Merging Neural Networks in a Multi-Agent System

Stephen Quirolgico
Teknowledge Corporation
Fairfax, Virginia 22030
squirolg@teknowledge.com

Kip Canfield and Tim Finin
University of Maryland, Baltimore County
Baltimore, Maryland 21250
{canfield, finin}@research.umbc.edu

James A. Smith
NASA Goddard Space Flight Center
Greenbelt, MD 21770
jasmith@hemlock.nasa.gsfc.gov

Abstract

The Connectionist Model Transfer (CMT) framework was proposed to allow an agent the ability to instantiate and execute neural networks received from other agents in order to maintain its learning or classification performance in a dynamic environment. One limitation of the CMT framework was that it assumed the existence of at least one network in the system that embodied the classification mappings required to exhibit satisfactory classification performance for each situation that an agent might encounter. In some cases, however, an agent required a single network that embodied classification mappings of several existing networks in order to maintain its performance. In such cases, a new network was manually trained and submitted to the system. In this paper, we extend the CMT framework for facilitating the dynamic merging of distributed networks. This extension allows agents to utilize new, automatically trained networks that embody the classification mappings of several distributed, and dynamically selected, networks. We apply this extension to a simulated aerial reconnaissance system in order to show how the merging of neural networks can help maintain the performance of agents tasked with recognizing images of mobile military objects.

Introduction

In many agent-based systems, neural networks are used to implement the mechanisms by which agents learn and classify patterns. In such systems, agents are typically embedded with a connectionist-based model during implementation with no means for overriding this model during run-time. Thus, the way in which an agent learns or classifies patterns often remains static during the life of the agent. As a result, agents that implement neural networks may be unable to adapt their learning or classification behavior to changes in their environment (especially if such changes are highly dynamic), and thus experience a degradation in performance. In order to resolve this limitation, the *Connectionist Model Transfer* (CMT) framework was proposed for supporting the communication of neural networks between agents (Quirolgico *et al.* 1999). By applying the CMT framework, a sending agent may communicate subsymbolic knowledge (i.e. knowledge encoded in a neural network) to a receiving agent which, in turn, could instantiate the received network in a "Plug-and-Play" fashion — allowing the receiving agent to dynamically modify its learning or pattern classification behavior in real-time.

In systems that implement the CMT framework, the acquisition of a new network by an agent is triggered by its network's learning or classification performance. If the performance of an agent's currently executing network drops below some minimum threshold, the agent may attempt to acquire a more appropriate network (i.e. a network that embodies the correct classification mappings for the task at hand) from another agent in the system. However, if no such network is available in the system, the agent must wait until an appropriate network is manually trained and submitted to the system. The process of manually training and submitting a new network requires human intervention in order to

- determine the classification mappings of the network.
- create an appropriate training set for the network.
- create the necessary environment-specific files for training the network.
- train and test the network.
- map the network to an appropriate representation for communication.
- submit the network to the system.
- inform the agent of the network.

Because of the time and effort required to manually train and submit networks to the system, this process may be inappropriate for systems comprised of agents that are expected to perform learning or classification tasks in real-time.

In some cases, an agent may require a network that embodies classification mappings that are already embodied by other networks in the system. In such cases, it is possible to automate the training of a new network that embodies the classification mappings of these networks. This process involves acquiring knowledge from distributed *source* networks (i.e. networks residing at remote locations), merging this knowledge into a form that may be used to train a new network, training the new network, and submitting the new network to the system. We refer to this process as *network merging*. Through network merging, we may derive a *composite* network that embodies the classification mappings of multiple source networks. In this paper, we extend the CMT framework in order to support network merging of dynamically selected source networks. This extension allows an agent that is restricted to executing a single network at a time the ability to potentially maintain its performance by acquiring composite networks that embody the most appropriate classification mappings for its specific learning or classification task. In addition, this extension may also potentially improve system performance by:

- facilitating the creation of more accurate networks.
- encouraging the introduction of smaller (and less complex) networks to the system.
- minimizing the need for human intervention.

