Communicating Neural Network Knowledge between Agents in a Simulated Aerial Reconnaissance System

Stephen Quirolgico and Kip Canfield
Department of Information Systems
University of Maryland, Baltimore County
Baltimore, MD 21250
{squiro1, canfield}@research.umbc.edu

Timothy Finin
Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County
Baltimore, MD 21250
finin@cs.umbc.edu

James A. Smith
Laboratory for Terrestrial Physics
NASA Goddard Space Flight Center
Greenbelt, MD 20771
jasmith@hemlock.gsfc.nasa.gov

Abstract

In order to maintain their performance in a dynamic environment, agents may be required to modify their learning behavior during run-time. If an agent utilizes a rule-based system for learning, new rules may be easily communicated to the agent in order to modify the way in which it learns. However, if an agent utilizes a connectionist-based system for learning, the way in which the agent learns typically remains static. This is due, in part, to a lack of research in communicating subsymbolic information between agents. In this paper, we present a framework for communicating neural network knowledge between agents in order to modify an agent's learning and pattern classification behavior. This framework is applied to a simulated aerial reconnaissance system in order to show how the communication of neural network knowledge can help maintain the performance of agents tasked with recognizing images of mobile military objects.

1. Introduction

In an agent-based system, agents are often required to learn in order to improve their individual performance, precision, efficiency and scope of solvable problems [15]. In order for an agent to learn, it must implement mechanisms that define how and what it learns. Such mechanisms typically come in the form of a symbolic (i.e. rule-based) or subsymbolic (e.g. connectionist-based) system. If an agent exists in a dynamic environment, or is tasked with learning new concepts, it may be necessary for the agent to modify how and what it learns in order to maintain its peformance. In an agent-based system where agents implement rule-based systems for learning, new rules may be easily communicated to agents in order to modify the way in which they learn. However, if an agent implements a connectionist-based system for learning, the way in which the agent learns typically remains static during the life of the agent. This is due, in part, to the lack of research in communicating subsymbolic information between agents. As a result, agents may be unable to adapt their learning or classification behavior to changes in the environment (especially if such changes are highly dynamic), and thus experience a degradation in performance. By allowing subsymbolic information to be communicated between agents, however, an agent may utilize new neural networks dynamically in order to potentially maintain or increase its learning or pattern classification performance.

In this paper, we present a framework for communi-

cating neural network knowledge between agents. We refer to this framework as the *Connectionist Model Transfer* (CMT) framework. The CMT framework is a general framework for communicating subsymbolic information between agents in a portable and efficient fashion, and is intended for applications where agents exist in a dynamic environment and implement neural networks for learning and pattern classification. One potential application of the CMT framework may involve intelligent interface agents tasked with learning user preferences in order to tailor their environments to specific users. Such agents may be associated with a variety of environments including operating systems, web pages, *etc*. As users move from environment to environment, models of their preferences (in the form of neural networks) may be communicated to agents so that they may tailor their respective environments accordingly. In this scenario, a received network may also be used by an agent to continue learning about a user's preferences in the context of the current environment in order to refine the profile of the user. The CMT framework may also be applied to a distributed intrusion detection system where agents are tasked with detecting anomalous system behavior that may indicate a potential computer-related attack. Here, a set of agents residing on remote hosts implement neural networks for learning normal system behavior in order to more accurately detect anomalous behavior. These networks may then be communicated between agents in order to facilitate intrusion detection among a set of hosts. Like the previous example, a received network may also be used by an agent to continue learning about system behavior in the context of the current environment in order to refine its notion of normal system behavior.

By applying the CMT framework, agents may communicate knowledge of trained neural networks and immediately instantiate this knowledge in a "Plug-and-Play" fashion — allowing them to dynamically modify their learning or pattern classification behavior in real-time. The CMT framework may also be used to facilitate *network transfer*. Network transfer refers to the reuse of neural network parameter values in order to improve the training of new neural networks [1, 17, 19]. Research in network transfer has shown that the use of parameters from existing networks can accelerate the training, and improve the accuracy, of new neural networks [16]. Finally, the CMT framework may be used to allow for the implementation of distributed neural networks by providing a means by which results of modular networks may be communicated as input into remote networks.

We begin this paper by presenting an overview of the relationship between neural networks and agent learning. The motivation for this overview is to clarify the distinction between agent learning and neural network learning, and to help establish the need for communicating neural network knowledge between agents. We then follow with a description of the general CMT framework. This will involve a discussion of issues related to the modeling of neural network knowledge, the functional requirements of agents, and the protocols required to communicate neural network knowledge between agents. We then describe the application of the CMT framework to a simulated aerial reconnaissance system. Using this system, we demonstrate how the communication of neural network knowledge can help to maintain the performance of agents in a dynamic environment.

2. Neural Networks and Agent Learning

In a number of agent-based systems, neural networks are used to implement the mechanisms for carrying out a variety of agent learning methods. Such methods, including *rote learning* and *learning by discovery*, are typically distinguished by the amount of learning effort required by an agent [20]. Rote learning refers to the immediate and direct implantation of knowledge and skills without requiring further inferencing or transformation by the learner (i.e. agent) [14, 20]. This type of learning requires the least amount of effort by an agent, but typically requires the highest levels of communication in order to acquire knowledge and skills from other agents. In contrast, learning by discovery (or *isolated learning*) is concerned with having agents learn by themselves [9, 20]. In this form of learning, agents acquire new knowledge and skills by making observations and conducting experiments. Thus, learning by discovery requires the most amount of learning effort by an agent and is not typically facilitated through communication with other agents.

A discussion of agent learning in the context of neural networks requires discussion of the relationship between the learning of an agent and the learning of its associated neural network. The life cycle of a neural network is comprised of two phases: a training phase and an execution (or *recall*) phase. In general, neural network learning is associated with the adjustment of a network's parameters (e.g. weights) during its training phase. It is through the adjustment of these parameters by which a neural network learns. Once a network has been trained, it represents "learned" knowledge and may be applied (i.e. executed) in some domain. The application of a neural network represents the recall phase of a network.

