMF: The Signal Metadata Format," in *GNU Radio Conference,* 2018.

[15] GNU Radio Foundation. (2020). "Signal Metadata Format (SigMF)." Accessed 2020-10-31, Online: https://github.com/gnuradio/SigMF.

[16] Sankhe, K., et al., "ORACLE: Optimized Radio Classification Through Convolutional Neural Networks," in *INFOCOM,* Dataset available at https://genesys-lab.org/oracle, 2019.

[17] Cooklev, T., R. Normoyle, and D. Clendenen, "The VITA 49 Analog RF-Digital Interface," *IEEE Circuits and Systems Magazine,* 2012.

[18] VMEbus International Trade Association (VITA). (2020). "Vita: Open Standards." Accessed 2020-10-31, Online: https://www.vita.com/.

[19] Google. (2020). "Protocol Buffers." Accessed 2020-11-07, Online: https://developers.google.com/protocol-buffers.

[20] World Wide Web Consortium (W3C), *SOAP,* Accessed 2020-12-08, 2020. Online: https://www.w3.org/TR/soap/.

[21] Potti, P., "On the Design of Web Services: SOAP vs. REST," M.S. thesis, University of Northern Florida, 2011.

[22] Quinlan, J., "The Effect of Noise on Concept Learning," in *Machine Learning, An Artificial Intelligence Approach,* Morgan Kaufmann, 1986.

[23] Zhu, X., and X. Wu, "Class Noise vs. Attribute Noise: A Quantitative Study of Their Impacts," *Artificial Intelligence Review,* Vol. 22, 2004.

[24] Haigh, K. Z., et al., "Parallel Learning and DM for a Smart Embedded Communications Platform," BBN Technologies, Tech. Rep. BBN-REPORT-8579, 2015.

[25] Haigh, K. Z., et al., "Machine Learning for Embedded Systems: A Case Study," BBN Technologies, Tech. Rep. BBN-REPORT-8571, 2015.

[26] Mirzasoleiman, B., J. Bilmes, and J. Leskovec, "Coresets for Data-Efficient Training of Machine Learning Models," in *ICML,* 2020.

[27] Gong, Z., et al., "An End-to-End Joint Unsupervised Learning of Deep Model and Pseudo-Classes for Remote Sensing Scene Representation," in *IJCNN,* 2019.

[28] Zhang, C., H. Kjellstrom, and S. Mandt, "Determinantal Point Processes for Mini-Batch Diversification," in *Uncertainty in AI,* 2017.

[29] You, X., R. Wang, and D. Tao, "Diverse Expected Gradient Active Learning for Relative Attributes," *IEEE Transactions on Image Processing,* Vol. 23, No. 7, 2014.

[30] Shi, L., and Y.-D. Shen, "Diversifying Convex Transductive Experimental Design for Active Learning," in *IJCAI,* 2016.

[31] Zhang, Y., et al., "Energy-Entropy Competition and the Effectiveness of Stochastic Gradient Descent In Machine Learning," *Molecular Physics,* No. 16, 2018.

[32] Gong, Z., P. Zhong, and W. Hu, "Diversity in Machine Learning," *IEEE Access,* Vol. 7, 2019.

[33] Ren, P., et al., *A Survey of Deep Active Learning,* 2020. Online: https://arxiv.org/abs/2009.00236.

[34] Shorten, C., and T. Khoshgoftaar, "A Survey on Image Data Augmentation for Deep Learning," *Journal Of Big Data,* Vol. 6, No. 1, 2019.

[35] Chung, Y., et al., *Learning Unknown Examples for ML Model Generalization,* 2019. Online: https://arxiv.org/abs/ 1808.08294.

[36] Brownlee, J. (2018). "Train Neural Networks with Noise to Reduce Overfitting." Accessed 2020-11-22, Online: https://tinyurl.com/noise-and-overfitting.

[37] Goodfellow, I., Y. Bengio, and A. Courville, *Deep* Learning, MIT Press, 2016.

[38] Chen, T., et al., "Big Self-Supervised Models Are Strong Semisupervised Learners," in *NeurIPS,* 2020.

[39] Sohn, K., et al., "FixMatch: Simplifying Semisupervised Learning with Consistency and Confidence," in *NeurIPS,* 2020.

[40] Reed, R., and R. Marks II, *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks,* Bradford Books, 1999.

[41] Bishop, C., "Training with Noise Is Equivalent to Tikhonov Regularization," *Neural Computation,* Vol. 7, No. 1, 2008.

[42] Mirzasoleiman, B., K. Cao, and J. Leskovec, "Coresets for Robust Training of Neural Networks Against Noisy Labels," in *NeurIPS,* 2020.

[43] Soltani, N., et al., "More Is Better: Data Augmentation for Channel-Resilient RF Fingerprinting," *IEEE Communications Magazine,* Vol. 58, No. 10, 2020.

[44] Sheeny, M., A. Wallace, and S. Wang, "RADIO: Parameterized Generative Radar Data Augmentation for Small Datasets," *Applied Sciences,* Vol. 10, No. 11, 2020.

[45] Huang, L., et al., "Data Augmentation for Deep Learning-Based Radio Modulation Classification," *IEEE Access,* Vol. 8, 2020.

[46] Yuan, X., et al., "Adversarial Examples: Attacks and Defenses for Deep Learning," *IEEE Transactions on Neural Networks and Learning Systems,* Vol. 30, No. 9, 2019.

[47] Ren, K., et al., "Adversarial Attacks and Defenses in Deep Learning," *Engineering,* Vol. 6, No. 3, 2020.

[48] Goodfellow, I., et al., "Generative Adversarial Nets," in *NeurIPS,* 2014.

[49] Karras, T., et al., "Progressive Growing of GANs for Improved Quality, Stability, and Variation," in *ICLR,* 2018.

[50] Li, M., et al., "Generative Adversarial Networks-Based Semisupervised Automatic Modulation Recognition for Cognitive Radio Networks," *Sensors,* 2018.

[51] Davaslioglu, K., and Y. E. Sagduyu, "Generative Adversarial Learning for Spectrum Sensing," in *ICC,* 2018.

[52] Majumder, U., E. Blasch, and D. Garren, *Deep Learning for Radar and Communications Automatic Target Recognition,* Norwood, MA: Artech House, 2020.

[53] Shi, Y., K. Davaslioglu, and Y. Sagduyu, *Generative Adversarial Network for Wireless Signal Spoofing,* 2019. Online: https://arxiv.org/abs/1905.01008.

[54] O'Shea, T., et al., "Physical Layer Communications System Design Over-the-Air Using Adversarial Networks," in *European Signal Processing Conference,* 2018.

[55] Wood, A., et al., "Differential Privacy: A Primer for a Nontechnical Audience," *Vanderbilt Journal of Entertainment & Technology Law,* Vol. 21, No. 1, 2018.

[56] Dwork, C., and A. Roth, *The Algorithmic Foundations of Differential Privacy,* Now Publishers, 2014.

[57] Bost, R., et al., *Machine Learning Classification Over Encrypted Data,* International Association for Cryptologic Research, 2014. Online: https://eprint.iacr.org/2014/331.pdf.

[58] Dowlin, N., et al., "CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy," in *ICML,* 2016.

[59] Tang, X., et al., *When Homomorphic Cryptosystem Meets Differential Privacy: Training Machine Learning Classifier with Privacy Protection,* 2018. Online: https://arxiv.org/abs/1812.02292.

[60] Lou, Q., and L. Jiang, "SHE: A Fast and Accurate Deep Neural Network for Encrypted Data," In *NeurIPS,* 2019.

[61] Catak, F., et al., "Practical Implementation of Privacy Preserving Clustering Methods Using a Partially Homomorphic Encryption Algorithm," *Electronics,* Vol. 9, 2020.

[62] Kumbhar, H., and R. Srinivasa, "Machine Learning Techniques for Homomorphically Encrypted Data," In *Applied Computer Vision And Image Processing,* 2020.

[63] Papernot, N., et al., "Semisupervised Knowledge Transfer for Deep Learning From Private Training Data," in *ICLR,* 2017.

[64] Geyer, R., T. Klein, and M. Nabi, "Differentially Private Federated Learning: A Client Level Perspective," in *NIPS Workshop Machine Learning on the Phone and Other Consumer Devices,* 2017.

[65] Haigh, K. Z., et al., "Rethinking Networking Architectures for Cognitive Control," in *Microsoft Research Cognitive Wireless Networking Summit,* 2008.

9

Architecture

Using the deeper understanding of SA and DM developed in Chapters 1–8, we can redraw the components of the cognitive system from Figure 1.4 as illustrated in Figure 9.1. A modular architecture provides the backbone to these capabilities, allowing different techniques to provide different services and ensuring consistent flow of information and control. This chapter briefly touches on software and hardware architecture concerns and provides a short developer's roadmap.

9.1 Software Architecture: Interprocess

While highly adaptive, advanced radar and SDR architectures generally still rely on tailored APIs that expose each parameter separately. This approach is not amenable to real-time cognitive control in EW systems, because the tight coupling means that an AI can't make global decisions.

To eliminate this barrier, the constituent modules must be highly modular and composable. A general interface allows modules to expose their parameters and dependencies and thus enables global optimization and computational load-balancing across multiple processors [1–5]. A modular system also allows for real-time composability of the system, where modules can be swapped in/out depending on mission needs [1]. To this end, we need a *broker* that provides the following.

- *One consistent interface* to any and all modules, so that if a module changes, or a new module is created, none of its controlling or subordinate modules need to be modified. Figure 9.2 sketches the difference between a tightly coupled and brokered interface. This modular approach solves the $m \times n$ problem (i.e., upgrades or replacements of modules).

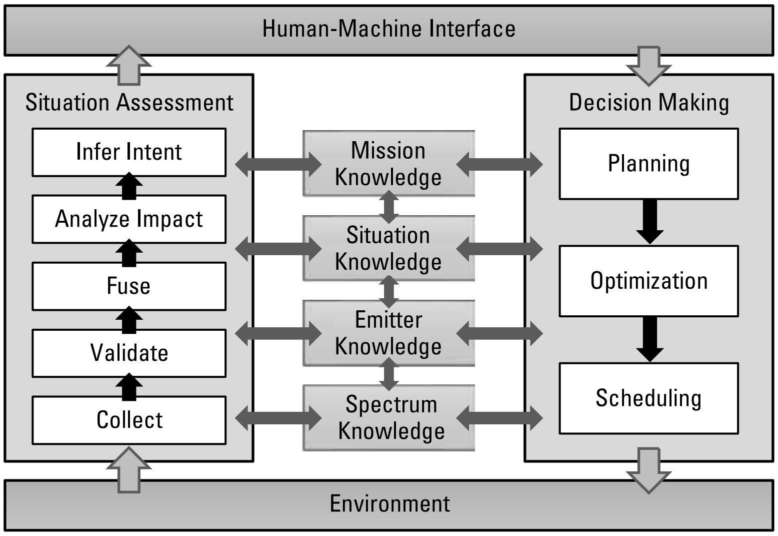


Figure 9.1 DM critically depends on SA.

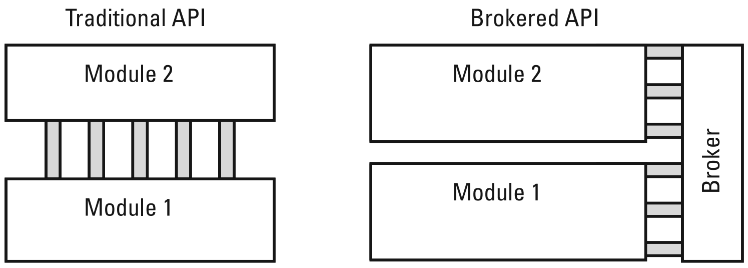
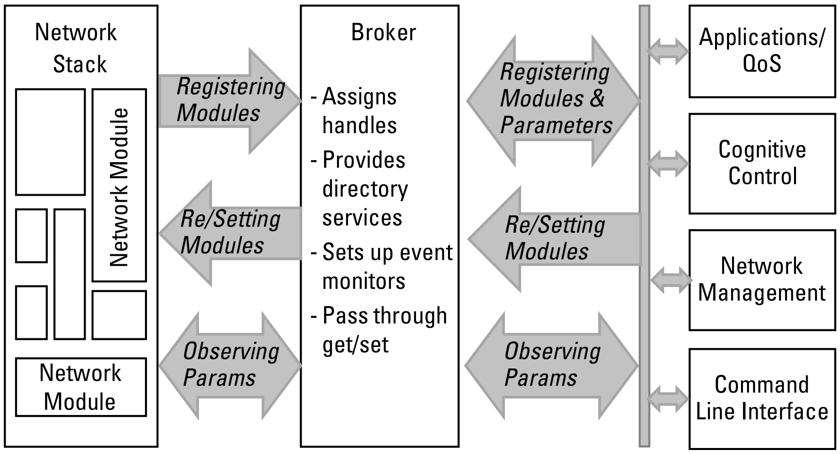


Figure 9.2 Adding or modifying modules in a tightly coupled traditional API requires updates to all connecting modules, while a brokered API doesn't impact other modules.

- *Coordination of control* to ensure that multiple controllers don't overwrite each other. A module may not expose its parameters for outside control via any interface except the broker.

The first software architecture that provided this capability in RF was the ADROIT broker [1], using the abstract structure illustrated in Figure 9.3. Each module exposes its parameters and their properties (especially read/write). When a module changes (e.g., adding a new parameter), it simply exposes the new parameter, `exposeParameter(name,properties)`, rather than adding a new API function for that new parameter. ADROIT instantiated the broker with the particular modules shown in Figure 9.4. ADROIT developers also performed



exposeParameter(*parameter_name, parameter_properties*)
setValue(*parameter_handle, parameter_value*)
getValue(*parameter_handle*)

Figure 9.3 ADROIT was the first network architecture to support cognitive control through a broker. (From [1].)

several detailed walk-throughs showing the separation of concerns of networking modules versus a cognitive layer [1].

The advantages of this generic, modular approach is that it doesn't restrict the form of cognition, allowing designers to choose techniques appropriate to the problem:

- The architecture supports almost *any* cognitive technique;
- The architecture can contain *multiple* cognitive techniques;
- The architecture doesn't *mandate* cognitive techniques.

The modularity allows for fine-grained decomposition of capability; for example, a module might compute and publish one (and only one) statistic (e.g., BER) that becomes an input to many modules.

An interesting side effect of a broker-based approach is that the distinction between "application" and "module" blurs. A given module may request changes not only "down" the stack, but also "up," for example, requesting that the video reduce its resolution to meet bandwidth limitations. In theory, any module can issue instructions to any other module.