We begin this paper by presenting an overview of the CMT framework in the context of network merging. This will involve a discussion of issues related to the modeling of neural networks, the communication of neural networks between agents, and the management of neural networks within a multi-agent system. We then describe the application of the CMT framework to a simulated aerial reconnaissance system. Using this system, we demonstrate how the use of composite networks derived through network merging can help maintain the performance of agents tasked with classifying images of mobile military objects.

General Framework

The CMT framework is comprised of a model specification for representing neural network knowledge, a protocol for communicating neural network knowledge between agents, and a specification of agent services for managing and using neural network knowledge.

Modeling Neural Networks

In order to communicate neural networks between agents, a suitable specification for their representation must exist. Although a number of specifications have been proposed for modeling the architecture of neural networks (Fiesler 1994; Demuth & Beale 1998), these models lack a representation that provides meta-knowledge about a network. Such knowledge may be used by agents in order to assess attributes of a network that are not typically specified by the network's structural properties. In the CMT framework, neural network knowledge (i.e. a network's structural parameters and meta-knowledge) is represented by a *Connectionist Model* (CM). A CM is an ontological specification for representing neural network knowledge. The general ontology of a CM is shown in Figure 1.

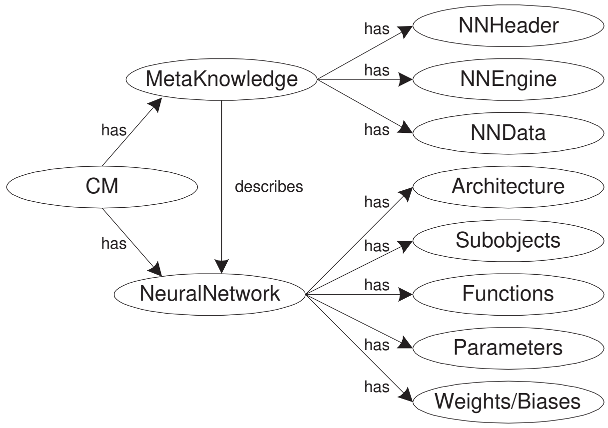

Figure 1: Connectionist Model ontology.

A CM is comprised of ontologies for specifying a network's meta-knowledge and structural properties. The meta-knowledge ontology of a CM is, in turn, comprised of *NNHeader*, *NNEngine* and *NNData* ontologies. The NNHeader ontology specifies general knowledge about a network including the network's

- name, description, keywords, version, type, *etc.*
- domain, environmental, and contextual constraints.
- author, institution, and contact information.

The NNEngine ontology specifies knowledge related to the instantiation and execution of a network including

- information on translators for mapping CMs between various knowledge representation formats as well between various neural network development and execution environments.
- information on available engines (e.g. libraries, classes, *etc.*) for executing this network.

The NNData ontology specifies knowledge about a network's input and output data including its

- sample training and test data.
- classification categories.
- the semantics of classification outputs.
- required sources of input or sensor data.
- required data preprocessor.

- optional rule-base for symbolically describing its input-to-output classification mappings.

With respect to network merging, the knowledge specified in a NNData ontology is essential for determining how to train a composite network. Because the CMT framework requires that each CM that exists in the system embody knowledge about its network's input and output data, such knowledge may be easily acquired and merged from several distributed CMs in order derive the knowledge required to train a composite network.

The neural network ontology of a CM describes the structural properties of a network and is based on the MATLAB network object specification (Demuth & Beale 1998). This ontology is, in turn, comprised of ontologies for specifying

- architectural properties (i.e. the number of network subobjects and they are connected).
- subobject structure properties (i.e. the properties of array structures that define the network's inputs, layers, outputs, targets, biases, and weights).
- functions (i.e. the algorithms used for initialization, adaptation, and training).
- parameters (i.e. the properties and values for the various initialization, adaptation, and training functions).
- weights and biases (i.e. the network's dynamic parameters including weight matrices and bias vectors).

Because CMs are defined by their ontological structure, they may be represented and communicated in a variety of languages including IDL, XML and KIF (Gesenereth & Fikes 1992), as well as binary formats (e.g. Java objects). In addition, their modular structure allows them to be communicated either in their entirety or partially in order to improve system performance.