In many cases, a trained neural network is used to implement systems that simply classify tuples of input data. Such systems are often referred to as *classifier systems* [7, 10, 12]. In a classifier system, the parameters of a network remain constant which prevent it from continuing to learn during execution. Neural networks may also be used, however, to implement systems that continue to learn, adapt, and strengthen their classification capabilities during execution by analyzing feedback from the environment and adjusting

their network parameters accordingly. Such systems are often referred to as *reinforcement learning systems* [8, 13]. Note that connectionist-based classifier and learning systems embody a potential ability to *generalize*. Generalization refers to the ability of a network to derive appropriate results given incomplete, noisy, or previously unseen data without modification to its internal parameter values. Thus, generalization is not a form of learning, but rather a result of learning. Through generalization, classifier and learning systems possess the ability to recognize new patterns, but only if these patterns are similar to those that the network has learned to recognize.

In this paper, we define agent learning as a reflection of the knowledge and skills acquired through (1) the instantiation and execution of a new network or (2) the continued learning of an embodied network. When an agent instantiates and executes a new network, it learns how to classify (or learn) new patterns immediately (i.e. through rote learning). If this network is used as a classifier system, its parameters will remain constant and will not continue to learn during execution. Thus, its embodying agent may continue to learn if and only if it instantiates a new network (or incorporates another learning mechanism). However, if the network is used as a learning system, it may continue to learn through the adjustment of its parameters, and its embodying agent will continue to exhibit new learning each time the parameters of the network are adjusted (or a new network is instantiated). In this paper, we simplify discussion by assuming that all agents implement a single network and that the learning or classification performance of an agent is highly dependent upon (and directly correlated with) the learning or classification performance of its associated network.

Currently, there are limitations of what a neural network can learn during its recall phase. In most cases, neural networks that learn during execution typically do so in order to refine their notion of concepts they already know. However, if an agent exists in a highly dynamic environment or is tasked with learning new concepts, a network embodied by the agent may be unable to sufficiently adapt in order to maintain its performance. This problem is even more severe for agents tasked with recognizing (significantly) new patterns using neural networks that do not learn during execution (i.e. classifier systems). In order to facilitate performance in such cases, it may be necessary to override the learning and pattern classification mechanisms of an agent with new models. In order to do this, however, a means of communicating such models between agents must exist.

3. General Framework

The idea of communicating neural network knowledge between agents raises a number of issues. One issue is concerned with the development of appropriate models for representing neural network knowledge. Another issue concerns the identification and development of agent services that are required in order to make communicated neural network knowledge useful and practical. Yet another issue concerns the identification of appropriate protocols for communicating neural network knowledge between agents. The CMT framework attempts to address these issues by (1) providing a model specification for representing neural network knowledge, (2) providing a specification of agent services for managing neural network knowledge within a multi-agent system, and (3) identifying a protocol for communicating neural network knowledge within the context of an agent communication language.

3.1. Representing Neural Network Knowledge

Before neural networks can be communicated between agents in a portable fashion, a model of their representation must be defined. Although a number of general specifications have been proposed for modeling the architecture of neural networks [3, 4], these models lack a representation that provides meta-knowledge about a network. Such knowledge is useful in providing information about a neural network for use in a multi-agent system. In this paper, we present a model for representing neural network knowledge — that is, a network's architecture and related meta-knowledge. We refer to this model as a *Connectionist Model* (CM). The general ontology of a CM is shown in figure 1.

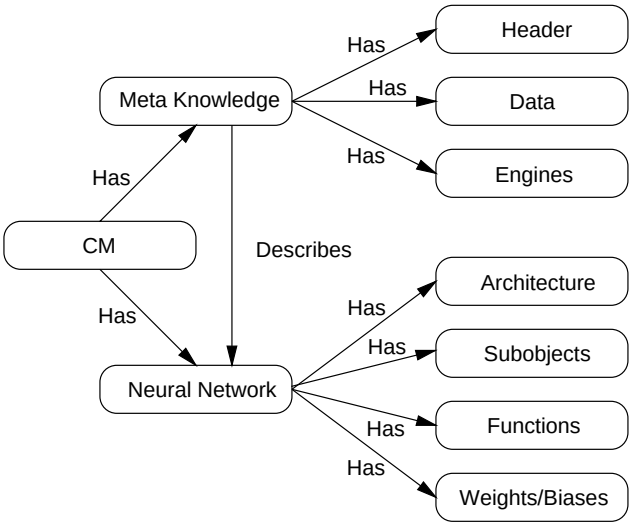


Figure 1. Connectionist model ontology.

The meta-knowledge ontology of a CM is comprised of

ontologies for specifying

- *header information*: This describes general information about the neural network including its
  - name, version, date, type, and purpose
  - learning and pattern classification capabilities
  - domain, environmental, and contextual constraints
  - author, institution, and contact information.
- *data information*: This describes information about the network's input and output data including
  - information surrounding appropriate sources of input and sensor data
  - the semantics of possible network output values
  - information about available input data preprocessors
  - actual data values used in the training and testing of the network
  - mappings of input vectors to output vectors (in the form of a rule-base) for explanation-based reasoning or validation of the network's results.
- *engine information*: This describes information related to the instantiation and execution of the network including
  - information related to translators for mapping object representations of CMs to and from various knowledge representation formats as well as translators for mapping CMs to and from various neural network development and execution environments
  - information related to processing engines available from other sources that may be retrieved and utilized in the agent's local environment. These processing engines may exist in many forms including objects, libraries, *etc*.

The neural network portion of a CM is based on the MATLAB network object specification [3] and is comprised of ontologies for specifying

- *architecture information*: This describes the number of network subobjects and how they are connected.
- *subobject structure information*: This describes properties of array structures that define the network's inputs, layers, outputs, targets, biases, and weights.
- *functions information*: This defines the algorithms used for initialization, adaptation, and training.
- *weights and biases*: This defines the network's dynamic parameters including weight matrices and bias vectors.