Modern RF systems use various "pub/sub" systems such as the data distribution service (DDS) [6, 7] or Kafka [8]. Pub/sub is an asynchronous messaging service that decouples services that produce events (publishers) from services that process events (subscribers). While pub/sub systems are most commonly used to

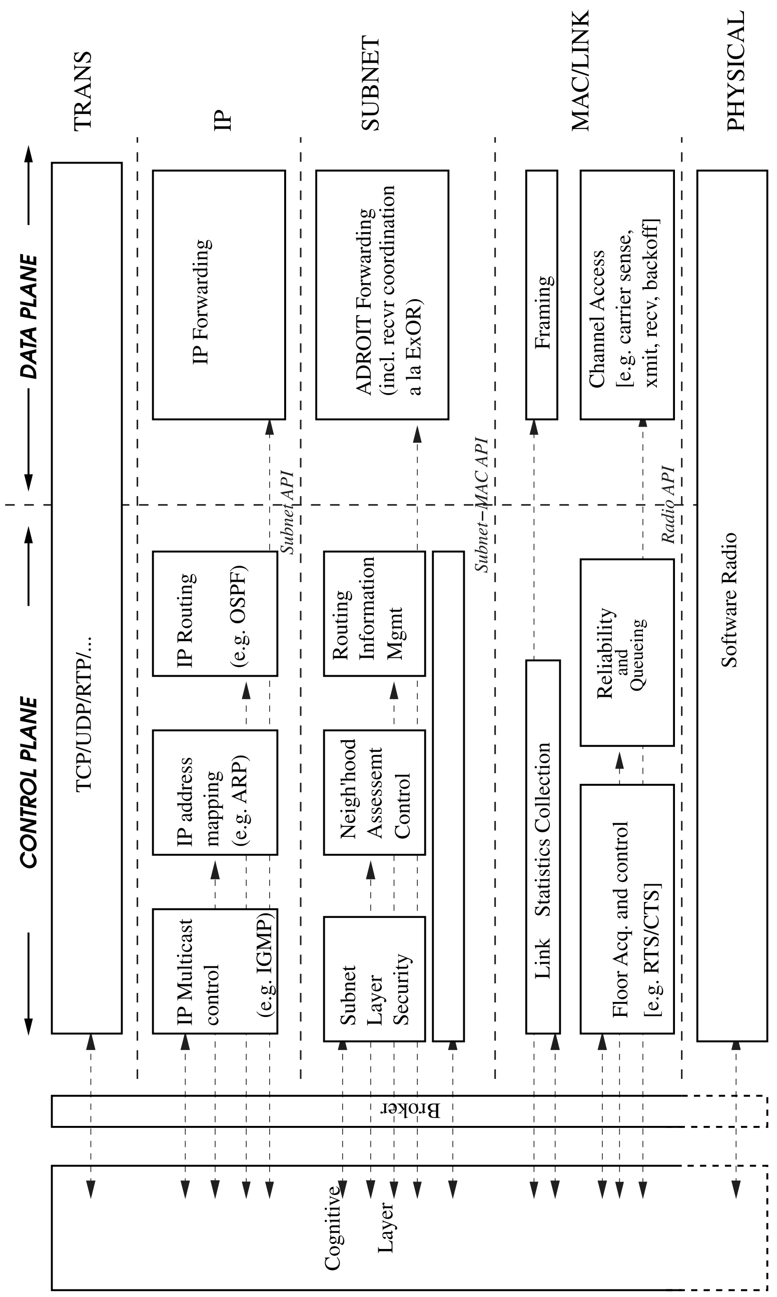

DATA PLANE
CONTROL PLANE
TRANS
IP
SUBNET
MAC/LINK
PHYSICAL
TCP/UDP/RTP/...
IP Forwarding
ADROIT Forwarding (incl. recvr coordination a la ExOR)
Framing
Channel Access [e.g. carrier sense, xmit, recv, backoff]
Subnet API
Subnet–MAC API
Radio API
IP Routing (e.g. OSPF)
IP address mapping (e.g. ARP)
IP Multicast control (e.g. IGMP)
Routing Information Mgmt
Neigh'hood Assessemt Control
Subnet Layer Security
Link Statistics Collection
Reliability and Queueing
Floor Acq. and control [e.g. RTS/CTS]
Software Radio
Broker
Cognitive Layer

Figure 9.4 ADROIT used a broker to provide cognitive control for a modular networking stack. (From [1].)

distribute services across a wide-area network, they are equally effective within a single CPU. Moreover, services can naturally be extended across a bus or multiple cores on a single platform, or across multiple platforms, with increasing latency.

One of the challenges in implementing cognitive control is deconflicting with legacy systems. Ideally, every module in the system should subscribe to the same broker so that every module has the context awareness it requires to make accurate inferences. In practice, however, legacy systems do not connect to a broker, and more importantly, often do not allow the cognitive system to track all the incoming data. A common problem is that the legacy system will respond to all of the RF signals it "knows," only passing on unknown signals. This approach leads to exclusion bias (Section 8.2) and incorrect conclusions due to lack of awareness of longer-term patterns.

9.2 Software Architecture: Intraprocess

Threading and shared memory approaches are appropriate when the interaction between processes needs to be more tightly coupled (e.g., memory access and similar computations). Any instance-based learning method that compares new instances directly against training instances is likely to fall into this category.

The SVMs of the BBN SO (Example 7.1) require the RRE and LTRE to be in separate threads of the same process. Both RRE and LTRE require high-frequency, low-latency access to the same internal datastore; selected instances from the training data become the support vectors. Figure 9.5 illustrates the distribution of function across threads, supporting the functions of Figure 5.2. This effort highlights one of the necessary activities in building the cognitive EW solution:

Translating from common ML libraries to embedded hard real-time software takes significant effort.

The SO's SVM model is derived from WEKA [10], which (at the time) was one of the fastest libraries available, and available in both Java and C/C++. The effort to translate the code for the embedded system involved removing unnecessary code, flattening or replacing object structures, and evaluating numerical representations [11]. The largest effort, by far, was to split the code to be multithreaded: to have the RRE operate in hard real time with guaranteed timing for strategy choices and the LTRE operate in a background thread when computation resources are available. Figure 9.6 illustrates this split. The embedded code ran at 5% of the runtime from the baseline C/C++ code, where baseline disabled the "easy-to-ablate" capabilities. The original (off-the-shelf) Java code had additional items that we could not ablate, such as shared computation across threads, and ran a 1,000,000 times slower than the embedded code. Putting the RRE and LTRE in separate *processes* would likely increase this time yet another > 1,000

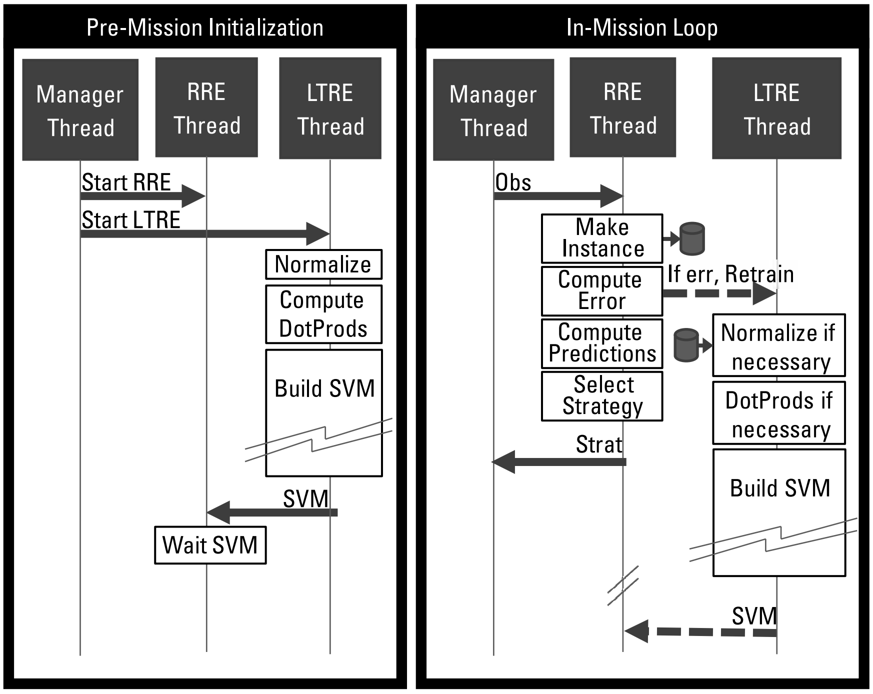


Figure 9.5 The BBN SO uses threads to share data between capabilities. (Example 7.1, [9].)

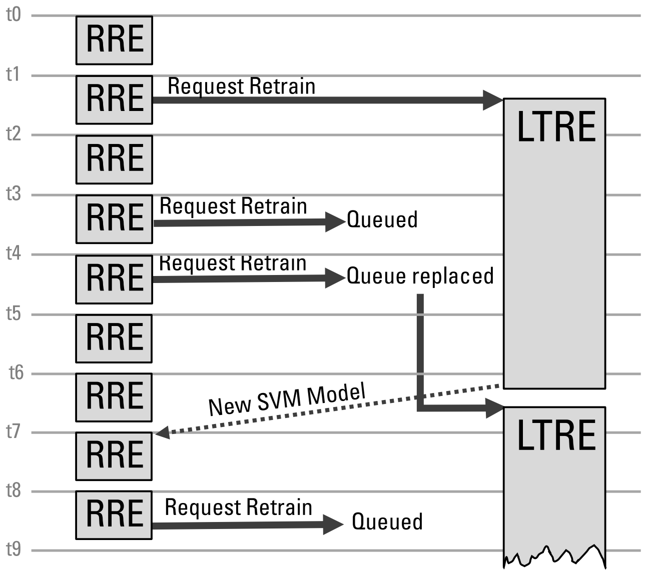


Figure 9.6 The RRE computes strategy choices on a hard real-time schedule, while the LTRE operates in a background thread when computation is available. (Example 7.1, [9].)

times, because the processes would both compute shared values, and the system would incur overhead for sharing the dataset across processes.

9.3 Hardware Choices

Many cognitive EW solutions will be software (or firmware) overlays on existing legacy systems, in which case this section isn't relevant. For those who have a choice, there are several trade-offs to consider.

> Customers are discovering that there is no single 'best' piece of hardware to run the wide variety of AI applications, because there's no single type of AI. The constraints of the application dictate the capabilities of the required hardware from data center to edge to device, and this reiterates the need for a more diverse hardware portfolio.
>
> —*Naveen Rao (Intel), 2018 [12]*

A combination of CPUs and FPGAs will generally be the most appropriate for EW. Callout 9.1 highlights some of the hardware design concerns for EW systems.

> *GPUs are unlikely to be useful for EW operations.*

Table 9.1 summarizes the key features of CPUs, FPGAs, GPUs, and custom ASICs. The main lesson learned is that GPUs will rarely meet the timing considerations of EW: GPUs work well for training DeepNets but are too power-hungry, and unreliable for hard real-time tasks [13, 14]. CPUs and FPGAs will generally be the preferred choice for EW applications. Some of the notable issues include:

Callout 9.1 Embedded EW systems, especially those at the tactical edge, create unique hardware considerations. Wide instantaneous bandwidth exacerbates the processing resources and power issue for EW.

Lack of access to cloud computing requires *local* processing	Limited power from generators and batteries requires *efficient* systems
Space and weight limitations require *compact* solutions	Mission-critical safety and security require *reliable* and *trusted* solutions
Upgrade lifecycle of 10+ years requires scalable, *modular* solutions	Harsh environments with varying temperatures, pressure, shock, and humidity require *ruggedized* systems

Table 9.1
The Key Features of CPUs, FPGAs, GPUs, and Custom ASICs

Processor	Characteristics	Pros/Cons
CPU	• Traditional processor for general-purpose applications • Single- and multicore, plus specialized blocks (e.g., floating point) • Lots of cache; best latency performance from memory to core • Can run independently and host an operating system • Optimized for sequential processing with limited parallelism	• Multiple mission modes; future-proofing • Data locality • Very reliable vendors • Industrial versions available • Software availability and programmer talent • High performance for non-DeepNet ML • High performance on inference that requires a lot of memory
FPGA	• Flexible collection of logical elements that can be changed in the field • Tailor-made architecture configured application • Higher performance, lower cost and lower power consumption	• Reconfigurable • Guaranteed timing performance • Large datasets • Compute-intensive applications • Not good for floating-point operations • Harder to program than other platforms, therefore relatively inflexible
GPU	• Built to do many identical parallel operations (e.g., matrix math) • Thousands of identical processor cores	• Reliable vendors • Most software support for DeepNets • High-power consumption • Computation improvement comes at a cost of memory transfer • Unreliable for hard real-time tasks • Black-box specifications • Shorter lifecycle than CPU or FPGA • Security features less advanced
ASIC	• Application-specific integrated circuity • The most efficient performance-to-power solution	• Single mode (dedicated AI process) • Short lifespan due to rapid technology obsolescence (<1 year) • Limited software support

- GPUs are power-hungry.
- Synchronization is notably difficult, sometimes even causing the CPU to block on *unrelated* tasks.
- The excellent performance for parallel computation comes at a cost of memory transfer, which means that real-time systems suffer from an unacceptable latency.
- The documentation can be inconsistent with, and sometimes *contradictory,* to observed performance.
- The black-box nature of GPUs means that developers do not have a reliable model of GPU behavior, and therefore what system designers learn about current GPUs may not apply in the future.

- GPU life cycles are significantly shorter than CPUs or FPGAs. In the context of DoD programs, this issue makes sustainment more difficult and diminishing manufacturing sources and material shortages (DMSMS) more likely.
- The security feature set of GPUs is significantly less advanced than CPUs and FPGAs, making them more difficult to fit into program protection approaches.

The fact that GPUs are inappropriate for EW is not actually a *loss:*

EW is not a "big-data" environment.

ASICs may have a role in EW, assuming that the short lifespan of the hardware technology, and thus the single application it can host, is acceptable to the community.

CPUs have a long history of good performance in hard real-time systems and are flexible to changes in the software applications over the decade-long life span. FPGAs are ideal because they outperform CPUs for fixed algorithms over streaming data, but the cost (dollars) can be prohibitive. FPGAs are reconfigurable and provide huge power-efficient processing capabilities, reducing thermal management and space requirements. This feature allows the integration of acceleration hardware in small housings and extreme environments.

9.4 Conclusion

From the software-architecture perspective, supporting system-wide cognitive control is best accomplished with a modular infrastructure, wherein each module connects to a backbone and exposes its parameters.

From the hardware perspective, GPUs are rarely appropriate for the embedded EW environment. A combination of CPUs and FPGAs will generally be the most appropriate for EW due to their support for hard real-time operations and streaming computation.

Getting started with cognitive techniques in an EW system is not as complex as many believe; it's a matter of choosing an appropriate-sized problem to tackle.

References

[1] Haigh, K. Z., et al., "Rethinking Networking Architectures for Cognitive Control," in *Microsoft Research Cognitive Wireless Networking Summit,* 2008.

[2] Troxel, G., et al., "Enabling Open-Source Cognitively-Controlled Collaboration Among Software-Defined Radio Nodes," *Computer Networks,* Vol. 52, No. 4, 2008.

[3] Blossom, E., "GNU Radio: Tools for Exploring the Radio Frequency Spectrum," *Linux Journal,* Vol. 2004, No. 122, 2004.

[4] Casimiro, A., J. Kaiser, and P. Verissimo, "An Architectural Framework and a Middleware For Cooperating Smart Components," in *Conference on Computing Frontiers,* 2004.