Communicating Neural Network Knowledge

Although CMs may be used in systems with varying communication infrastructures (e.g. CORBA, FIPA, JINI™, *etc.*), the CMT framework uses the *Knowledge Query and Manipulation Language* (KQML) for communicating CMs, or parts thereof, between agents. KQML (Finin, Labrou, & Mayfield 1997; Labrou & Finin 1997) is a language and protocol for exchanging knowledge between agents. 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. With respect to network merging, the CMT framework inherits the infrastructure by which to communicate NNData from distributed CMs between agents. For example, suppose that an agent **Agent A** wishes to acquire NNData of a particular CM from another agent **Agent B** where the name of the CM is `RECON` and the version of the CM is `2.0.1`. This scenario is shown in Figure 2.

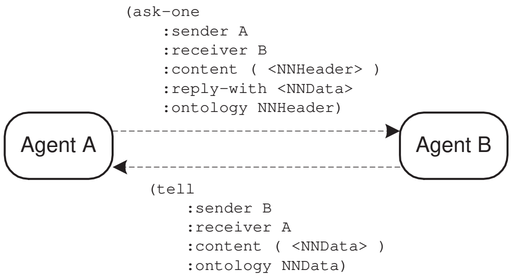

Figure 2: Using KQML to communicate CMs, or parts thereof.

Here, **Agent A** sends a KQML message to **Agent B** containing NNHeader knowledge that is used to define the constraints of the query. In this case, the NNHeader knowledge defines the parameter values `name=RECON` and `version=2.0.1`. In addition, **Agent A** uses the KQML `ask-one` performative which instructs **Agent B** to return the NNData portion of a CM that matches the query constraints as defined by the communicated NNHeader knowledge. Note that we may specify constraints on the structure to be returned to the querying agent through the `:reply-with` field of a KQML message. KQML is also used to submit new trained composite networks back to the system.

Managing Neural Network Knowledge

The CMT framework requires that a core set of CM-related services be provided by agents in order to manage and use neural networks, and that these services be initiated through protocols inherited from the underlying KQML infrastructure. The set of agents used to perform CM-related services are comprised of *CMProducer*, *CMRepository* and *CMConsumer* agents. In addition, we present a *CMTrainer* agent for facilitating network merging.

CMProducer. A CMProducer is responsible for services that are concerned with the creation of CMs from manually trained neural networks as well as the submission of these CMs to the system. The general architecture of a CMProducer is shown in Figure 3.

CMProducers are the least autonomous of all CM agents 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 parameter values from a specific neural network that has been trained in the local training environment and to map these values into a CM in some

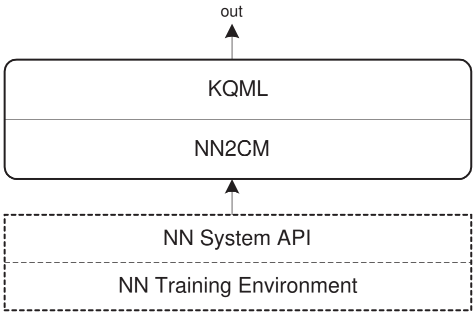

Figure 3: CMProducer agent.

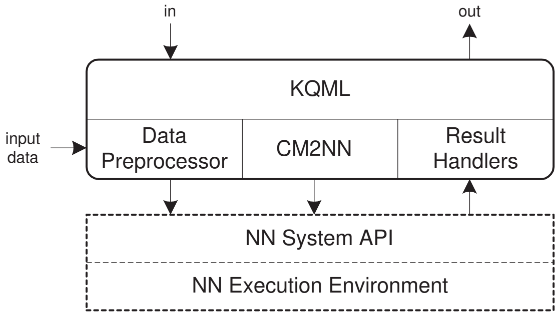

Figure 4: CMConsumer agent.

appropriate representation language or binary format. The mapping from neural network parameter values to a CM is performed using an environment-specific *NN2CM* translator. Once a CM is created, a CMProducer may send the CM to an appropriate agent via a KQML message.