A primary motivation for the use of a CM is to support the communication of neural networks in a portable fashion. Because a CM does not require the use of a specific representation language in order to be communicated between agents, CMs may be represented and communicated in a variety of languages including LISP, Prolog and KIF [6]. CMs may also be represented as objects comprised of multiple subobjects. For example, table 1 shows an InputWeights object that is comprised of a set of parameters including a learning parameter (`learn_param`) that is defined by a LP subobject [3]. Note that because of its modular structure, a CM may be communicated either partially or in its entirety. This may help to improve system performance by allowing only those parts of a CM that are required by agents to be communicated.

Although portions of a CM may be of interest to human readers (e.g. header information), a CM is intended to be communicated in a machine-readable representation. Thus, agents are required to make such knowledge available for human consumption where necessary. Providing a means by which humans can examine knowledge related to CMs would be required in order to help agent or neural network designers identify existing CMs that satisfy their needs.

3.2. Communicating Neural Network Knowledge

Depending on the requirements of a system, the protocols required to communicate knowledge between agents may vary. In the context of the CMT framework, however, protocols are established that offer some standardized means by which to communicate neural network knowledge. These protocols are required in order for agents to conduct intelligent dialog with respect to such knowledge.

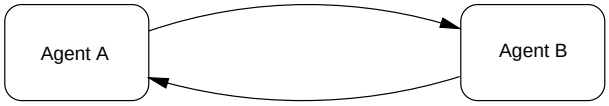
The CMT framework uses the *Knowledge Query and Manipulation Language* [5, 11] to communicate CMs, or parts thereof, between agents. KQML is a language and protocol for exchanging knowledge. When an agent uses KQML to communicate knowledge, it does so by passing a KQML message. Each KQML message is associated with a *performative* that defines the permissible operations that may be attempted on knowledge maintained and communicated by agents. In the CMT framework, KQML is used to perform a variety of operations on CMs and is particularly useful in defining the context surrounding a communicated CM. For example, suppose a sending agent wishes to insert a CM into a receiving agent's knowledge base. In this case, the sending agent may send a KQML message containing the CM and an associated `insert` performative. By examining the received performative, the receiving

Table 1. InputWeights object.

InputWeights		
Type	*Parameter*	*Description*
int	delay	defines a tapped delay line between the *jth* input and its weight to the *ith* layer
string	init_func	defines the function used to initialize the weight matrix going to the *ith* layer from the *jth* input
int	learn	defines whether the weight matrix to the *ith* layer from the *jth* input is to be altered during training and adaptation
string	learn_func	defines the function used to update the weight matrix going to the *ith* layer from the *jth* input during training
LP	learn_param	defines the learning parameters and values for the current learning function of the *ith* layer's weight coming from the *jth* input
int	size	defines the dimensions of the *ith* layer's weight matrix from the *jth* network input
string	weight_func	defines the function used to apply the *ith* layer's weight from the *jth* input to that input
LP		
Type	*Parameter*	*Description*
double	lr	defines the learning rate
double	mc	defines the momentum constant

agent will understand that a request has been made to insert the content of the received message (in this case, a CM) into its knowledge base. As another example, suppose that a sending agent wishes to have a receiving agent execute a CM in the receiving agent's local environment. In this case, the sending agent may send a KQML message containing the CM and an associated `achieve` performative. When the message is received, the receiving agent will examine the received performative and understand that a request has been made to execute the received CM in its local environment. In some cases, only a portion of a CM will be communicated between agents. For example, suppose a sending agent wishes to acquire a CM contained in the receiving agent's knowledge base where the name of the CM is `RECON` and the version is `2.0.1`. The sending agent may define the query constraints as a Header structure containing two defined parameter values `name=RECON` and `version=2.0.1`. This agent may then send a KQML message containing the Header structure and an associated `ask-one` performative. When the receiving agent receives the message, it will understand that a request has been made to locate the first CM that matches the query constraints as defined by the received Header structure. Figure 2 shows an example KQML message where an agent A queries an agent B for a `<CM>` that contains a specified `<HEADER>` where `<HEADER>` and `<CM>` represent logic-based expressions (or objects) of Header and CM knowledge, respectively. Note that we may specify constraints on the structure to be returned by the receiving agent (i.e. a full or partial CM) through the `:reply-with` field of a KQML message.

```
(ask-one
  :sender A
  :receiver B
  :content ( <Header> )
  :reply-with <CM>
  :ontology CM)
```




```
(tell
  :sender B
  :receiver A
  :content ( <CM> )
  :ontology CM)
```

Figure 2. Querying an agent using a KQML ask-one performative.

By using KQML, the CMT framework inherits the communication protocols required in order to communicate CMs in a multi-agent system. Note that although the CMT framework is designed to facilitate the communication of CMs between agents, agents may also be required to communicate additional knowledge depending on the task domain.

3.3. Managing Neural Network Knowledge

Although different multi-agent applications will necessitate varying functional requirements of its agents, the CMT framework requires that agents must facilitate a core set of services for managing CMs, and that these services be initiated through protocols inherited from the underlying

KQML communication infrastructure. Depending on the system, services for managing CMs may be carried out by a wide number and variety of agents. However, agents that are responsible for carrying out such services will typically have the same general architecture. We refer to a set of agents that carry out CM-related services as *CM agents*. In general, CM agents will have a set of proactive services for initiating communication with other agents and a set of reactive services for responding to communication initiated by other agents. CM agents may be comprised of *CMProducer*, *CMConsumer*, *CMRepository*, and *CMBroker* agents.

A CMProducer agent is primarily responsible for CM creation-related services. These services are concerned with the creation of CMs from trained neural networks as well as the submission of these CMs to the system. CMProducer agents are the least autonomous of all the CM agent as they require interaction with human operators (i.e. neural network developers). In order to create a CM, a human operator instructs the CMProducer to extract parameters from a specific neural network that has been trained in the local training environment and to map these parameters into a CM in some appropriate representation language or binary format. This mapping of neural network parameters to a CM is performed using an environment-specific *NN2CM* translator. Once a CM is created, it may be submitted to the system for use. Submission of CMs to the system may involve conducting dialog with other agents in order to determine where to send a newly created CM. The general architecture of a CMProducer agent is shown in figure 3. Here, we note that a CMProducer may also be responsible for receiving neural network knowledge from other agents in order to facilitate network transfer. In such cases, parameters from received CMs are extracted using a CM2NN translator and made available to the local training environment for (manual) training by a human operator.