[5] Hiltunen, M., and R. Schlichting, "The Cactus Approach to Building Configurable Middleware Services," in *Workshop on Dependable System Middleware and Group Communication,* 2000.

[6] Object Management Group, *Data Distribution Service (DDS), version 1.0*, 2004. Online: https://www.omg.org/spec/DDS/1.0.

[7] Object Management Group, *Data Distribution Service, DDS Portal.* Accessed 2020-10-21, 2020. Online: https://www.omg.org/omg-dds-portal/.

[8] Apache. (2020). "Kafka," Online: https://kafka.apache.org/.

[9] Haigh, K. Z., et al., "Parallel Learning and Decision Making for a Smart Embedded Communications Platform," BBN Technologies, Tech. Rep. BBN-REPORT-8579, 2015.

[10] University of Waikato, NZ, *WEKA: The Workbench for Machine Learning.* Accessed: 2020-04-12. Online: https://www.cs.waikato.ac.nz/ml/weka/.

[11] Haigh, K. Z., et al., "Machine Learning for Embedded Systems: A Case Study," BBN Technologies, Tech. Rep. BBN-REPORT-8571, 2015.

[12] Rao, N., *Intel AI—The Tools for the Job,* 2018. Online: https:// tinyurl.com/intel-hw-trades.

[13] Yang, M., et al., "Avoiding Pitfalls When Using NVIDIA GPUs for Real-Time Tasks in Autonomous Systems," in *Euromicro Conference on Real-Time Systems,* 2018.

[14] Maceina, T., and G. Manduchi, "Assessment of General Purpose GPU Systems in Real-Time Control," Vol. 64, No. 6, 2017.

10

Test and Evaluation

AI has unique evaluation requirements, especially if it's going to learn during a mission. This chapter describes how to test and evaluate performance so that customers and end users will trust the system, even when it encounters unexpected conditions. This chapter answers the question of how to test something that changes. The key steps to test and evaluation are:

- Specify the metrics of performance (MOPs) and the metrics of success (MOS) that are relevant to the mission and end user (Section 2.3 and Callout 10.2).
- Use a closed-loop interactive test framework (Section 10.1).
- Use ablation trials to demonstrate that the system will work in previously unknown environments (Section 10.2).
- Compute accuracy of classification, regression, and strategy choice (Section 10.3).
- Use a learning assurance process to formally and empirically validate results (Section 10.4).

The goal of evaluation is to determine how much trust to place in the AI. Simply put, *is the model useful, and making valid decisions in the mission context?* Should you trust the AI enough to report observations? To fly the platform? To target an adversary? To learn rules of engagement?

Trust is a function of risk: The more risk for the trustor and authority granted to the AI, the greater the validation and assurance requirements.

Trust must start with safety and security [1], and the degree of trust depends on the amount of risk the trustor can tolerate, as illustrated in Figure 10.1. The degree of trust determines the amount of authority they will grant to the AI.

10.1 Scenario Driver

ML systems commonly use static datasets to learn how to classify objects. In EW, this approach is insufficient because it doesn't show how the system handles novel examples, responds to dynamic situations, or operates against an adversary. The environment responds to every action.

To validate a cognitive-decision engine, a closed-loop testing framework is essential.

Figure 10.2 illustrates the structure of the scenario driver (SD) that interacts with the cognitive engine, responding appropriately to stimuli.

A ground-truth data file (GTDF) contains all the known data, however it was generated; Table 10.1 shows an example from a real communications EP system. The GTDF might contain raw I/Q samples, PDWs, or inferred features, as required by the AI module. The GTDF consists of all mission scenarios for which

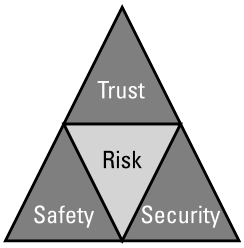


Figure 10.1 Trust depends on safety and security, and depends on the acceptable level of risk.

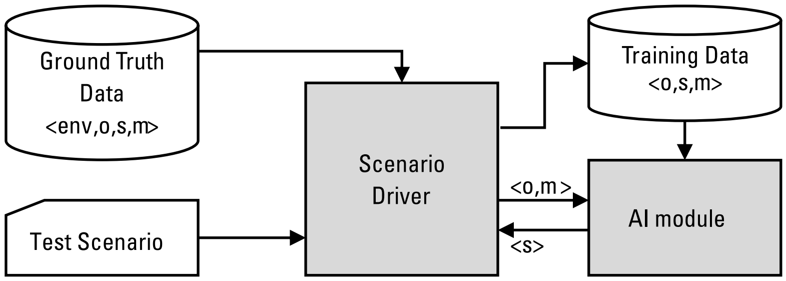


Figure 10.2 A closed-loop scenario driver ensures that a cognitive-decision engine responds correctly to stimuli, sending observables *o* and metrics *m* to the AI module, which computes a strategy *s*.

Table 10.1
The GTDF that Contains All Known Data, Simulated, Emulated, or Real-World.

ENV	SHo1	SHo2	e1o1	e1o2	e1o3	e1o4	e1o5	e1o6	e2o1	e2o2	e2o3	e2o4	e2o5	e2o6	MT1	MT2	MT3	MT4	MT5	m1	m2
env1	-64	-10	123	127	-99	NaN	-127	-127	126	8	127	-127	-43	127	0	0	1	0	0	120	112
env1	-80	-10	123	127	-113	NaN	-127	-127	126	54	127	-127	-24	83	1	0	0	0	0	114	56
env1	-80	-10	126	-20	127	-127	-33	127	126	34	127	-127	-46	127	0	0	0	1	0	127	112
env1	-80	-10	126	23	127	-127	-47	127	123	127	-64	NaN	-127	-127	0	0	1	0	0	120	112
env2	127	-12	126	125	127	118	-127	77	NaN	NaN	NaN	NaN	NaN	NaN	0	0	0	1	0	127	127
env2	15	-56	126	124	123	119	-127	-127	73	127	-118	NaN	-127	-127	0	0	0	1	0	127	127
env2	15	-56	126	125	124	119	-127	-59	87	127	-127	NaN	-127	-127	0	0	0	0	0	88	0
env3	-127	NaN	126	NaN	NaN	NaN	-127	-127	NaN	NaN	NaN	NaN	NaN	NaN	0	0	0	0	0	0	0
env3	-127	NaN	126	NaN	NaN	NaN	-127	-127	NaN	NaN	NaN	NaN	NaN	NaN	0	1	0	0	0	0	70
env4	-64	-14	126	-53	93	-67	60	127	126	-39	88	-113	42	127	0	0	0	1	0	127	127
env4	-64	-14	126	-53	93	-67	60	127	126	-39	88	-113	42	127	0	1	0	0	0	101	70
env4	-64	-14	126	-54	107	-85	64	36	126	63	-64	NaN	-127	-113	0	0	0	1	0	127	127
env4	-64	-16	126	-53	96	-76	72	127	126	-41	111	-42	62	127	0	0	0	1	0	127	127
env5	-48	-14	126	-59	102	104	55	83	126	-54	109	51	40	107	0	0	0	1	0	127	127
env5	-48	-14	126	-66	115	95	68	95	126	-34	46	7	85	127	0	0	0	0	0	88	0
env5	-48	-14	126	-66	115	95	68	95	126	-34	46	7	85	127	0	0	0	0	0	88	0
env5	-48	-14	126	-66	115	95	68	95	126	-34	46	7	85	127	0	0	0	0	0	88	0
env5	-64	-16	126	-37	95	77	60	10	126	11	81	13	46	54	0	0	1	0	0	76	0
env5	-64	-16	126	-37	95	77	60	10	126	11	81	13	46	54	0	1	0	0	0	101	70
env6	47	-10	126	-127	127	127	100	127	NaN	NaN	NaN	NaN	NaN	NaN	1	0	0	0	0	63	62
env6	47	-10	126	-127	127	127	104	127	NaN	NaN	NaN	NaN	NaN	NaN	0	0	1	0	0	101	93
env6	47	-10	126	-127	127	127	76	127	126	-127	3	-127	127	127	0	0	1	0	0	101	93
env6	47	-10	126	-127	127	127	82	127	126	-127	4	-127	127	127	0	0	0	0	1	76	0
env6	47	-10	126	-127	127	127	82	127	126	-127	4	-127	127	127	1	0	0	0	0	63	62

This dataset is normalized to `int8` representation, and represents inferred features for (a) two shared observables that describe the entire RF spectrum, (b) six observables for each of two emitters, (c) five techniques that can be combined, and (d) two metrics.

ground truth conditions are known: all combinations of tasks, nodes (friendly, neutral, and adversarial), and configurations.

An *RF environment* is one of the various test scenario conditions. In the simplest case, it might be free-and-clear communications under clear-weather and low-traffic requirements. Other environments might add mobility, jammers, weather, and topography. During controlled experiments, each environment is tagged with its known environment, and recorded in the GTDF.

If the data is recorded in a comma-separated value (CSV) file, each row captures a single example of the observables, controllables, and metrics $< o,s,m >$, describing how the strategy performed against that environment for node *n*. The observables correspond to a single observation of the environment.

In theory, the GTDF could have a very large number of rows for each tested RF environment and an exponential (or even infinite) number of columns corresponding to every possible strategy. In practice, a smaller, sparse, dataset with high diversity is more valuable (Section 8.3.2). As the mission proceeds, the system collects the data, adds experience, and updates the table. A typical approach might be to collect data for each independent controllable (varying one controllable while the others are set to default values), and a few pairs (e.g., Table 7.1).

A *test scenario* describes which environments to use as training data and which to use as test data. The SD selects those examples of the GTDF that match the training environments and generates a training data file for the AI module to use premission. (Figure 9.5 shows the premission steps for the BBN SO.) The closed-loop test contains the following steps:

- SD generates one example observation of the test data (observables *o* and metrics *m*).
- AI chooses a strategy (or characterizes, for example, the signal) and returns the result to the SD.
- SD evaluates that strategy, and computes the metric *m* at the next iteration and the optimal performance per Section 10.3.3.

The SD plays sequences that interact appropriately with the decisions made by the AI. Using the simple state machine of Figure 10.3, for example, if the AI observes the request-to-send (RTS) and then successfully chooses a jamming technique to block the clear-to-send (CTS), then the SD would replay another RTS. If the AI did not choose to jam the CTS, then the SD would start sending the data. In the radar world, these labels might correspond to radar modes. Each arc indicates the trigger that causes the state to transition.

For open-set classification (Section 3.4), this interactive process supports evaluation of how many new examples the system requires before correctly identi-

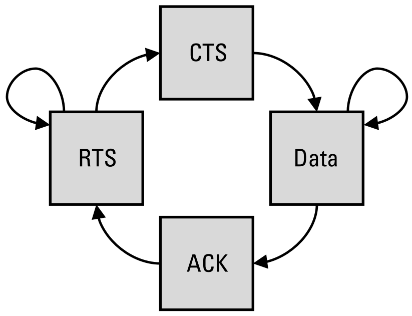


Figure 10.3 The SD can replay data from the GTDF using state machines such as this simple comms sequence. The "environments" in the GTDF would correspond to the RTS, CTS, Data, and ACK labels.

fying a novel class. For ablation trials (Section 10.2), it evaluates how quickly the system learns how to handle a new RF environment.

Structure the SD so that it can replay any kind of data, using any underlying state machine. This approach allows testing of multiple AI modules, for example one that generates I/Q samples to test a deinterleaver, one that generates PDWs to test a classifier, and one that generates PDWs with a state machine to test ECM choices.

Since system requirements documents are often poorly specified with respect to learning, it is crucial to determine early which scenarios to test.

It is impossible to test all ∞ configurations by ∞ scenarios. Identify which of the three axes illustrated in Figure 1.3 are relevant to system performance—and customer needs—and test at multiple points along each axis. Moreover, combinatorial explosion is tempered by the fact that EW engagements are constrained by physics and progression of engagement state by collection/denial of information, thus helping focus test requirements.

Figure 10.4 illustrates a notional architecture for developing and testing multiple AI modules. The SD generates data using the same format as it would arrive in the fielded system. Each module should be independently unit-tested before testing the sequence. When bringing modules into the sequence, replace

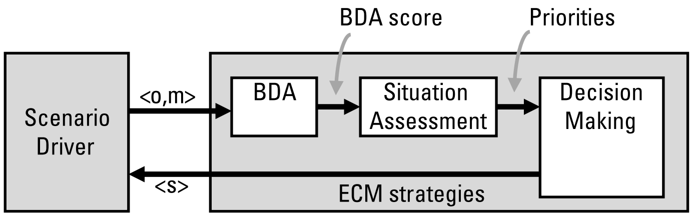


Figure 10.4 The SD can drive testing of multiple AI modules.

other modules with "omniscient" components that are always correct. (For example, to test the SA module, the DM can be a simple set of handwritten rules, and BDA knows exactly how well the chosen ECM worked.) Evaluate the BDA and SA modules using a weighted accuracy (Section 10.3.1 or Section 10.3.2.3), and the DM module using adequacy (Section 10.3.3.1).

10.2 Ablation Testing

n-choose-*k* ablation testing proves that a cognitive system is capable of learning how to handle *new environments.* An ablation trial tests how much given training example(s) contribute to the generalization ability of the model.[1] The ground truth data has *n* known cases; we train the system on $k \subseteq n$ cases, and test on all *n,* for all values of *k* and all subsets $\binom{n}{k}$. Thus, during the test, $n-k$ environments are novel. For $n = 3$ environments known to the SD, there are a total of eight ablation tests:

- $k = 0$. SD creates one test scenario $\binom{3}{0}$. The AI receives no a priori training data; during the test, all $n = 3$ environments are novel.
- $k = 1$. SD creates three test scenarios $\binom{3}{1}$. In each, the AI trains on one environment; during the test, one is known, and two are novel.
- $k = 2$. SD creates three test scenarios $\binom{3}{2}$. In each, the AI trains on two environments; during the test, two are known, and one is novel.
- $k = 3$. SD creates one test scenario $\binom{3}{3}$. The AI trains on all three environments; during the test, only the order of environments is unknown.

Ablation tests are similar to *leave-one-out testing,* which would train on $n-1$ cases and test on the remaining one case. Likewise, *k-fold cross-validation* trains *k* models and tests each on a different $1/k$th of the data. The idea is to demonstrate that the system can learn to handle new environments, regardless of what it was initially trained on.

Figure 10.5 shows *n*-choose-*k* results collected when developing the BBN SO (Example 7.1,[3]). For each of $n = 9$ environments, the SD trains the SO on

1. Ablation studies traditionally remove components to understand the contribution of the component to the system; ablation studies require that the system exhibit graceful degradation as components are removed [2].