CMRepository. A CMRepository 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'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. A CMRepository may also initiate dialog with other agents in order to inform agents of new CMs, forward queries to other agents, *etc.*

CMConsumer. A CMConsumer is responsible for CM execution-related services. These services are concerned with the instantiation and execution of neural networks extracted from received CMs. When a CMConsumer instantiates and executes a network from a received CM, it exhibits *rote learning.* Rote learning refers to the immediate and direct implantation of knowledge and skills without requiring further inferencing or transformation by the learner (Nwana 1996; Weiβ 1996). Thus, CMConsumers reflect the knowledge and skills embodied by their currently executing network. In a dynamic environment, a CMConsumer may acquire new, more appropriate CMs as its situation in the environment changes. The general architecture of a CMConsumer is shown in Figure 4.

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 must also determine an appropriate set of input (e.g. sensor) data and provide 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 semantics of network results. Finally, a CMConsumer may initiate dialog with other agents in order to acquire new networks for its current situation, communicate network results/performance information, *etc.* Note that a CMConsumer may need to acquire and sample a number of networks for its current situation before it finds one that exhibits sufficient performance for the learning or classification task at hand.

CMTrainer. A CMTrainer is responsible for facilitating network merging. The general architecture of a CMTrainer agent is shown in Figure 5.

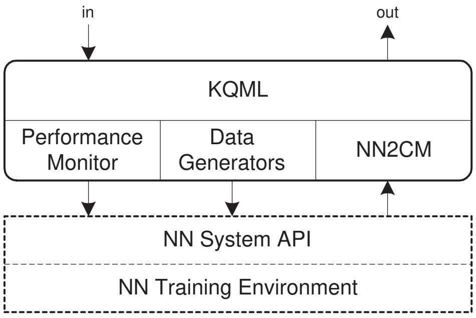

Figure 5: CMTrainer agent.

In the CMT framework, network merging is initiated by a CMConsumer whenever it exists in an environment such that the learning or classification performance of its acquired networks drop below some specified threshold and no other networks are available in the system. In such cases, we refer to the state of the CMConsumer's environment as a *trigger state.* When a CMConsumer initiates network merging, it notifies the CMTrainer that it requires a composite network for its current trigger state. Using a *PerformanceMonitor* that monitors the peformance of all networks sampled in a particular trigger state, the CMTrainer proceeds to identify those networks that have exhibited the highest performance in the CMConsumer's current trigger state. The CMTrainer then acquires the NNData information of these networks from other agents and uses a *DataGenerator* to merge this knowledge into several training files that are used to train a new, composite network. Training files may be comprised of training data, test data, data preprocessors, and environment-specific neural network files. Once the training files are derived, the CMTrainer initiates the training process through a local training environment and derives the composite network. The CMTrainer then maps this composite network into a CM that is submitted back to the system (e.g. either back to the CMConsumer or to a CMRepository).

Note that because the CMT framework supports the communication of structural parameters of a neural network, we may potentially improve the automatic training of composite networks through *network transfer*. Network transfer refers to the reuse of neural network parameter values in order to improve the training of new neural networks (Abu-Mostafa 1989; Pratt 1993; Towell & Shavlik 1991). Research in network transfer has shown that the reuse of parameter values from existing networks may potentially accelerate the training, and improve the accuracy, of new neural networks (Pratt 1994).

Experiment in Network Merging

In order to demonstrate the benefits of network merging, we conducted a simple experiment that showed how the use of composite networks could help maintain the performance of CMConsumers tasked with recognizing images of mobile military objects. This experiment used a simulated aerial reconnaissance system as the testbed application. In this system, CMConsumers representing *Unmanned Aerial Vehicles* (UAVs) were tasked with traversing a geospatial region and capturing images of mobile military objects as shown in Figure 6. The goal of each CMConsumer agent was to correctly classify each image it captured as it moved within the geospatial region. Figure 7 shows a sample of the images used in this simulation.