A CMConsumer agent is primarily responsible for CM execution-related services. These services are concerned with the execution of received neural networks. In order to execute a received neural network, a CMConsumer must first instantiate a received CM by using a CM2NN translator that maps knowledge from the received CM into a representation that may be used in the local execution environment. A CMConsumer is also responsible for determining an appropriate set of input (e.g. sensor) data and providing this data to the execution environment in real-time. In many cases, a CMConsumer may use meta-knowledge from a received CM to determine the appropriate sets of input data required by the network. In addition, such knowledge may also be used to assess the meaning of network results. Finally, a CMConsumer may also be responsible for initiating dialog with other agents in order to communicate results, search for more powerful CMs, *etc*. The general architecture of a CMConsumer is shown in figure 4.

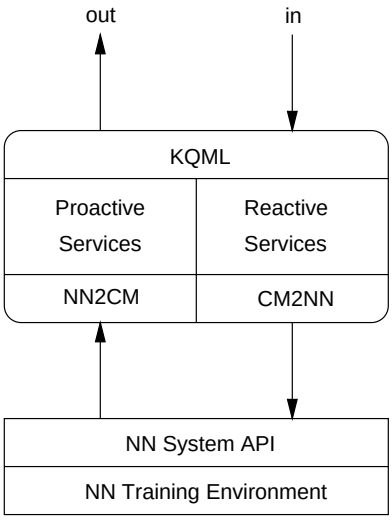


Figure 3. CMProducer agent.

A CMRepository agent is responsible for CM maintenance and storage services. These services include CM query and search services as well as translation services for mapping CMs to and from various representation languages and formats. CM query and search services allow agents (and humans) to search a CMRepository agent's knowledge base for appropriate CMs. By allowing humans to browse the contents of a repository and examine descriptions of CMs, they may be better able to determine which neural networks are appropriate for use in their agents, or to determine appropriate networks from which to extract parameter values to be used for network transfer. A CMRepository may also initiate dialog with other agents in order to alert agents of updated CMs, forward queries to other agents, *etc*.

Finally, a CMBroker agent is responsible for carrying out marketing and advertising services on behalf of CM agents. When a CM agent is instantiated, it registers its services with a CMBroker which, in turn, recommend, advertise and market these services to other agents. In general, a CM agent will typically initiate dialog with a CMBroker before communicating with other CM agents in order to determine which agents provide the most appropriate services for its needs.

4. Simulating Aerial Reconnaissance

In a military campaign, the activity of aerial reconnaissance is invaluable in providing information on the location of mobile military objects including tanks, troops, *etc*. In an aerial reconnaissance mission, Unmanned Aerial Vehicles (UAVs), reconnaissance aircraft, and satellites are used

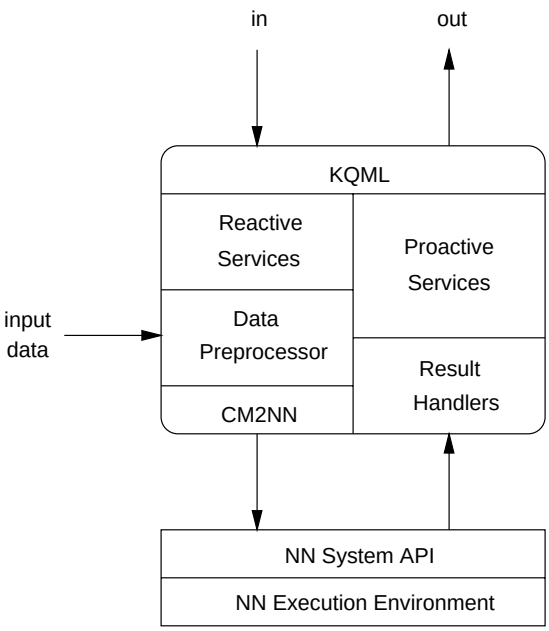


Figure 4. CMConsumer agent.

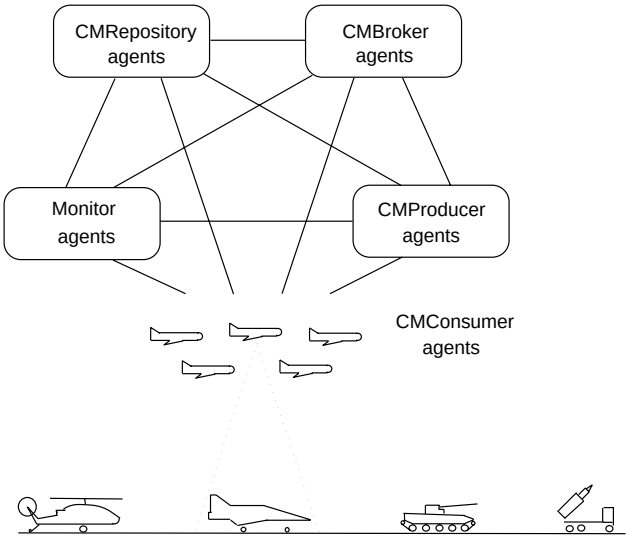


Figure 5. AIFFAR system.

to provide still images and video of targets in a defined geospatial region. Often, these images are first forwarded to a central command location where they are manually analyzed and then forwarded to appropriate commanders in the field. Unfortunately, this process is time consuming. Even in cases where video is streamed directly to commanders in the field, commanders must still manually analyze the video stream and wait until events of interest are displayed.

In order for commanders to assess battlespace conditions in a timely fashion, it is important to receive aerial reconnaissance information in a timely manner. In addition, it is important for commanders to receive only those images that are required to assess the situation at hand. In this section we present a system that simulates the filtering and forwarding of reconnaissance images to appropriate commanders in the field. We refer to this system as the *Automatic Image Filtering and Forwarding for Aerial Reconnaissance* (AIFFAR) system. The AIFFAR system is shown in figure 5.