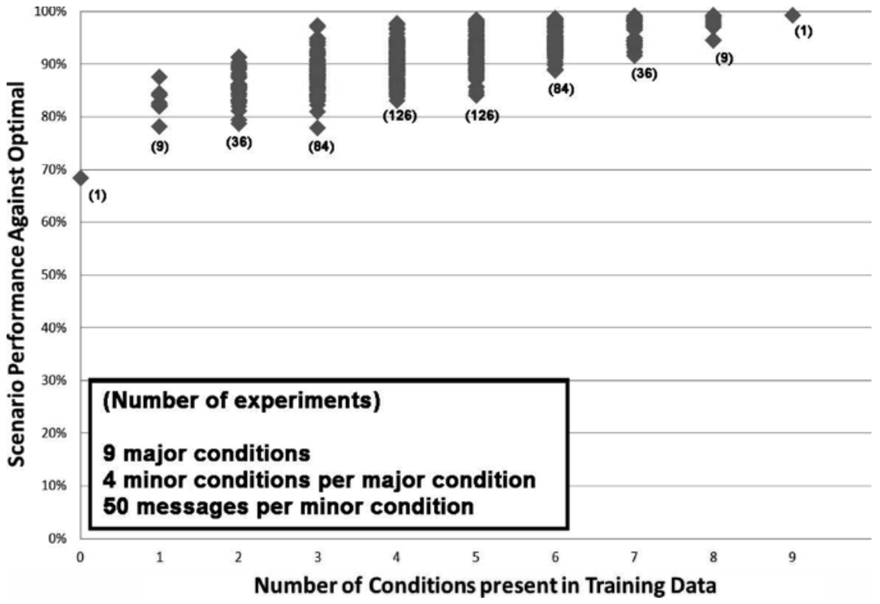


Figure 10.5 An *n*-choose-*k* ablation trial for the BBN SO shows that the performance gracefully degrades as the SO's a priori training data contains fewer conditions. Each point corresponds to the adequacy of one experiment such as Figure 10.8; the x-axis is *k* showing how many conditions were present in the training dataset, and the y-axis is the adequacy value per Section 10.3.3.1. Parenthetical "(x)" values show number of experiments performed for 9-choose-*k*. (Example 7.1, [3].)

all subsets $k \subseteq n$, and tests on all nine environments. This chart shows 512 individual tests, ranging from the case of $k = 0$ that has no a priori training data, to the case of $k = 9$ that trains on all $n = 9$ environments, but during the test the SO doesn't know their order.

Even in the extreme case when the SO receives no a priori training data, it achieves 70% of optimal performance due to in-mission learning.

Figures 10.8 and 10.9 show single scenarios each corresponding to one point on Figure 10.5, and highlight how *in-mission-learning can learn how to handle a novel environment in only one or two examples.* With about 30% of the training data, the SO can achieve 98% adequacy in the best case; in the worst case, it achieves 78% adequacy.

This difference is explained by diversity: *which* environments were used in the size-*k* training data. The worst case uses three very similar environments, while the best case uses three very different environments. Section 2.1.1 describes how to compute environment similarity: Create unsupervised clusters of RF environments that can be visualized with a dendrogram. The dendrogram corresponds to observables for the *n* known environments, and the scenario determines which of these to use for training and testing.

Because it is unsupervised, the EW decision-maker can use this clustering and associated dendrogram to drive active learning (Section 7.3).

The SD can also use this clustering to evaluate performance against optimal (Section 10.3.3.1); the clusters are named based on the ground-truth labels. Using Figure 2.3 as an example, a good test to see how well the system performs on extremely different data would train the models on the 15 environments in the top cluster (16 to 23 vertically), and then test on eight environments the bottom cluster (22 to 08).

Referring again to Figure 10.5, the best case might correspond to the three very different environments 16, 23, and 22 because the model generalizes effectively against all 20 of the other environments. Meanwhile, the worst case might correspond to training on the three similar environments 22, 02, and 07, causing the system to take longer to learn how to handle the novel situations, and thus receive a lower adequacy score.

In Algorithm 4.1, during step 4 (classify new radars) the AI-system-under-test does not know the ground-truth environment, but the SD can correctly evaluate the performance of an open-set classifier. The score of the classifier depends on the environments in question. For example, it may be acceptable to consider environments 05 and 06 as the same, especially if the environments represent jammers and the chosen mitigation techniques have the same performance. Likewise, an open-set classifier should be able to recognize a novel transmitter more quickly when it is really different than when it is only slightly different from known examples.

Ablation tests demonstrate empirically that the cognitive system can learn to generalize from its experience to handle novel environments.

10.3 Computing Accuracy

EW systems use both regression and classification learning approaches. Regression models predict numerical values [i.e., $y = f(x)$ for $y \in \mathbb{R}$]. The accuracy of a regression model is evaluated using normalized root-mean-squared error (RMSE) (Section 10.3.1).

Classification models label discrete classes [i.e., $y = f(x)$ and $y \in S$ for a discrete set of classes S]. Classification models are evaluated using confusion matrices and related statistics (Section 10.3.2).

In EW, we also capture cases when multiple EP/EA strategies yield different performance for different RF environments. For example, multiple EP strategies might protect against one jammer type with different efficacy. These models are evaluated with a modified confusion matrix (Section 10.3.3).

10.3.1 Regression and Normalized Root-Mean-Square Error

Regression algorithms are usually evaluated with RMSE, the square root of the average squared difference between the observed values $\hat{y}$ and the estimated values y. *Normalized RMSE (nRMSE)* allows results from different data distributions to be compared fairly. nRMSE is the RMSE divided by the standard deviation σ of the values. For v instances with a mean value of $\mu = \frac{1}{v}\left(\sum_{i=i}^{v} \hat{y}_i\right)$:

$$\text{RMSE} = \sqrt{\frac{1}{v}\sum_{i=1}^{v}\left(\hat{y}_i - y_1\right)^2} \qquad \text{nRMSE} = \frac{\text{RMSE}}{\sigma} = \frac{\sqrt{\frac{1}{v}\sum_{i}^{v}\left(\hat{y}_i - y_1\right)^2}}{\sqrt{\frac{1}{v}\sum_{i}^{v}\left(\hat{y}_i - \mu\right)^2}}$$

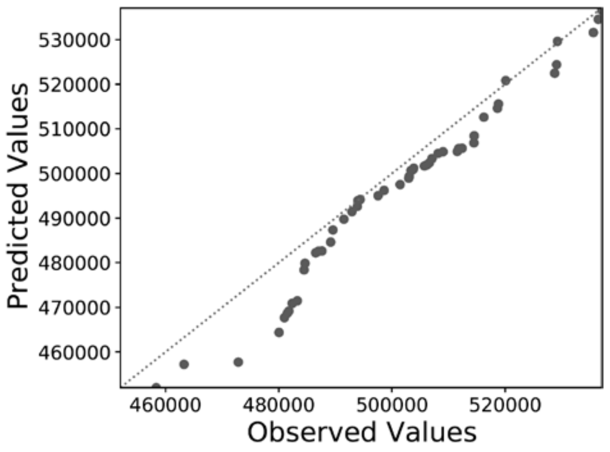
The standard deviation represents the performance of a learner that uses the mean as the prediction for all instances. Our goal is to achieve as low an nRMSE as possible; 0.0 indicates that every instance is predicted with no error, while >1.0 indicates that no "fancy" model is needed because the mean value is better. Figure 10.6 illustrates a simple example allowing us to compare the models for two metrics.

A high nRMSE tells us that the model struggled to capture the performance surface. This result can help the system engineer understand and identify systematic problems, such as missing observables, faulty sensors, unreliable metrics, and temporal latencies.

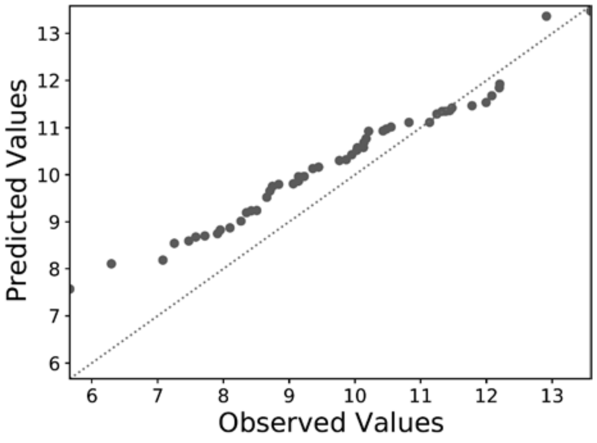
10.3.2 Classification and Confusion Matrices

Classification algorithms are usually evaluated using accuracy values computed from *confusion matrices* using the structure illustrated in Table 10.2. The rows correspond to the known *observed* classes, while the columns correspond to the classes *predicted* by the model. A true positive or a true negative indicates that the model was correct, while a false positive or a false negative indicates that the model was wrong. Table 10.3 shows some notional results for the acoustic signal classes biological and man-made; this notional classifier yields 94.5% accuracy, with a propensity to label objects as man-made. Each cell in this confusion matrix counts the number of examples of each type, for two classes.

Depending on the problem, false negatives and false positives may have different consequences. For example, if an explosives detector in airport baggage scanner identifies a nonexplosive as an explosive, then the false positive creates some additional screening time. If the scanner misses an explosive, however, then the false negative could cost lives.



(a)



(b)

Figure 10.6 To compare models that predict different ranges, we must normalize the RMSE by the standard deviation. (a) Metric m_1 ranges from 458,365 to 537,068, with $\mu = 501{,}041$ and $\sigma = 18{,}061$; RMSE is 6,328, yielding an nRMSE of 0.350. (b) Metric m_2 ranges from 5.7 to 13.6, with $\mu = 9.7$ and $\sigma = 1.7$; RMSE is 0.76, yielding an nRMSE of 0.442. (nRMSE=0.0 occurs when all predictions are identical to the true values, and lie along the diagonal grey line.)

10.3.2.1 Accuracy, Precision, Recall, and Class Imbalance

The *accuracy* of the confusion matrix is the diagonal counts divided by the total number of test instances. For a confusion matrix $\mathcal{M}$ that counts labels across x classes, $\mathcal{M} \in \mathbb{J}^{x \times x}$:

$$\text{Accuracy} = \frac{\sum_{i=1}^{x} \mathcal{M}_{i,i}}{\sum_{i=1}^{x} \sum_{j=1}^{x} \mathcal{M}_{i,j}}$$

Table 10.2
Confusion Matrix Showing How a Classification Algorithm Identifies Objects

	Predicted Positive	**Predicted Negative**
Observed Positive	True positive	False negative
Observed Negative	False positive	True negative

Table 10.3
Notional Results for the Acoustic Signal Classes Biological and Man-Made

	Bio	**Man-made**
Biological	90	10
Man-made	1	99

Accuracy can be a misleading metric for imbalanced data sets (for example, when there are many fewer man-made objects than biological objects). *Balanced accuracy* normalizes predictions by the number of examples in each class:

$$\text{Balanced Accuracy} = \frac{1}{x}\sum_{i=1}^{x}\left(\frac{\mathcal{M}_{i,i}}{\sum_{j=1}^{x}\mathcal{M}_{i,j}}\right)$$

Recall corresponds to the number of samples in a class that were correctly labeled, by computing the diagonal value against the sum of the row *i*:

$$\text{Recall}_i = \frac{\mathcal{M}_{i,i}}{\sum_{j=1}^{x}\mathcal{M}_{i,j}}$$

Precision corresponds to how tightly the model predicts a class, by computing the diagonal value against the sum of the column *j*:

$$\text{Precision}_j = \frac{\mathcal{M}_{j,j}}{\sum_{i=1}^{x}\mathcal{M}_{i,j}}$$

10.3.2.2 Multiple Classes

When there are more than two classes, we expect high match counts along the diagonal of the confusion matrix; off-diagonal results show misclassification errors. Rows and columns are usually sorted by some form of similarity, making it easier to see (and possibly ignore) misclassification errors between similar classes. For example, in a modulation classification evaluation, we would expect QAM16 and QAM64 to be adjacent, where misclassifications are unsurprising.

In the SEI example of Figure 10.7, each pair of rows/columns is two transmitters mounted on the same radio platform that share a power supply. Therefore, misclassifications within each "block" of four cells around the diagonal are "less" incorrect than misclassifications elsewhere.

10.3.2.3 Weighted Accuracy and Model Quality

When the reward for a classification differs by class, we bias the accuracy evaluation using a reward matrix, $\mathcal{R} \in \mathbb{R}^{x \times x}$, and $0 \leq \mathcal{R}_{i,j} \leq 1$. In this case, we weight the confusion matrix $\mathcal{M}$ by the rewards $\mathcal{R}$, obtaining a weighted accuracy matrix $\mathcal{Q} \in \mathbb{R}^{x \times x}$: each cell i,j in $\mathcal{M}$ is weighted by its corresponding cell i,j in $\mathcal{R}$:

$$\mathcal{Q}_{i,j} = \mathcal{M}_{i,j} \mathcal{R}_{i,j}$$

Overall quality of the model is computed as a percentage of the total (label quality divided by total number of labels):

$$\text{Quality} = \frac{\sum_{i=1}^{x} \sum_{j=1}^{x} \mathcal{Q}_{i,j}}{\sum_{i=1}^{x} \sum_{j=1}^{x} \mathcal{M}_{i,j}}$$

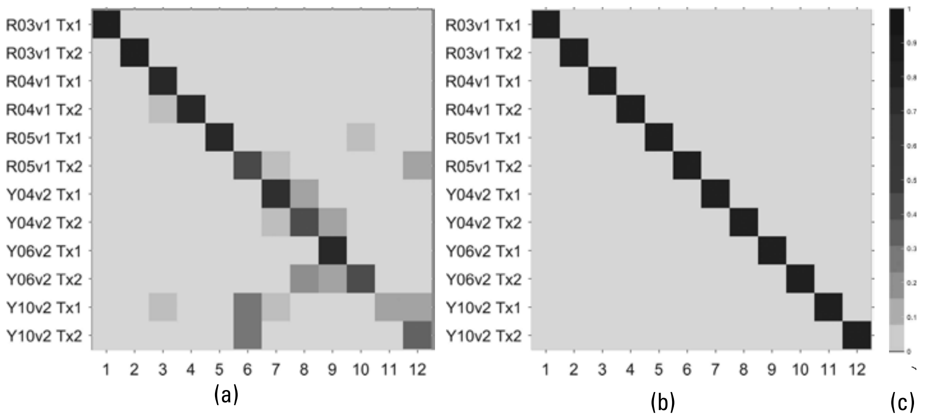


Figure 10.7 These confusion matrices represent the labels for specific emitter identification using two different classification algorithms [4]. (a) DeepNets achieved 71.9% accuracy. (b) A multistage training algorithm achieved 98.7% accuracy.

As an example, consider a notional set of rewards for six radar classes (two radars with three modes each) in Table 10.4. Diagonal cells capture the case when the model correctly labeled both the radar and its mode. It is partially correct to identify the correct radar (but the wrong mode), or the correct mode (but the wrong radar). Using these reward weightings $\mathcal{R}$ to weight the classification results $\mathcal{M}$ of Table 10.5, we obtain the quality matrix $\mathcal{Q}$ shown in Table 10.6. Overall quality is 90%.