The AIFFAR system is a multi-agent system comprised of CM agents. Here, CMProducer agents are used to create CMs from neural networks trained to recognize images of mobile military objects. CMProducer agents then submit these CMs to the system by first initiating dialog with a CMBroker in order to determine appropriate CMRepository agents to store the CMs, and then forwarding the CMs to those CMRepository agents directly.

In the AIFFAR system, CMConsumer agents represent UAVs capable of capturing, filtering and forwarding aerial reconnaissance images. Using the architecture as shown in figure 4, a CMConsumer is responsible for

- receiving CMs through KQML messages
- mapping received CMs to neural network parameters using a CM2NN translator
- instantiating network parameters in the local network execution environment
- capturing and providing continuous, raw image data to the system
- mapping raw image data into a numerical representation for input into the network
- executing the network and deriving classification results
- computing classification performance
- returning results.

Results generated by a CMConsumer may be forwarded to other agents including *monitor agents*. Monitor agents are used to provide commanders an interface to AIFFAR. This interface allows commanders to request notification when an image of interest is recognized.

In our aerial reconnaissance scenario, CMConsumer agents traverse a geospatial region divided into multiple sectors as shown in figure 6. Here, each sector represents a typical military area of interest. For example, sector A may represent a battlefield containing objects typically found in ground-based warfare including tanks, trucks and surface-to-air missiles (SAMs), while sector B may represent an

airfield containing objects including fighter jets, helicopters, and radar systems. In addition, objects that are not normally found in a particular sector may appear periodically. Thus, for example, it may be possible to capture an image of a helicopter in sector A or a tank in sector B. In this simulation, each CMConsumers starts in a particular sector and, upon initialization, is assigned a network for classifying images of mobile military objects typically found in that sector. The goal of each CMConsumer is to maintain its ability to correctly classify images of objects while it moves from sector to sector.

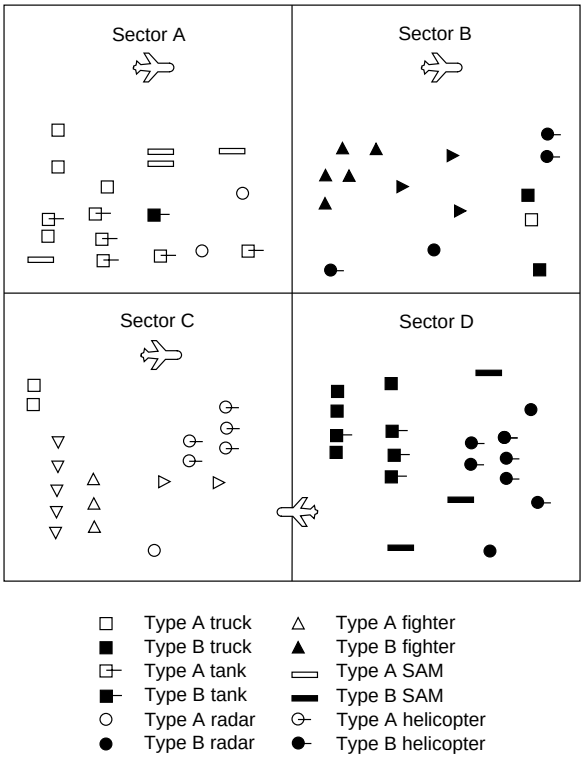


Figure 6. UAVs traverse a geospatial region divided into multiple sectors.

The AIFFAR system is a controlled simulation and intended as a proof-of-concept implementation of the CMT framework. The motivation for the use of this simulated scenario was to illustrate situations where agents exist in a dynamic environment and require the communication of neural network knowledge in order to maintain their classification performance. We do not, however, make any claims about the applicability of the CMT framework to actual aerial reconnaissance systems nor provide any assessments regarding the performance of this framework with respect to such systems. The AIFFAR system was implemented and tested under Solaris 2.6 using Sun JDK1.2, the Jackal 3.1 KQML package [2] and the MATLAB 5.2/C API for the training and execution of neural networks.

5. Experiment in Neural Network Communication

In order to show the benefits of communicating neural networks between agents, we conducted a simple experiment in order to show how the communication of CMs could improve a CMConsumer agent's ability to maintain its classification performance in a dynamic environment. In this experiment, neural networks were implemented as classifier systems rather than learning systems. The motivation for this was to be able to illustrate the benefits of communicating neural network knowledge for facilitating agent performance without having to also assess the contribution to agent performance due to an internal (and continuously learning) connectionist-based mechanism. By eliminating a network's ability to learn, we were better able to effectively demonstrate the impact of communicating CMs in order to maintain agent performance. As described in Section 2, however, we note that agents that use connectionist-based learning systems may also require an ability to receive and instantiate new neural network knowledge if they are unable to adapt sufficiently to maintain their performance.

In this experiment, CMConsumer agents were tasked with correctly classifying images of mobile military objects. The experiment was divided into two phases: a *control phase* and a *test phase*. These phases were distinguished by the ability to communicate CMs in their respective systems. In the control phase of the experiment, each CMConsumer was assigned a CM for performing classification of objects typically found in its initial sector. Because of their inability to communicate CMs, however, CMConsumer agents used their initially assigned network to classify objects in all other sectors that they visited. Similarly, each CMConsumer in the test phase of the experiment was assigned a CM for performing classification of objects typically found in its initial sector. However, unlike the system in the control phase, the system in the test phase allowed for the communication of new CMs to CMConsumers. We hypothesized that by allowing for the communication of CMs between agents, CMConsumer agents in the test phase of the experiment would exhibit a greater ability to classify images of mobile military objects over those in the control phase of the experiment.

In both the control and test phases of the experiment, ten CMConsumer agents traversed a geospatial region divided into four sectors: two battlefield sectors and two airfield sectors (as shown in figure 6). Four neural networks for classifying images typically found in each of these four sectors were trained, converted into CMs, and submitted to

the system by four CMProducer agents. These CMs were subsequently stored and maintained by two CMRepository agents — one for storing battlefield-related CMs and the other for storing airfield-related CMs. As CMConsumer agents were initialized, they would conduct dialog with other CM agents in order to identify and acquire an initial CM from one of the two CMRepository agents.