Table 10.4
Notional Radar Reward Matrix $\mathcal{R}$

	R1M1	**R1M1**	**R1M3**	**R2M1**	**R2M2**	**R2M3**
Radar 1, Mode 1	1.0	0.5	0.5	0.3	—	—
Radar 1, Mode 2	0.5	1.0	0.5	—	0.3	—
Radar 1, Mode 3	0.5	0.5	1.0	—	—	0.3
Radar 2, Mode 1	0.3	—	—	1.0	0.5	0.5
Radar 2, Mode 2	—	0.3	—	0.5	1.0	0.5
Radar 2, Mode 3	—	—	0.3	0.5	0.5	1.0

Table 10.5
Notional Model Classifies Radars and Modes, Yielding a Confusion Matrix $\mathcal{M}$ Showing 85% Accuracy

	R1M1	**R1M1**	**R1M3**	**R2M1**	**R2M2**	**R2M3**	**Recall**
Radar 1, Mode 1	10	—	—	2	—	—	83%
Radar 1, Mode 2	1	11	—	—	—	—	92%
Radar 1, Mode 3	—	1	5	—	1	5	42%
Radar 2, Mode 1	—	—	—	11	—	—	100%
Radar 2, Mode 2	—	—	—	1	12	—	92%
Radar 2, Mode 3	—	—	—	—	—	12	100%
Precision	91%	92%	100%	79%	92%	71%	

Table 10.6
Combining the Confusion Matrix $\mathcal{M}$ of Table 10.4 and Reward Weights $\mathcal{R}$ of Table 10.5, Yielding Quality Matrix $\mathcal{Q}$ with a Quality Score of 90%

	R1M1	**R1M1**	**R1M3**	**R2M1**	**R2M2**	**R2M3**
Radar 1, Mode 1	10	—	—	0.6	—	—
Radar 1, Mode 2	0.5	11	—	—	—	1.5
Radar 1, Mode 3	—	0.5	5	—	0	—
Radar 2, Mode 1	—	—	—	11	—	—
Radar 2, Mode 2	—	—	—	0.5	12	—
Radar 2, Mode 3	—	—	—	—	—	12

10.3.3 Evaluating Strategy Performance

To evaluate the *effect* of the strategy on system performance, neither nRMSE nor confusion matrices capture an essential concept:

> *For a given RF environment, there may be multiple strategies that contribute to performance with a nonzero utility.*

A *modified confusion matrix* measures performance of strategies against environments. Instead of the confusion matrix of environments by environments, we use environments by strategies.

The SD maintains a performance table $\mathcal{P}^{m_k} \in \mathbb{J}^{a \times b}$ for each metric m_k, where $\mathbb{J}$ is the set of positive integers, $a << \overline{o_n}$ is the number of environments, and b is the number of strategies $\Pi_{\forall c} v_c$. Each cell $\mathcal{P}^{m_k}_{i,j}$ corresponds to the observed performance of metric m_k using strategy $j = s_n$ in environment i associated with o_n.[2] Section 2.1.1 describes how to compute an environment for collected observables.

Essentially, $\mathcal{P}^{m_k}$ summarizes the data from the GTDF, plus any experience collected in-mission. The GTDF is a collection of examples, $<o,s,m>$, while $\mathcal{P}^{m_k}$ is a table of size $a \times b$ for a given m_k. In theory, $\mathcal{P}^{m_k}$ could have a very large number of rows, but the SD controls the number of environments a by setting the maximum number of environment clusters (Section 2.1.1) .

10.3.3.1 Adequacy: Performance over a Scenario

Adequacy is the performance of the system over a scenario. A simplistic way to compute adequacy is to count the strategy choices for each environment and score against the best strategy for that environment, similar to computing the quality matrix Q.

This approach, however, doesn't take into account cases when changing strategies is part of the utility function (i.e., when the best strategy at time t depends on which strategy was in place at time $t - \delta$).

To compute adequacy during a mission, the SD uses the observed performance $U_n(t)$ divided by the best possible performance $\hat{U}_n(t)$:

$$\mathcal{A}_n(t) = \frac{U_n(t)}{\hat{U}_n(t)}$$

$\hat{U}_n(t)$ corresponds to the strategy that could have achieved best performance at time t. The SD computes $\hat{U}_n(t)$ from the performance tables $\mathcal{P}^{m_k}$ using the

2. It may be appropriate to maintain a list of the obtained performances and then use a probability distribution or other statistical measure to determine goodness relative to optimal.

utility function in place at time t at node n, and computes $U_n(t)$ from the performance feedback returned at time $t+\delta$. (It is not the true optimal, but rather the optimal among previously-collected ground truth.)

Adequacy for the scenario is the average over all timestamps:

$$\mathcal{A}_n = \frac{1}{T}\sum_{t=1}^{T}\mathcal{A}_n(t)$$

Figure 10.8 shows results from a scenario with six environments changing every five time units. Generally, the BBN SO chooses the best strategy for the environment, yielding an overall adequacy $\mathcal{A}_n = 0.86$. Most errors occur when the environment changes, in that the strategy in place in env_i has less utility against the next environment env_{i+1}. When the adequacy stays at 1.0 even when the environment changes, that indicates that the chosen strategy is equally useful in both environments. Adequacy can be negative (e.g., at $t = 1$), because costs outweigh benefit. Adequacy can improve over multiple timesteps (e.g., $t = [11,...,13]$) due to a ramp-up time of the strategy, or to incremental learning where the decision maker tries different strategies.

Figure 10.9 shows the impact of using incremental in-mission learning. The adaptive and cognitive systems start with a model trained on the same data. The SD exposes each system to the same sequence of novel environments. The adaptive system is not allowed to retrain and uses the learned-but-unchanging model throughout the scenario, yielding a final adequacy score of 0.28. The cognitive system retrains three times, which enables it to choose better strategies, and

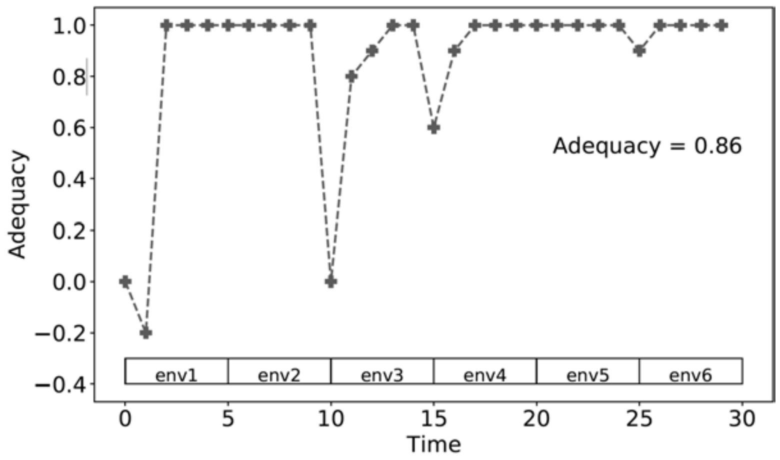
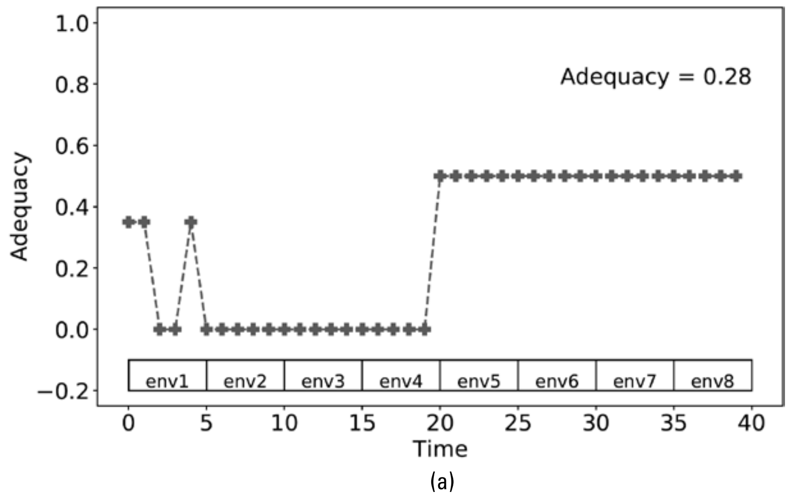


Figure 10.8 The SD evaluates adequacy at each time step using the decision-maker's strategy choice against the best-known strategy for that environment. (Example 7.1, [3].)



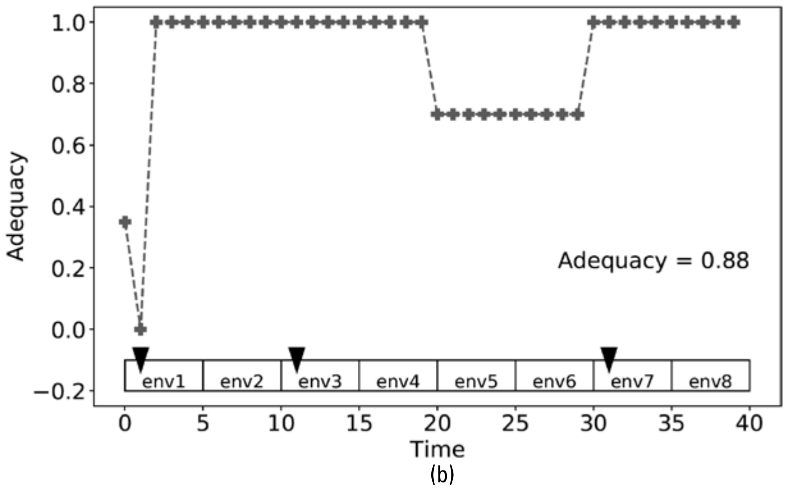


Figure 10.9 A cognitive system in (b) performs better than an adaptive system in (a) the same test scenario. Triangles indicate when the decision maker triggered learning events (Example 7.1. Redrawn from [3].)

achieves an adequacy score of 0.88. In-mission learning learns to handle the novel environments using only one example.

Each single point in Figure 10.5 represents the adequacy score for one train/test scenario, showing adequacy across all combinations of training/testing environments.

Incremental learning handles new environments with only a small loss of optimality.

Unsupervised clustering determines the environments, and real-world observed performance determines the utility. The optimal value for a given environment may change during a mission, especially if exploration exposes better candidate strategies. When running in a setting without ground truth, "optimal" is the best known so far.

10.4 Learning Assurance: Evaluating a Cognitive System

Evaluating a cognitive system has many similarities to traditional validation and verification (V&V) approaches. Tests include normal cases, edge cases, stress conditions, and adversarial situations. A *learning* system, however, requires adaptation to the V&V process. Notably, this process ensures and evaluates model robustness, particularly with respect to changes in the data. All of the steps of data management, model development, and metalearning affect the quality of the final system.

> Validation [of complex models] is a process involving measurements, computational modeling, and subject-matter expertise, for assessing how well a model represents reality for a specified quantity of interest and domain of applicability.
>
> —*National Research Council, 2012 [5]*

10.4.1 Learning Assurance Process

Learning assurance is a *quality assurance* or *quality control* process for any ML-enabled system. The learning assurance processes aim to provide the (cognitive) EW stakeholders with the same trust levels that existing EW systems boast, having undergone the traditional "V"-shaped development assurance process.

Callout 10.1 presents a list of tasks necessary for developing an effective AI system. The design loop of Figure 10.10 ensures data quality, model accuracy, and model generalizability. The design phase is iterative, ensuring that all changes are verified: changes to the data formats, choices in model structure, and adaptations to hyperparameters.

To bring these design ideas into a formal certification methodology, the European Aviation Safety Agency and Daedalean AG adapted the traditional "V"-shaped process to handle cognitive systems. The learning assurance "W" of Figure 10.11 incorporates steps for learning systems: data life-cycle management, and model training and verification [6], described as follows:

Callout 10.1 The five steps of an AI project guide and constrain development effort to encompass only necessary and sufficient actions.

In 1988, Cohen and Howe [2] presented an AI research five-stage model that describes evaluation guidelines per stage. These guidelines are provided in the form of detailed evaluation criteria and techniques, describing how to conduct an evaluation, and they are still valid today:

- *Refine the topic to a task.* Is the task significant? Is it representative of the class of tasks?
- *Design the method.* Is the method an improvement over existing approaches? What is the scope of the method? What alternative approaches exist?
- *Build a program.* How demonstrative is the program? Is it tuned for an example? Are the results predictable?
- *Design experiments.* How many examples can be demonstrated? What benchmarks should be used to compare results? How does each component impact the result?
- *Analyze the results.* What is the performance? How efficient is the algorithm? What are its limitations?

The paper goes into much more detail about each task, and how to evaluate steps.

- *Data management* identifies candidate datasets, data preparation methods, data quality requirements, and validation objectives with respect to the product/system requirements and concept of operations (ConOps).
- *Learning process management* drives the selection and validation of elements such as the training algorithm, the initialization strategy, and hyperparameters. It also considers the hardware and software frameworks and chooses the evaluation metrics. The core measurement is *accuracy.* Accuracy depends on a variety of other contextual factors, and these trade-offs should be measured (Callout 10.2).
- *Model training* trains the model and evaluates model parameters using the validation dataset.
- *Learning process verification* evaluates the trained model against the test dataset, using the measurements identified for evaluation.

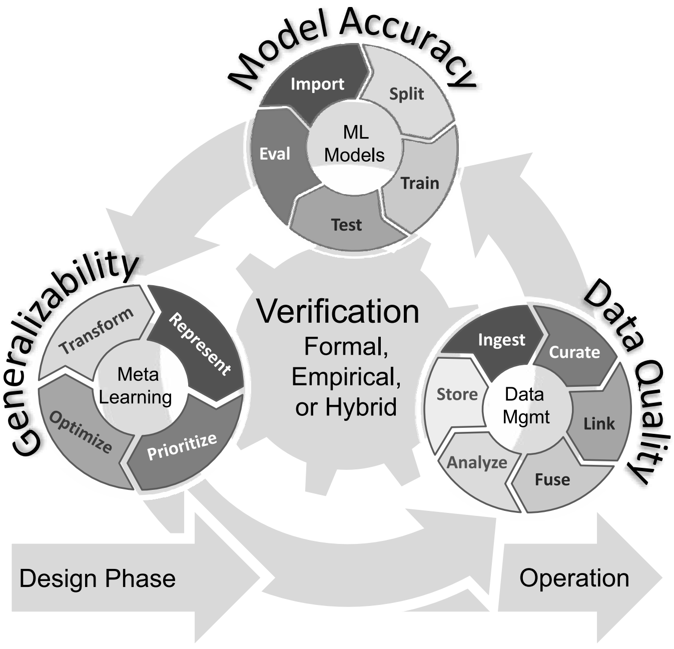


Figure 10.10 During the design phase, principles of verification apply to the iterative process of data management, model development, and metalearning. Good design verification leads to performance assurance during the operational phase.