The capturing of images by a CMConsumer agent was simulated through the random selection of image files. To provide a more realistic scenario, a CMConsumer would occasionally capture the image of an object not typically found in its currently occupied sector, or wander into an adjacent sector and capture images from that sector. After every 30 images captured and processed, a CMConsumer would randomly select a new sector to patrol. In addition, each CMConsumer was given an ability to compute its own classification performance by comparing the results of its network with additional information provided about each image file. This information contained a description of the object in the image including the object's type. The motivation for allowing a CMConsumer to access this information was to simulate the feedback it may receive from some external source (e.g. human monitor). After an image was presented to a CMConsumer, its embodied network would derive a classification result, compare this result to information about the object in the image, and compute its classification performance.

As CMConsumer agents moved from sector to sector in the control phase of the experiment, they would continue to capture and classify images regardless of potentially dramatic decreases in their classification performance. In contrast, CMConsumer agents in the test phase of the experiment would attempt to acquire new CMs if their classification performance dipped below some specified threshold. In this case, the minimum classification performance threshold was set at 80.0% (i.e. each CMConsumer was required to maintain its ability to correctly classify 80% of its captured images). The acquisition of new, more appropriate CMs involved the initiation of dialog with other agents via the CMT framework. If a CMConsumer reached the minimum classification performance threshold, it would initiate dialog with a CMBroker to find other CMConsumer agents occupying the same sector. The CMConsumer would then initiate dialog with these agents in order to assess their current classification performance. If their classification performance exceeded the minimum defined threshold, the CMConsumer would then acquire a CM from one of these agents (e.g. the one with the highest classification performance). If no other CMConsumer agents were found in the same sector, or those that did exist in the same sector exhibited poor classification performance, the CMConsumer would initiate dialog with a CMRepository in order to acquire a new CM.

5.1. Reconnaissance Image Data

In this experiment, we used simulated reconnaissance data in the form of GIF and JPEG images as input for network training and recall. The images were grouped into seven general object types: `fighter`, `helicopter`, `radar`, `sam`, `tank`, `troop`, and `truck`. The motivation for the selection of these object types was to provide a set of mobile military objects that may be of interest to commanders in combat situations. Figure 7 shows a sample of images used in this simulation.

Figure 7. Simulated reconnaissance images (clockwise from top left): helicopter, fighter, tank, and SAM.

Each object type was associated with three image variations: `rotation`, `low-visibility`, and `low-light`, and some object types consisted of two different models, `A` and `B`. Figure 8 shows two fighter models with rotation, low-visibility and low-light image variations. In addition, some object types had a `camouflage` image variation. Two such objects are shown in figure 9.

To simplify the experiment, a number of assumptions and constraints regarding the simulated images were made. First, we assumed that all images were captured from a relatively constant altitude and from an identical type of camera with identical zoom settings. This assumption was necessary in order to restrict large variations in the size of objects in an image. Second, we assumed that each CMConsumer was either remotely controlled or used a global positioning system (GPS) in addition to possible image preprocessing algorithms for centering objects within the image (e.g.

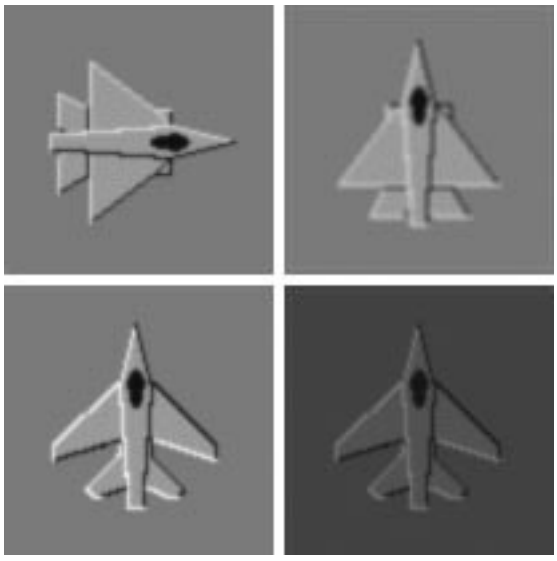


Figure 8. Sample image variations (clockwise from top left): Type A fighter (rotated), Type A fighter (low-visibility), Type B fighter (low-light), Type B fighter (normal).

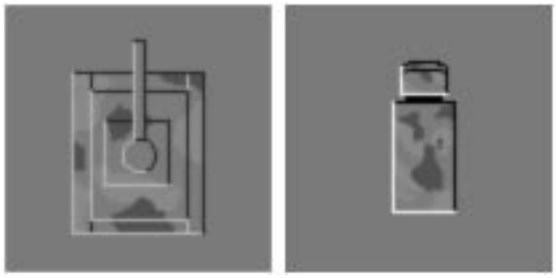


Figure 9. Camouflaged tank and truck.

edge detection algorithms). This assumption was necessary in order to ensure that all objects would occupy the center portion of an image so that the object was not cropped at the edge of the image. Third, we assumed that objects within an image depicted little or no shadow and that images of objects were always captured at a 180° angle to the object. This assumption was necessary in order to restrict variations in the color/shadow of an image which may have affected the recognition of the object. Finally, we assumed that each image had a constant dimension. This assumption was necessary in order to restrict variations in the size of the image background.

Before information from images could be used for training by CMProducer agents, or execution by CMConsumer agents, the raw data from these images was first preprocessed. Because the focus of this paper was not concerned with the development of new, robust computer vision algorithms, we used a simple ad-hoc algorithm for extracting RGB color and alpha values from raw image data. This algorithm involved scanning each pixel of a GIF or JPEG image and extracting its associated RGB/alpha value. The RGB/alpha value for each pixel was then stored in a hashtable as a single integer. This integer was also used as the key into the hashtable. If a RGB/alpha value for a pixel mapped to the same slot in the hashtable, the number of pixels associated with this slot was incremented. The idea of this algorithm was to sum the number of pixels that had the same RGB/alpha value. The sums of the number of pixels for each RBG/alpha value were then sorted in descending order into a RGB/alpha pixel array. This array defined the number of pixels for each RGB/alpha value such that the first element in the array represented the number of pixels for the most frequent RGB/alpha value in the image. For most images, this value represented the number of pixels of the image background.

5.2. Network Architectures

In this experiment, four neural networks NN-A, NN-B, NN-C, and NN-D were used to classify images of objects in sectors A, B, C, and D, respectively. All four networks were relatively simple, three-layer, backpropagation networks that used a Levenberg-Marquardt algorithm for accelerated training. In addition, each network used four inputs (one for each of the first four values of a generated RGB/alpha pixel array).

As shown in table 2, each network was trained on images of objects typically found in its corresponding sector. For example, network NN-A was trained on images of `Type A tank` and `Type A truck` objects while network NN-B was trained on images of `Type B fighter` and `Type B helicopter` objects.

One of the motivations for the CMT framework was to provide a framework that allowed agents to modify not only the types, but also the number, of possible classifications it might derive for a single input vector. In this experiment, we illustrated this property through the use of networks that varied in the number of elements contained in their output vectors. Here, the number of elements in an output vector denoted the number of possible classifications a network could derive. For example, the topology of network NN-A contained five outputs denoting its ability to classify five types of objects while the topology for network NN-B contained three outputs denoting its ability to classify three types of objects. The architectures of the networks also varied in their number of hidden layers as well as their values for weight and bias matrices.

In addition to architectural variations, variations also

Table 2. Summary of neural network properties.

Property	*NN-A*	*NN-B*	*NN-C*	*NN-D*
layers	3	3	3	3
input	4	4	4	4
hidden	6	8	8	4
output	5	3	4	4
learning rate	.05	.10	.09	.03
momentum	.90	.50	.01	.40
avg. conv.	66	54	130	49
objects				
fighter		B		B
helicopter		B	A	B
radar	A	B		B
sam	A			
tank	A		A	
troop	A		A	
truck	A		A	B

existed between the network's training parameters including their momentum constants and learning rates. Table 2 also shows each network's average convergence (in epochs) given randomly initialized weights.

5.3. Simulation Results

Results of this experiment were obtained by comparing the classification performance of CMConsumer agents during the control phase with the classification performance of these agents during the test phase. For the control phase of the experiment, we let $C = \{c_1, c_2, \ldots, c_n\}$ be the set of CMConsumer agents tasked with classifying images using networks assigned at initialization. Each CMConsumer agent was defined as a quadruple $c_i = (K_i, V_i, U_i, r_i)$, $1 \leq i \leq n$, where K_i represented a set of correctly classified images, V_i represented a set of incorrectly classified images, U_i represented a set of unclassifiable images, and

$$r_i = \frac{\|K_i\|}{\|K_i\|+\|V_i\|+\|U_i\|}$$

represented the classification rate of the agent. For the test phase of the experiment, we let $C' = \{c_1', c_2', \ldots, c_n'\}$ be the set of CMConsumer agents tasked with classifying images using networks acquired through communication with other agents. Here, each CMConsumer agent was defined as a quadruple $c_i' = (K_i', V_i', U_i', r_i')$, where K_i' represented a set of correctly classified images, V_i' represented a set of incorrectly classified images, U_i' represented a set of unclassifiable images, and

$$r_i' = \frac{\|K_i'\|}{\|K_i'\|+\|V_i'\|+\|U_i'\|}$$

represented the classification rate of the agent. In both phases of the experiment, each CMConsumer agent attempted to correctly classify 100 images. The final classification rates for each CMConsumer agent during the control and test phases of the experiment are shown in table 3.

Table 3. Final classification rates for CMConsumer agents tasked with classifying 100 images during the control and test phases.

Final Classification Rates		
CMConsumer	r	r'
1	0.552	0.823
2	0.597	0.805
3	0.667	0.842
4	0.644	0.840
5	0.602	0.838
6	0.733	0.859
7	0.632	0.844
8	0.529	0.816
9	0.648	0.801
10	0.721	0.842

Here, we see that the final classification rates for the test phase of the experiment (column r') were greater for each CMConsumer agent than during the control phase of the experiment (column r). The final mean classification rate during the control phase was

$$\overline{x}_r = \frac{\sum_i^n r_i}{n} = 0.632$$

while the final mean classification rate for the test phase was

$$\overline{x}_{r'} = \frac{\sum_i^n r_i'}{n} = 0.831.$$

Thus, CMConsumer agents in the test phase showed a 31.4% increase in their final average classification rates. This result supported the hypothesis that the communication of CMs between agents increased the image classification capabilities of CMConsumers agents during run-time.

Figure 10 compares the average classification rate for all agents during the control phase with the average classification rate for all agents during the test phase. Note that in the test phase, CMConsumer agents initiated a search for more powerful models when their classification performance dipped below 80.0%.

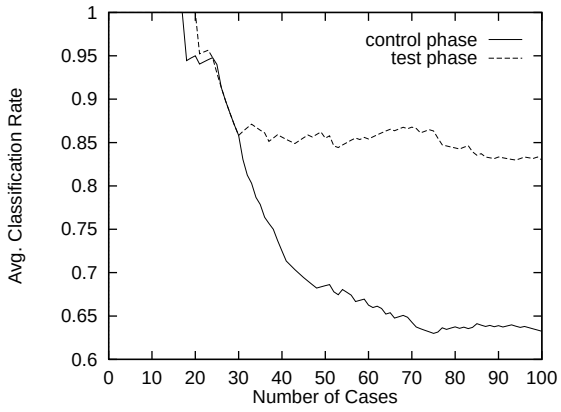


Figure 10. Comparison of average classification performance of agents during control and test phases.

6. Related Work

Previous research efforts have studied issues related to the use of parameters from existing neural networks to facilitate the training of new networks. In [17] Pratt introduced the notion of transferring neural networks, or parts thereof, in the same representational formalism to new learning tasks. The motivation for her research was to show the effects of using parameter values from existing networks on the training of new networks. Although her research supported the claim that network transfer was valuable for increasing the training performance of a network, it assumed the existence of mechanisms for communicating neural network parameters from the source network to the new (target) network. Thus, no protocols for communicating network parameters, nor models for representing such parameters, were presented. One motivation for the development of the CMT framework was to provide the mechanisms by which network transfer may take place in an agent-based system.