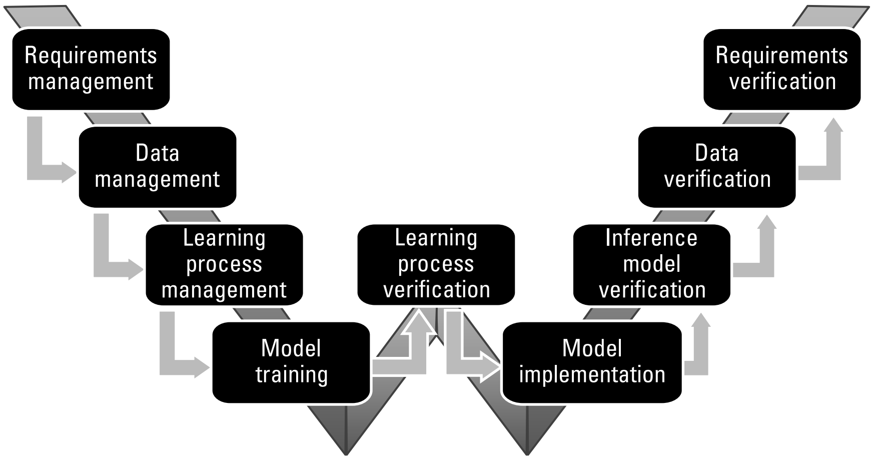

Requirements management
Data management
Learning process management
Model training
Learning process verification
Model implementation
Inference model verification
Data verification
Requirements verification

Figure 10.11 V&V for a learning system. (Source: [6]. Reprinted with permission.)

- *Model implementation* transforms the model into an executable model that can run on a target hardware. All optimizations and modifications for the embedded hard real-time environment must be validated.

Callout 10.2 Accuracy is the main system evaluation metric, but depends on a variety of other contextual factors and system requirements.

Accuracy: Successful characterization of the environment, behaviors, and causal event patterns, and high adequacy of EP/EA strategies.

Scalability: Number of emitters, number of strategies, size of training data.

Portability: Hardware and operating system context.

Complexity: Mission/threat complexity.

Data requirements: Amount, type, completeness, correctness; see also Table 8.1.

Computation effort: Time to learn a model, time to make an inference, sample efficiency.

Adaptability: Time and number of examples required to update a model to novel conditions.

Generalizability: Accuracy outside of the domain of applicability.

Robustness/stability: Brittleness and sensitivity to variation of input values.

Uncertainty/confidence: Ability to self-determine the confidence on the estimate.

Usability and explainability: Ease for a human to understand the results, both in-mission, and for forensics.

Safety: Ease of providing performance guarantees.

Transferability: Effectiveness of model in slightly different contexts (e.g., trained on synthetic data and tested on real data).

Cumulative gains: Impact of performance over time/experience, or in conjunction with additional AI/ML components.

Performance over time: System performance over time; concept drift; whether stakeholders develop genuine trust and willingness to rely on system decisions.

- *Inference model verification* ensures that the embedded model continues to meet expected requirements.
- *Data verification* ensures that data assumptions have not changed.

Multiple-version dissimilar software is a system design technique that involves producing two or more components of software that provide the same function in a way that may avoid some sources of common errors between the components [7]. Correlated to ensemble learning (Section 3.2), redundant solutions increase system resilience and improve confidence in learning approaches.

Learning systems should be verified with both empirical and formal methods (or a combination thereof) [6, 8]. Figure 10.12 briefly defines each category. Sections 10.4.2 and 10.4.3 present specific examples of formal and empirical verification methodologies, respectively.

The National Research Council describes how to assess the reliability of complex models [5]. Luckcuck et al. [9] provide a detailed overview of formal specification and verification for autonomous robotics. Jacklin et al. [10] address some of the unique V&V challenges for adaptive flight-critical control software. Lahiri and Wang [11] present various formal methods for ANN verification for safety-critical systems. Example 10.1 presents an example of an assurance architecture for an aircraft taxiing application.

From a practical perspective, useful outputs for verification methodologies include [6, 13]:

- Inputs that violate constraints (out-of-sample error);
- Amount that inputs can change before causing outputs to fail requirements;
- What outputs to expect for a given set of inputs;
- Confidence estimates that describe when a model is likely to fail.

Empirical Verification	Semiformal Verification	Formal Verification
Treat model as a white box	Combine mathematical and empirical concepts	Obtain worst case robustness bounds using formal methods
• Conduct systematic testing of trained models • Examine both intrinsic and extrinsic features/behaviors	• Identify boundary cases for the trained models • Identify confidence of predictions • Adversarial testing	• Theoretically determine when the model changes greatly under the training dataset perturbations (e.g., when ML algorithm stability is compromised)

Figure 10.12 Learning verification approaches include empirical, formal, and hybrid methods.

Example 10.1 Multiple assurance approaches prove the safety ML-based components within an aircraft taxiing application.

Cofer et al. [12] demonstrated a run-time assurance architecture for an aircraft-taxiing application. The demonstration included a safety architecture based on the ASTM F3269-17 standard for bounded behavior of complex systems, diverse run-time monitors of system safety, and formal synthesis of critical high-assurance components. The developed architecture demonstrated the ability to maintain system safety in the presence of defects in the underlying ANN-enabled components.

Cofer et al. used the following techniques to assure the system: (1) modeling system architecture using the Architecture Analysis and Design Language (AADL), (2) formally verifying system behaviors using the Assume Guarantee Reasoning Environment (AGREE), (3) using an architecture-based assurance case for showing correct implementation with the resolute language, (4) employing various run-time monitors for system safety, integrity, and availability, and (5) developing synthesis from formal specifications with proof of correctness for critical high-assurance components. Note that AGREE uses *k*-induction as the underlying algorithm for model checking.

10.4.2 Formal Verification Methods

Computational learning theory (CLT) and *statistical learning theory (SLT)* are subfields of AI that study the design and analysis of ML algorithms [14–16]. While CLT and STL both share the same theoretical framework, CLT attempts to determine what problems are "learnable," and STL focuses on improving the accuracy of existing ML algorithms.

The *Vapnik–Chervonenkis (VC) dimension* determines the complexity of a given classifier [17]. A large VC dimension indicates that the classifier is more complex and vice versa. The VC dimension can, for example, predict a probabilistic upper bound on the test error of a classifier [18].

Probably approximately correct (PAC) learning is a theoretical framework proposed in 1984 by Leslie Valiant [19]. PAC learning is used for analyzing ML generalization errors in terms of their errors on training datasets, while also providing some measures of their complexity. Its typical objective is to demonstrate a high probability (hence the "probably" part) of an algorithm achieving a low generalization error (the "approximately correct" part). PAC learning has been extended to include *PAC-Bayesian* inequalities [20–22]. McAllester's PAC-Bayesian analysis, for example, derives upper empirical bounds for Bayesian classifiers [23]. In a sense, this analysis could be considered semiformal.

Katz et al. [24, 25] develop a theoretical framework for verifying deep neural networks (DNNs). *Marabou* is based on *satisfiability modulo theories (SMT)* solvers and can answer queries about a DNN's properties by transforming these queries into constraint satisfaction problems. Marabou formally proved that DNNs avoid midair collisions. SMT methods can also detect adversarial perturbations, that is, the minimal changes that cause the network to misclassify. The method can guarantee that adversarial examples, if they exist, are found for the given region and family of manipulations [26]. SMT solvers belong to a class of automated theorem provers that can deduce the satisfiability and validity of first-order formulas, in particular logical theories. Balunovic et al. [27] show that SMT solvers are more general than satisfiability (SAT) solvers, and have been used for verification of neural networks, program synthesis, static analysis, and scheduling. SAT solvers essentially provide a generic combinatorial reasoning-and-search platform and have been used for software verification, planning, and scheduling [28].

Wang et al. [29] use *interval arithmetic* to compute rigorous bounds on the DNN outputs and symbolic interval analysis to minimize overestimations of output bounds. Amini et al. [30] compute precise and calibrated *uncertainty estimates* to estimate out-of-distribution samples and recognize when the model is likely to fail. Uncertainty can come from both data (*aleatoric* uncertainty) and from the prediction (*epistemic* uncertainty).

Maher and Orlando [31] use *ontology-based knowledge graphs* to enhance operational training of ML capabilities. The use of an ontology model, through the creation of knowledge graphs, can be a solution for dealing with bad data. This methodology allows for a formal and explicit expression of domain's concepts by adding the semantics to describe how domain data relate to each other.

Formal verification methods are attractive due to their ability to theoretically determine algorithmic stability. However, the complexity of DeepNets and other ML algorithms makes it difficult to utilize formal software verification processes. Issues include computational complexity, scalability, applicability, lack of benchmarking, and trade-off between runtime and results completeness.

10.4.3 Empirical and Semiformal Verification Methods

Empirical verification approaches must be conducted in a closed-loop manner because the environment responds to actions (Section 10.1).

During the design phase, verification testing can be performed with synthetic data, emulated data, real training data, augmented data, or any combination. Performance measurement(s) should be conducted in the field, especially during the operational phase of learning assurance. For example, if real raw I/Q data is unavailable, RF signal classifiers can be initially trained/verified on synthetic I/Q data (e.g., generated via MATLAB). Although synthetic data introduces unwanted artifacts and lack realism (e.g., I/Q imbalance or phase noise of a given OFDM system), it can also provide data diversity and capture edge cases (Section 8.3.2). Some of the general drawbacks of empirical verification methodologies can be (training) data integrity and data bias. Approaches include:

- *Ablation testing: n*-choose-*k* ablation testing (Section 10.2) demonstrates that the system can handle new environments, because it trains on *k* cases and tests all *n* known ground-truth environments, thus evaluating performance on $n - k$ novel environments.
- *Adversarial testing:* This approach assesses ML algorithms' robustness against an adversary who does the worst possible thing (e.g., [32]).
- *White-box testing:* White-box verification methods assess the behavior of a model through testing. The white-box concept determines the final output behavior and its internal parameters such as neural coverage and activation patterns [33, 34].
- *Limitation studies:* To understand how well the system will perform in unexpected conditions, test it with known edge cases, such as increased noise, incomplete data, or errors. Falsification, for example, generates edge cases. While there are no guarantees of completeness or correctness with this method, test coverage is typically expanded in efficient ways that make sense [35, 36].
- *Mistake-bound model:* A mistake-bound model addresses the question of how many mistakes an online learning algorithm will make before it learns the intended concept or target function f [37, 38]. Learning occurs in rounds, where in each round the SD provides an unlabeled example x, and the learner must predict the value $f(x)$ of the unknown target function f. The SD then provides the performance feedback, allowing the learner to retrain and update its hypothesis. The *mistake bound* of the learner is the worst-case number of mistakes it makes across all rounds.
- *Simplification:* Elboher et al. [39] propose a verification framework for simplifying neural nets by using overapproximation to reduce the size of the network. *Knowledge distillation* trains "teacher" models and then trains a simple "student" model to mimic the teacher [40–42]. Abstract interpretation approximates the behavior of the network, explicitly managing the trade-off between approximations and precisions [43].
- *Occam learning:* Occam learning is closely related to PAC learning but can produce tighter (empirical) bounds on the training dataset's complexity. The algorithm's objective is to output a *concise* representation of the received training dataset. Occam learning was named after Occam's razor, a principle stating that, given two explanations for observed data, all other things being equal, the simpler explanation is preferred [44].
- *Sensitivity analysis/tuning studies:* Metalearning (Section 5.1.3) can estimate how sensitive the models are to changes in hyperparameters, yielding estimates of stability and robustness to change. Output reachability

analysis computes the maximum sensitivity of neural networks, and then simulates the output reachable set [45].

- *Direct assessment:* In some cases, an end user can evaluate the "plausibility" of system performance. While this approach doesn't provide any "publishable" results, it builds stakeholder trust.

10.5 Conclusion

Evaluating a learning system requires thinking outside the traditional V&V methods. Traditional approaches tend to validate system performance on one (or several) fixed, known scenarios, and do not portray novel situations. The learning assurance process must validate data quality, model accuracy, and model generalizability. System requirements generally handle pure DM (optimization) approaches but have not yet caught up to the needs of learning-based AI. Data quality, data storage, model security, and validation objectives are lacking.

Due to the interactive and adversarial nature of EW, cognitive systems must be tested in a closed-loop environment. System requirements highlight which of the three axes of cognition (Figure 1.3) matter to the mission, ensuring that test scenarios and measured characteristics meet goals.

References

[1] McLeod, S., "Maslow's Hierarchy of Needs," *Simply Psychology,* 2018.

[2] Cohen, P., and A. Howe, "How Evaluation Guides AI Research: The Message Still Counts More Than the Medium," *AI Magazine,* Vol. 9, No. 4, 1988.

[3] Haigh, K. Z., et al., "Parallel Learning and Decision Making for a Smart Embedded Communications Platform," BBN Technologies, Tech. Rep. BBN-REPORT-8579, 2015.

[4] Youssef, K., et al., "Machine Learning Approach to RF Transmitter Identification," *IEEE Journal of Radio Frequency Identification,* Vol. 2, No. 4, 2018.

[5] National Research Council, *Assessing the Reliability of Complex Models,* National Academies Press, 2012.

[6] Cluzeau, J., et al., *Concepts of Design Assurance For Neural Networks (CoDANN),* European Union Aviation Safety Agency. Online: https://tinyurl.com/CoDANN-2020.

[7] *Software Considerations in Airborne Systems and Equipment Certification,* RTCA DO-178C/EUROCAE ED-12C Standard, Radio Technical Commission for Aeronautics (RTCA), 2011.

[8] Sun, X., H. Khedr, and Y. Shoukry, "Formal Verification of Neural Network Controlled Autonomous Systems," in *Hybrid Systems: Computation and Control,* 2019.

[9] Luckcuck, M., et al., "Formal Specification and Verification of Autonomous Robotic Systems: A Survey," *ACM Comput. Surv.,* Vol. 52, No. 5, 2019.

[10] Jacklin, S., et al., "Verification, Validation, and Certification Challenges for Adaptive Flight-Critical Control System Software," in *AIAA Guidance, Navigation, and Control Conference and Exhibit,* 2004.

[11] Lahiri, S. K., and C. Wang, *Computer Aided Verification,* Springer Nature, 2020, Vol. 12224.

[12] Cofer, D., et al., "Run-Time Assurance for Learning-Enabled Systems," in *NASA Formal Methods Symposium,* Springer, 2020.

[13] Liu, C., et al., *Algorithms for Verifying Deep Neural Networks,* 2019. Online: https://arxiv.org/abs/1903.06758.

[14] Bousquet, O., S. Boucheron, and G. Lugosi, "Introduction to Statistical Learning Theory," in *Summer School on Machine Learning,* Springer, 2003.

[15] Kearns, M., U. Vazirani, and U. Vazirani, *An Introduction to Computational Learning Theory,* MIT Press, 1994.

[16] Russell, S., and P. Norvig, *Artificial Intelligence: A Modern Approach,* Pearson Education, 2015.

[17] Vapnik, V., and A. Chervonenkis, "On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities," *Theory Of Probability & Its Applications,* Vol. 16, No. 2, 1971.

[18] Vapnik, V., *The Nature of Statistical Learning Theory* (Second ed.), New York: Springer-Verlag, 2000.

[19] Valiant, L., "A Theory of the Learnable," *Communications of the ACM,* Vol. 27, No. 11, 1984.

[20] Shawe-Taylor, J., and R. Williamson, "A PAC Analysis of a Bayesian Estimator," in *Computational Learning Theory,* 1997.

[21] McAllester, D., "Some PAC-Bayesian Theorems," *Machine Learning,* Vol. 37, No. 3, 1999.

[22] McAllester, D., "PAC-Bayesian Model Averaging," in *Computational Learning Theory,* 1999.

[23] Guedj, B., *A Primer on PAC-Bayesian Learning,* 2019. Online: https://arxiv.org/abs/1901.05353.