Previous research efforts have also studied the communication of learned models between agents. Perhaps the most similar to the ideas presented in this paper were those presented by Stolfo *et. al.* [18]. Their research presented a system, known as the *JAM system*, for communicating learned models and classifiers between agents. Like the CMT framework, the JAM system facilitated ”Plug-and-Play” of learned classifiers. However, the concepts presented in this paper differs from described in [18] in a number of significant ways. Perhaps the most important difference lies in the models used by each system. In the JAM system, agent learning was carried out by a variety of algorithms including ID-3, Bayesian, and CART while in the CMT framework, agent learning is connectionist-based. Second, the JAM system used ad-hoc communication protocols in order to communicate models between agents. This differs from the CMT framework where the communication protocols are inherited from its underlying KQML infrastructure. Third, models and classifiers in the JAM system were communicated as native applications or applets which potentially limit the portability of such models. In the CMT framework, CMs may be communicated in a number of representations languages and formats allowing them to be used in a variety of environments.

7. Conclusions

This paper presented the CMT framework for communicating neural network knowledge between agents. The motivation for this framework was to allow for the communication of neural network knowledge between agents in order to facilitate agent learning and pattern classification. The CMT framework was comprised of a model specification for representing neural network knowledge, a specification of services for managing neural network knowledge in a multi-agent system, and a protocol for communicating neural network knowledge between agents.

In this paper, we presented the connectionist model (CM) as a model for representing neural network knowledge. Unlike other specifications that have been proposed for modeling neural networks, a CM is comprised of both network parameters and meta-knowledge that describes additional information about a network. Such information may be useful in allowing agents to determine, for example, the semantics of network outputs. Because CMs may be communicated in a variety of representation languages and formats, they provide a flexible model that may be used to communicate neural network knowledge in a portable fashion. In addition, we identified and described protocols for communicating CMs between agents. These protocols were inherited through the use of the KQML agent communication language. In the CMT framework, KQML is used to perform a variety of operations on CMs and is particularly useful in defining the context of a communicated CM. Through KQML, agents may communicate CMs as well as initiate CM-related services for creating, storing, managing, and executing CMs.

Finally, we described the application of the CMT frame-

work to a simulated aerial reconnaissance system. This system, referred to as the AIFFAR system, was comprised of a set of CM agents for managing networks trained to recognize images of mobile military objects. Using the AIFFAR system, we conducted a simple experiment that showed how the use of the CMT framework could help agents maintain their classification performance in a highly dynamic environment.

References

[1] Y. S. Abu-Mostafa. Learning from hints in neural networks. *Journal of Complexity*, 6(2):192–198, 1989.

[2] R. S. Cost, T. Finin, Y. Labrou, X. Luan, Y. Peng, I. Soboroff, J. Mayfield, and J. Boughannam. Jackal: A java-based tool for agent development. In J. Baxter and B. Logan, editors, *Software Tools for Developing Agents: Papers from the 1998 AAAI Workshop*, pages 73–83. AAAI Press, 1998.

[3] H. Demuth and M. Beale. *Neural Network Toolbox User's Guide, Version 3.0*. The MathWorks, Inc., Natick, MA, 1998.

[4] E. Fiesler. Neural network classification and formalization. *Computer Standards and Interfaces*, 16(3):231–239, 1994.

[5] T. Finin, Y. Labrou, and J. Mayfield. KQML as an agent communication language. In J. Bradshaw, editor, *Software Agents*, Cambridge, MA, 1997. MIT Press.

[6] M. R. Gesenereth and R. Fikes. Knowledge interchange format, version 3.0 reference manual. Technical Report Logic-9201, Computer Science Department, Stanford University, 1992.

[7] J. H. Holland. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning, an Artificial Intelligence Approach*. Morgan Kaufmann, 1986.

[8] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[9] M. Kaiser, R. Dillman, and O. Rogalla. Communication as the basis for learning in multi-agent systems. In *ECAI '96 Workshop on Learning in Distributed AI Systems*, Budapest, Hungary, 1996.

[10] R. L. Kashyap. Algorithms for pattern classification. In J. M. Mendal, editor, *A Prelude to Neural Networks: Adaptive and Learning Systems*, pages 81–113, Englewood Cliffs, 1994. Prentice-Hall.

[11] Y. Labrou and T. Finin. A proposal for a new KQML specification. Technical Report TR CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, 1997.

[12] P. Maes. Modeling adaptive autonomous agents. In C. G. Langton, editor, *Artificial Life, An Overview*, Cambridge, Massachussets, 1995. MIT Press.

[13] M. Mataric. Reinforcement learning in the multi-robot domain. *Autonomous Robots*, 4(1):78–83, 1997.

[14] H. S. Nwana. Software agents: An overview. *Knowledge Engineering Review*, 11(3):1–40, 1996.

[15] E. Plaza, J. Arcos, and F. Martin. Cooperative case-based reasoning. In G. Weiβ, editor, *Distributed Artificial Intelligence Meets Machine Learning*, pages 180–201, Berlin, 1997. Springer-Verlag.

[16] L. Pratt. Experiments on the transfer of knowledge between neural networks. In S. Hanson, G. Drastal, and R. Rivest, editors, *Computational Learning Theory and Natural Learning Systems, Constraints and Prospects*, pages 523–560. MIT Press, 1994.

[17] L. Y. Pratt. *Transferring Previously Learned Back-Propagation Neural Networks to New Learning Tasks*. PhD thesis, Department of Computer Science, Rutgers University, 1993.

[18] S. Stolfo, A. Prodromidis, S. Tselepis, W. Lee, D. Fan, and P. Chan. JAM: Java agents for meta-learning over distributed databases. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD-97)*, Newport Beach, CA, 1997.

[19] G. G. Towell and J. W. Shavlik. The extraction of refined rules from knowledge-based neural networks. *Machine Learning*, 13(1):71–101, 1991.

[20] G. Weiβ. Adaptation and learning in multi-agent systems: Some remarks and a bibliography. In G. Weiβ and S. Sen, editors, *Adaptation and Learning in Multi-Agent Systems*, pages 1–21. Springer-Verlag, 1996.