[24] Katz, G., et al., "The Marabou Framework for Verification and Analysis of Deep Neural Networks," in *Computer Aided Verification,* Springer, 2019.

[25] Katz, G., et al., "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks," in *International Conference on Computer Aided Verification,* Springer, 2017.

[26] Huang, X., et al., "Safety Verification of Deep Neural Networks," *Lecture Notes in Computer Science,* 2017.

[27] Balunovic, M., P. Bielik, and M. Vechev, "Learning to Solve SMT Formulas," in NeurIPS, 2018.

[28] Gomes, C., et al., "Satisfiability Solvers," *Foundations of Artificial Intelligence,* Vol. 3, 2008.

[29] Wang, S., et al., "Formal Security Analysis of Neural Networks Using Symbolic Intervals," in *Conference on Security Symposium,* USENIX Association, 2018.

[30] Amini, A., et al., "Deep Evidential Regression," in *NeurIPS,* 2020.

[31] Maher, M., and R. Orlando, "Solving the "Garbage In-Garbage Out" Data Issue Through Ontological Knowledge Graphs," Processus Group and Kord Technologies, Tech. Rep., 2019.

[32] Katz, G., et al., "Towards Proving the Adversarial Robustness of Deep Neural Networks," in *Formal Verification of Autonomous Vehicles,* 2017.

[33] Pei, K., et al., "Deepxplore: Automated Whitebox Testing of Deep Learning Systems," in *Symposium on Operating Systems Principles,* 2017.

[34] Tian, Y., et al., "Deeptest: Automated Testing of Deep-Neural-Network-Driven Autonomous Cars," in *International Conference on Software Engineering,* 2018.

[35] Dreossi, T., A. Donzé, and S. A. Seshia, "Compositional Falsification of Cyberphysical Systems with Machine Learning Components," in *NASA Formal Methods Symposium,* Springer, 2017.

[36] Zhang, Z., et al., "Two-Layered Falsification of Hybrid Systems Guided by Monte Carlo Tree Search," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* Vol. 37, No. 11, 2018.

[37] Buhrman, H., D. García-Soriano, and A. Matsliah, "Learning Parities in the Mistake-Bound Model," *Information processing letters,* Vol. 111, No. 1, 2010.

[38] Littlestone, N., "Learning Quickly When Irrelevant Attributes Abound: A New Linear-Threshold Algorithm," *Machine Learning,* Vol. 2, No. 4, 1988.

[39] Elboher, Y., J. Gottschlich, and G. Katz, "An Abstraction-Based Framework for Neural Network Verification," in *International Conference on Computer Aided* Verification, Springer, 2020.

[40] Hinton, G., O. Vinyals, and J. Dean, *Distilling the Knowledge in a Neural Network,* 2015. Online: https://arxiv.org/abs/1503.02531.

[41] Papernot, N., et al., "Semisupervised Knowledge Transfer for Deep Learning From Private Training Data," in *ICLR,* 2017.

[42] Mishra, A., and D. Marr, *Apprentice: Using knowledge Distillation Techniques to Improve Low-Precision Network Accuracy,* 2017. Online: https://arxiv.org/abs/1711.05852.

[43] Singh, G., et al., "Fast and Effective Robustness Certification," in *NeurIPS,* 2018.

[44] Blumer, A., et al., "Occam's Razor," *Information Processing Letters,* Vol. 24, No. 6, 1987.

[45] Xiang, W., H.-D. Tran, and T. Johnson, "Output Reachable Set Estimation and Verification for Multilayer Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems,* Vol. 29, No. 11, 2018.

11

Getting Started: First Steps

Creating a cognitive EW system is not the hurdle that many believe.

It's easy to start small and grow, as follows:

1. Choose a bite-sized task.
2. Choose an ML toolkit and prototype a model.
3. Evaluate with representative data.
4. Implement on representative hardware.

As with all things, the devil's in the details, but starting small develops (human) expertise and awareness of which details will affect the final product.

Step 1: The first step in adopting cognitive solutions in an existing system is to identify a bite-sized task that is easy to accomplish. Typically, this new component would replace, or augment, existing traditional approaches. Good candidates include:

- Learn to classify signals (Algorithm 4.1);
- EW BDA (Section 7.1.1);
- Learn the performance of actions (Section 4.2), and then choose a strategy (Algorithm 5.1). Possible metrics include BER or link stability. For BER, Figure 2.1 lists some example observables and Table 5.1 lists some example controllables. Table 11.1 lists some example observables and

Table 11.1
Example Observables and Controllables for Link Stability

Observables	Controllables
BER (moving average; hysteresis)	Implement or deactivate retransmission
Packet error rate (moving average; hysteresis)	Modify waveform modulation
Link disruptions (time of most recent, hysteresis)	Implement or deactivate network coding
Correlation of link disruptions to context (e.g., geography, EW activity)	Modify network coding parameters
	Change the physical node position

controllables for link stability. Link stability decisions depend on BER and other variables; hysteresis or other statistics may also be relevant.

Step 2: Choose one of the ML toolkits of Section 11.2.1. Implement an initial prototype with the process outlined in Algorithm 4.1. The goal is to *develop the pipeline,* not to test the accuracy. Test the pipeline on any available machine.

The main factor driving rapid prototyping is training time, which in turn is driven by amount of data and number of training iterations. Use a very small sample of data (e.g., < 100 samples); the data should have approximately similar ranges and numerical representations (preferably normalized), but doesn't have to be realistic or representative. (e.g., Table 11.2). (That said, the more that the characteristics of this initial data represent the characteristics of the expected data, the better.) Train on 10% of the data, test on 90%, and choose loose hyperparameters (e.g., increase error limits and reduce the number of training epochs).

Step 3: Once the logic flow is established, switch to a representative dataset. Start with synthetic data and move to emulated and real data wherever possible; focus on maintaining diversity (Section 8.3.2). Evaluate candidate models, and vary their hyperparameters over acceptable ranges. Use the evaluation approaches in Chapter 10 to choose an ML algorithm.

Step 4: Implement on representative hardware. SDR platforms are a good stand-in for many EW platforms: they are relatively inexpensive and provide an RF front end [1, 2] (Figure 11.1). Expect translation from prototype model to embedded code to be a significant effort. When stable, transition to the target platform.

11.1 Development Considerations

Callout 10.1 suggests a framework for expanding to a broader cognitive solution for EW. EW efforts should factor in the following items:

- *Define a scenario.* This step is too often neglected. A scenario ensures collecting the right data and choosing the right algorithms for the platform

Table 11.2
Development of the Initial Logic Flow that Do Not Need Real or Realistic Data

Class	Duty	Freq	PRF	PW
Pulse	2.8	9200	5048	0.0005546
Pulse	2.9	7430	6373	0.0004550
Pulse	2.5	9790	1396	0.0017908
Pulse	1.5	4640	4609	0.0003254
PulseDoppler	38	1900	7799	0.0048724
PulseDoppler	49	4600	5148	0.0095182
PulseDoppler	34	8590	1619	0.0210006
PulseDoppler	26	5730	5116	0.0050820
CW	100	8830	7559	0.0132292
CW	100	610	3810	0.0262467
CW	100	7240	6772	0.0147666
CW	100	1570	9280	0.0107758

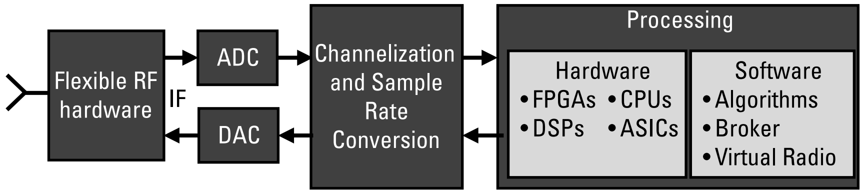


Figure 11.1 SDR platforms provide RF front end, CPU, FPGA and in some cases ASICs.

and mission. It is easy to collect the wrong data, develop "too much" ontology, choose the wrong hardware, or neglect end-user concerns such as explainability. Create a set of requirements, asking questions such as those in Table 3.2, Figure 6.2, and Chapter 8, and determine where and how the adversary plays a part.

- *Bring in data engineers from day zero.* There are too many cases where RF engineers collect good data that can't be reused later. The initial data framework (e.g., structure and ontology) should capture the initial use cases [3] and provide hooks for extensibility. Clearly define the metadata. (Section 8.1.1)
- *Identify the required data early.* Too many solutions expect to have a good dataset, but in EW there will be no good existing dataset; it will be small, and there may be no ground truth. Knowing the characteristics of the data will drive the solution choice more than any other factor (Section 11.2.3).
- *Assume that your offline data is only somewhat indicative of online data.* The solution *must* handle novel conditions. While ablation trials (Section 10.2)

establish confidence that the system will work on unknown conditions, events will occur outside *all* expected boundaries.

- *Expect that most of your time will be cleaning data.* Common problems include correcting sensor errors, deconflicting with legacy capabilities, and only receiving the "scraps" of data that the legacy system will share. (Section 8.2).
- *Prepare a closed-loop test environment.* EW systems operate against an adversary; testing using a static training set will not fulfill the goal. There is generally a lack of closed-loop test systems on which to test DM (Section 10.1).
- *Ensure that you have a scalable software architecture.* Interoperability, modularity, and scalability are hard to retroengineer. Expect conflicts with legacy systems that will need to be resolved (Section 9.1).
- *Ensure system security from the start.* Adding security infrastructure later will not achieve objectives. Protect the data and the model (Section 8.3.5).
- *Prototype and evaluate a variety of candidate solutions.* Choose a tool that allows rapid development and comparison of algorithms (Section 11.2.1). When choosing a final solution, consider not only the accuracy, but other requirements (Section 3.6), including data availability, training time, inference time, memory usage, explainability and security.
- *Expect to manually translate the prototype code for the embedded platform.* No existing vendor tool chain is sufficient for deployment or memory and timing management on embedded hard real-time platforms (example in Section 9.2).

5G's use cases and AI/ML solutions have many similarities to EW systems, as outlined in Callout 11.1 and Section 4.3.2. Tracking and adopting the work for 5G holds promise for new capabilities in EW. NATO's Cognitive Radar working group released a cognitive radar report that also compiles good examples [17].

11.2 Tools and Data

Existing ML toolkits, datasets, and simulation frameworks can accelerate the prototype and development process.

11.2.1 ML Toolkits

Popular ML libraries include scikit-learn [18], TensorFlow [19], MATLAB Machine Learning Toolbox [20], R [21], and WEKA [22]. Many of these kits have

Callout 11.1 5G use cases and solutions have direct application to EW.

5G is the fifth generation of commercial cellular networks. Developed by the 3rd Generation Partnership Project (3GPP) standardization body, it is a new global wireless standard that could be transformative as it promises significantly higher data rates (multi-gigabits-per-second peak data speeds), lower latency (in milliseconds), ubiquitous connectivity, and higher reliability than 3G and 4G technologies. 3GPP defines three primary 5G new radio (NR) use cases:

- *Enhanced mobile broadband (eMBB):* Large amounts of data will be transmitted at much higher throughputs compared to 3G and 4G. eMBB will address bandwidth-hungry applications, such as massive video streaming and virtual/augmented reality (VR/AR) [4].
- *Ultra-reliable and low-latency communications (URLLC):* Also known as mission-critical communications, URLLC will provide a stable network and the lowest possible latency (in milliseconds) to initiate connectivity (e.g., tactile Internet, autonomous vehicles (cars and drones) and the low latency of 1 millisecond or less for collision avoidance).
- *Massive machine-type communications (mMTC):* Sets the foundation for the Internet of Things (IoT). mMTC will allow machines (up to one million devices per 1 square km) to communicate with one another while requiring only a minimal human involvement (e.g., industrial applications, metering, or massive sensor networks).

Moreover, new 5G features, such as the dynamic air interface, network function virtualization, and network slicing, introduce additional system design complexity and optimization requirements to address challenges related to network operation and maintenance. Consequently, ML has recently come back into focus in the field of communications for its potential to address these challenges that cannot be resolved with traditional methodologies. For example, 3GPP and International Telecommunications Union (ITU) have both proposed 5G research projects involving various AI/ML techniques. You et al. [5] discuss four problems in 5G whose solutions have direct application to EW:

1. *Network resource allocation:* The 5G NR orthogonal frequency-division multiplexing (OFDM) resource block (RB) allocation is much more complex than that of 4G long-term evolution (LTE) because of the required support for the three aforementioned use cases. RL performs RB allocation [5] and 5G network slicing [6].

Yao et al. [5] discuss how to achieve energy-efficient beamforming in a massive MIMO system by finding the beamforming matrix, which introduces minimum power amplifier nonlinearities within a large solution space. The RNN recursively learns about the nonlinearities of the PAs and finds suitable neuron weights that satisfy two requirements: (1) zero forcing beamforming, meaning minimal multiuser interference and (2) minimum overall nonlinear distortion. The RNN models the nonlinearity of the power amplifier array, which then optimized to minimal transmitted power while providing a zero-forcing solution. The RNN then notifies the 5G system on how to set its zero-forcing beamforming weights [4].

Other ML techniques incorporate context awareness for resource allocation [7], game theory to optimize power control for users from multiple cells that are assigned the same RB [8], and managing and orchestrating cellular network resources [9].

2. *Self-organizing networks (SONs):* SON is a new way of network management that provides intelligence in the operation and maintenance of the network. 3GPP introduced SON as a key component of 4G LTE networks. 3GPP divides SON solutions into three categories: self-configuration, self-optimization, and self-healing. With 5G, SON functions will have to improve due to network ultra-densification and dynamic resource allocation as well as overall increased network complexity.

   ML techniques such as ANN and genetic algorithms solve various SON functions such as new cell and spectrum deployment, automatic base station configuration, coverage and capacity optimization, and cell-outage detection and compensation [5, 10–12].

   Gomez et al. [13] developed a root-cause analysis system that combines supervised and unsupervised learning techniques in three steps: (1) unsupervised SON training, (2) unsupervised clustering, and (3) labeling by experts.

3. *Uniform 5G baseband acceleration:* The 5G baseband signal processing comprises a series of signal-processing blocks, including massive MIMO detection and polar codes for channel decoding. The increased number of baseband blocks results in much more complex hardware designs and implementations. To speed up baseband signal processing, a uniform accelerator can be designed with the factor-graphs-based belief propagation algorithms (applied to all the blocks), aided by DeepNets (e.g., [14, 15]).

4. *Optimization of end-to-end physical layer communication:* O'Shea and Hoydis [16] propose an autoencoder-based end-to-end PHY optimization method. By interpreting a communications system as an autoencoder, the authors propose a novel approach of thinking about communications-system design as an end-to-end reconstruction task that seeks to jointly optimize transmitter and receiver components in a single process.

The demonstrated solutions to the 5G use cases map to problems in the communications, radar, and cognitive EW systems across many operational domains.

good online documentation. The book *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow* [23] is an excellent foundation for the Python environment. These libraries are effective for prototyping but will need significant modification for embedded hard real-time operation [24].

11.2.2 ML Datasets

Kaggle [25], IEEE [26], and Google Dataset [27] provide links to many public datasets. Good, comprehensive RF-specific datasets are desperately needed [28]. Even if the data *exists,* few of the datasets are tagged well enough for reuse (Section 8.1.1). Starting points include snowfall properties of radars [29] and USRP fingerprinting [30]. Flowers [31] provides code to generate synthetic data for fingerprinting.

11.2.3 Radio Frequency Data-Generation Tools

Given the complexity of the RF environment, exacerbated by the adversarial nature of EW, synthetic and emulated data-generation capabilities are key, both for basic data generation (Section 8.3.2) and interactive simulation/emulation (Section 10.1).

> In contrast to frequent and dedicated Chinese or Russian experimentation, [US] DoD does not pursue extensive EW or EMS experimentation due to concerns about operational security and access to ranges and other appropriate instrumented facilities... An increased reliance on virtual and constructive systems for EW and EMSO training could improve DoD's ability to experiment, as could a new approach to R&D that combines technical and operational innovation.
>
> —*Center for Strategic and Budgetary Assessments, 2019 [32]*

MATLAB's RF Toolbox [33] is an easy-to-access signal generator. Flowers [31] provides code to generate synthetic data for fingerprinting.

A variety of testbeds have been developed that support interactive design and evaluation of cognitive EW systems, e.g., [2, 34–36]. Many commercial signal generators exist that can be integrated with the test engine [37]. CEESIM is a plug-in to MATLAB [38]. A small testbed using Raspberry Pi [39] or USRP [40] radios supports inexpensive over-the-air signal collection and testing for initial prototypes, to develop and test the decision logic.

RF simulation and emulation tools exist that support high-fidelity development and testing of EW systems, including RFNest [41], NEWEG [42], RFView [43], and RES [44].

11.3 Conclusion

Creating a cognitive EW system is conceptually easy to grasp. Taking small incremental steps to incorporate cognitive concepts in an existing system will ensure the success of the final product. Don't aim for a fully cognitive system (Figure 1.3) in a single step; choose the most brittle piece that needs to be replaced with a more robust empirical or heuristic method. Ask yourself:

- Does your system need better situation assessment? Deeper understanding of the RF environment, the anomalies, and the intent of the emitters?
- Does your system need better DM? Something that can adapt to changing conditions and surprises?
- Does your system need to learn from its mistakes?

Each small step is a step in the right direction.

> We don't need to wait for a 'boil the ocean' 'big bang' 'all in' new network of the future to get software defined networking (#SDN) into tactical systems. We can layer it on top of what we have today.
>
> —*Tim Grayson (DARPA), 2020, referring to #DyNAMO [45]*

References

[1] Gannapathy, V., et al., "A Review on Various Types Of Software Defined Radios (SDRs) in Radio Communication," *International Journal of Research in Engineering and Technology,* Vol. 03, 2014.

[2] Christiansen, J., G. Smith, and K. Olsen, "USRP Based Cognitive Radar Testbed," in *IEEE Radar Conference,* 2017.

[3] Haigh, K. Z., et al., "Rethinking Networking Architectures for Cognitive Control," in *Microsoft Research Cognitive Wireless Networking* Summit, 2008.

[4] Yao, M., et al., "Artificial Intelligence Defined 5G Radio Access Networks," *IEEE Communications Magazine,* Vol. 57, No. 3, 2019.

[5] You, X., et al., "AI for 5G: Research Directions and Paradigms," *Science China Information Sciences,* Vol. 62, No. 2, 2019.

[6] Li, R., et al., "Deep Reinforcement Learning for Resource Management in Network Slicing," *IEEE Access,* Vol. 6, 2018.

[7] Bogale, T. E., X. Wang, and L. Le, *Machine Intelligence Techniques for Next-Generation Context-Aware Wireless Networks,* 2018. Online: https://arxiv.org/abs/1801.04223.

[8] Wang, J., et al., "Distributed Optimization of Hierarchical Small Cell Networks: A GNEP Framework," *IEEE Journal on Selected Areas in* Communications, Vol. 35, No. 2, 2017.

[9] Li, R., et al., "Intelligent 5G: When Cellular Networks Meet Artificial Intelligence," *IEEE Wireless Communications,* Vol. 24, No. 5, 2017.

[10] Wang, X., X. Li, and V. Leung, "Artificial Intelligence-Based Techniques for Emerging Heterogeneous Network: State of the Arts, Opportunities, and Challenges," *IEEE Access,* Vol. 3, 2015.

[11] Klaine, P., et al., "A Survey of Machine Learning Techniques Applied to Self-Organizing Cellular Networks," *IEEE Communications Surveys Tutorials,* vol. 19, no. 4, 2017.

[12] Pérez-Romero, J., et al., "Knowledge-Based 5G Radio Access Network Planning and Optimization," in *International Symposium on Wireless Communication Systems,* 2016.

[13] Gómez-Andrades, A., et al., "Automatic Root Cause Analysis for LTE Networks Based on Unsupervised Techniques," *IEEE Transactions on Vehicular Technology,* Vol. 65, No. 4, 2016.

[14] Tan, X., et al., *Improving Massive MIMO Belief Propagation Detector with Deep Neural* Network, 2018. Online: https://arxiv.org/abs/1804.01002.

[15] Liang, F., C. Shen, and F. Wu, "An Iterative BP-CNN Architecture for Channel Decoding," *IEEE Journal of Selected Topics in Signal Processing,* Vol. 12, No. 1, 2018.

[16] O'Shea, T., and J. Hoydis, "An Introduction to Deep Learning for the Physical Layer," *IEEE Transactions on Cognitive Communications and Networking,* Vol. 3, No. 4, 2017.

[17] Task Group SET-227, "Cognitive Radar," NATO Science and Technology, Tech. Rep. TR-SET-227, 2020.

[18] *Scikit-learn: Machine Learning in Python,* Accessed: 2020-03-22. Online: https://scikit-learn.org/stable/.

[19] Abadi, M., et al., *TensorFlow: Large-scale Machine Learning on Heterogeneous Systems,* 2015. Online: https: //www.tensorflow.org/.

[20] MathWorks, *Statistics and Machine Learning Toolbox.* Accessed: 2020-03-22. Online: https://www.mathworks.com/products/statistics. html.

[21] R Core Team, *R: A Language and Environment for Statistical Computing,* R Foundation for Statistical Computing, Vienna, Austria, 2013.

[22] University of Waikato, NZ, *WEKA: The Workbench for Machine Learning*. Accessed: 2020-04-12. Online: https://www.cs.waikato.ac.nz/ml/weka/.

[23] Géron, A., *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow,* O'Reilly, 2019.

[24] Haigh, K. Z., et al., "Machine Learning for Embedded Systems: A Case Study," BBN Technologies, Tech. Rep. BBN-REPORT-8571, 2015.

[25] *Kaggle*. Accessed: 2020-12-08, 2020. Online: https://www.kaggle.com/.

[26] *IEEE dataport*. Accessed: 2020-12-08, 2020. Online: https://ieeedataport.org/.

[27] *Google dataset*. Accessed: 2020-12-08, 2020. Online: https://datasetsearch.research.google.com/.

[28] Hall, T., *et al*., "Reference Datasets for Training and Evaluating RF Signal Detection and Classification Models," in *IEEE Globecom Workshops,* 2019.

[29] Illinois Data Bank, *Dataset for: 'A Dual-Frequency Radar Retrieval of Snowfall Properties Using a Neural Network,'* Accessed: 2020-12-08, 2020. doi: 10.13012/B2IDB-0791318_V1.

[30] Sankhe, K., et al., "ORACLE: Optimized Radio Classification Through Convolutional Neural Networks," in *INFOCOM*, Dataset available at https://genesys-lab.org/oracle, 2019.

[31] Flowers, B., *Radio Frequency Machine Learning (RFML) in PyTorch*. Accessed: 2020-12-08, 2020. Online: https://github.com/brysef/rfml.

[32] Clark, B., W. McNamara, and T. Walton, "Winning the Invisible War: Gaining an Enduring U.S. advantage in the Electromagnetic Spectrum," Center for Strategic and Budgetary Assessments, Tech. Rep., 2019.

[33] MathWorks, *RF Toolbox*. Accessed: 2020-12-13. Online: https://www.mathworks.com/products/rftoolbox.html.

[34] Reddy, R., et al., "Simulation Architecture for Network Centric Sensors and Electronic Warfare Engagements," in *Interservice/Industry Training, Simulation, and Education Conference,* Software available as a MATLAB module: https://tinyurl.com/csir-sewes, 2018.

[35] Oechslin, R., et al., "Cognitive Radar Experiments with CODIR," in *International Conference on Radar Systems,* 2017.

[36] Smith, G., et al., "Experiments with Cognitive Radar," in *Computational Advances in Multi-Sensor Adaptive Processing,* 2015.

[37] J.-J. DeLisle, *Product Trends: Signal Generators Meet the Latest Standards Head-On*. Accessed 2020-12-13, 2014. Online: https://tinyurl.com/signal-generators.

[38] Northrop Grumman, *Combat Electromagnetic Environment Simulator (CEESIM)*. Accessed 2020-12-13, 2020. Online: https://tinyurl.com/ems-ceesim.

[39] Raspberry Pi Foundation. Accessed 2020-12-13, 2020. Online: https://www.raspberrypi.org/.

[40] Ettus Research. (2020). "USRP Software Defined Radio Device," Online: https://www.ettus.com/.

[41] Intelligent Automation, Inc, *RFnest*. Accessed 2020-12-13, 2020. Online: https://www.i-a-i.com/product/rfnest/.

[42] Naval Air Warfare Center Training Systems Division, *Next-Generation Electronic Warfare Environment Generator (NEWEG).* Accessed 2020-12-13, 2020. Online: https://tinyurl.com/navair-neweg.

[43] Information Systems Laboratories, *RFView: High-fidelity RF signals and system modeling.* Accessed 2020-12-13, 2020. Online: https://rfview.islinc.com.

[44] Mercury Systems, *Radar Environment Simulators.* Accessed 2020-12-13, 2020. Online: https://tinyurl.com/mrcy-res.

[45] Grayson, T., *LinkedIn Post: #DyNAMO.* Accessed 2020-12-19. Online: https://tinyurl.com/grayson-dynamo.

Acronyms

ACK	Acknowledge
ACO	Ant Colony Optimization
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
ASIC	Application-specific Integrated Circuit
BDA	Battle Damage Assessment
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CR	Cognitive Radio
CRN	Cognitive Radio Network
CTS	Clear-to-Send
DM	Decision Making
DoD	Department of Defense (USA)
DSA	Dynamic Spectrum Access
EA	Electronic Attack
EBM	Electronic Battle Management
ECM	Electronic Countermeasures
EMS	Electromagnetic Spectrum

EMSO	Electromagnetic Spectrum Operations
EP	Electronic Protect
ES	Electronic Support
EW	Electronic Warfare
EW BDA	Electronic Warfare Battle Damage Assessment
EWO	Electronic Warfare Officer
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
GAs	Genetic Algorithms
GMM	Gaussian Mixture Model
GPU	Graphics Processing Unit
I/Q	In-phase and Quadrature
IP stack	Internet Protocol 7-layer stack
ISR	Intelligence, Reconnaissance, and Surveillance
kNN	k-Nearest Neighbor
MAC	Medium Access Layer in the IP stack
MANET	Mobile Ad-hoc Network
MDP	Markov Decision Process
ML	Machine Learning
NLP	Natural Language Processing
PHY	Physical layer in the IP stack
POI	Probability of Intercept
POMDP	Partially-observable MDP
Pd	Probability of detection
Pfa	Probability of false alarm
QoS	Quality of Service
RF	Radio Frequency
RL	Reinforcement Learning
RNN	Recurrent Neural Network

RTS	Request-to-Send
SA	Situation Assessment
SAR	Synthetic Aperture Radar
SD	Scenario Driver
SO	Strategy Optimizer
SDR	Software Defined Radio
SEI	Specific Emitter Identification
SVM	Support Vector Machines
SWaP	Size, Weight, and Power

About the Authors

Dr. Karen Zita Haigh, the fellow chief technologist (AI) at Mercury Systems, is an evangelist for AI and ML for embodied embedded systems. Formerly at Honeywell, BBN, and L3, Dr. Haigh has worked in a wide variety of complex systems, including cognitive RF systems, smart homes, cyber security, jet engines, oil refineries, and space systems. Dr. Haigh is a pioneer in three different fields that are now common across the globe. Her work in these fields is described as follows:

- Autonomous vehicles (closed-loop planning and ML on autonomous robots): Dr. Haigh's Ph.D. work at Carnegie Mellon University was the first to put AI-style symbolic planning onto a real-hardware robot and then do ML to update the planning models (i.e., it was the first full-cycle closed-loop of an autonomous robot doing both planning and learning). This capability is crucial to all autonomous-vehicle work being conducted globally today.
- Smart homes for elder care (passive behavior monitoring to assist the elderly and help them stay at home): Dr. Haigh was the principal investigator for Honeywell's Independent LifeStyle Assistant™. ILSA was an intelligent, adaptive home-automation system with sophisticated SA and DM capabilities that reasoned over a diverse set of sensors, medical devices and "smart" appliances to enable elderly and infirm users to live and function safely at home. ILSA. was a multiagent system that incorporated a unified sensing model, probabilistically derived situation awareness, intent recognition, hierarchical task network-response planning, real-time action-selection control, ML, and human factors. ILSA. was the first system of its nature in a field currently heavily researched and led to Honeywell making a strategic acquisition in the area.

- Cognitive RF systems (ML to control complex multiobjective communications systems): Dr. Haigh has worked in cognitive RF systems for over 15 years (Example 7.1). She designed the cognitive controller for Adaptive Dynamic Radio Open-source Intelligent Team (ADROIT), funded by DARPA. ADROIT is the first known real-world system (not simulation) to use ML to dynamically control a MANET. Haigh designed the cognitive engine for DARPA's CommEx program, which optimized communications-network performance in the presence of previously unknown interference and jamming conditions. CommEx was the first demonstration of real-time in-mission learning for embedded electronic protect.

This is Dr. Haigh's third book, after *The Dinner Co-op Recipe Collection* published online in 1997, and *Scripting Your World: The Official Guide to Scripting in Second Life* with Dana Moore and Michael Thome, published in 2008 by Wiley.

Julia Andrusenko is a senior communications engineer at the Johns Hopkins University APL and is the chief engineer of the Tactical Communications Systems group. Andrusenko has more than 19 years of experience in communications theory, wireless networking, satellite communications, RF propagation prediction, communications-systems vulnerability, computer simulation of communications systems, evolutionary computation, genetic algorithms/programming, MIMO, and millimeter-wave technologies. She also has substantial experience developing EW methodologies for various advanced commercial communications systems and military data links. Andrusenko is a published author of many technical papers and coauthored a book titled *Wireless Internetworking: Understanding Internetworking Challenges* through Wiley/IEEE Press. Ms. Andrusenko received her bachelor's and master's degrees in electrical engineering from Drexel University, Philadelphia. She is a member of the IEEE Communications Society and a voting member of the IEEE 1900.5 Working Group on Policy Language and Architectures for Managing Cognitive Radio for Dynamic Spectrum Access Applications.

Index