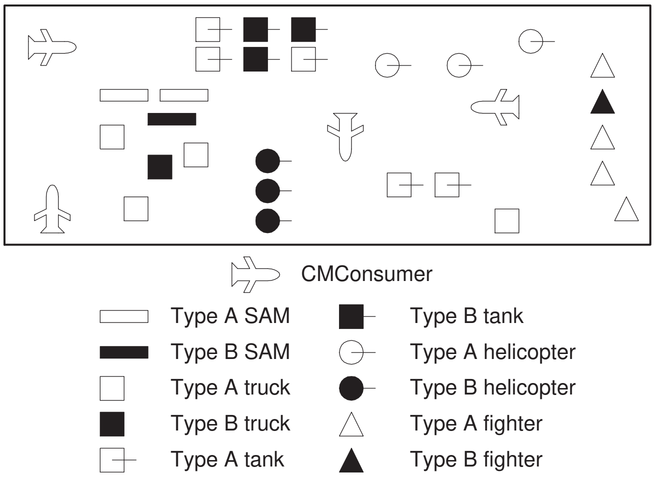

Figure 6: CMConsumer agents traversing a geospatial region in a simulated aerial reconnaissance system.

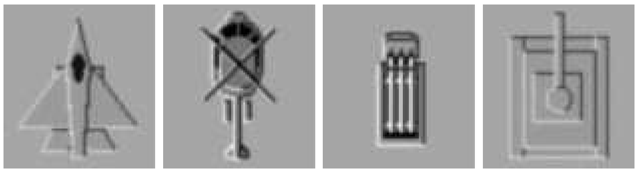

Figure 7: Sample reconnaissance images: Type A fighter, Type B helicopter, Type A SAM, and Type A tank

In this experiment, CMConsumers entered an *initialization stage* followed by an *execution stage*. The initialization stage was concerned with having CMConsumers acquire and sample various networks in the system in order to determine which network exhibited the highest classification performance. Sampling of networks during the initialization stage was carried out by having each CMConsumer perform image classification on 20 captured images for each network it acquired. Each CMConsumer then used its highest-performing network for carrying out image classification on 100 captured images during its execution stage.

In order to highlight the impact of network merging on agent performance, we divided the experiment into two phases: a *control* phase and a *test* phase. In the control phase of the experiment, each CMConsumer was restricted to using only those networks that already existed in the system. However, in the test phase of the experiment, if the highest performing network used by a CMConsumer during its initialization stage performed below some specified threshold, the CMConsumer initiated network merging in order to derive a composite network that embodied the classification mappings of the two highest-performing networks that it sampled during its initialization stage. This composite network was then used by the CMConsumer during its execution stage. In both phases of the experiment, the goal of each CMConsumer was to accurately classify each of the 100 images it captured during its execution stage.

Our simulated aerial reconnaissance system was implemented and tested under Solaris 2.6 using Java 2, the Jackal 3.1 KQML package (Cost *et al.* 1998) and the Matlab 5.3 API. This system was comprised of ten CMConsumers, two CMRepositories, and three CM-Producers for both phases of the experiment. In addition, a CMTrainer was used during the test phase of the experiment. In addition, three initial CMs were introduced to the system for classifying (1) `TANK` and `TRUCK` objects, (2) `HELICOPTER`, `FIGHTER` and `TRUCK` objects, and (3) `TANK`, `SAM` , and `HELICOPTER` objects. In this experiment, we defined the performance of a network used by a CMConsumer during its execution stage as a quadruple $p = \{K, V, U, r\}$ where K represented the set of correctly classified images, V represented the set of incorrectly classified images, U represented the set of unclassifiable images, and

$$r = \frac{\|K\|}{\|K\| + \|V\| + \|U\|}$$

represented the *classification rate* of the network. For n processed images, we computed the average classification rate

$$r_{avg} = \frac{\sum_{i=1}^{n} r_i}{n}$$

of each network used by a CMConsumer. The average classification rate of each network used by a CMConsumer during both phases of the experiment are shown in Table 1.

Average Classification Rates		
CMConsumer	*control*	*test*
1	0.70	0.89
2	0.56	0.95
3	0.57	0.85
4	0.55	0.94
5	0.56	0.88
6	0.52	0.87
7	0.54	0.87
8	0.50	0.90
9	0.62	0.92
10	0.62	0.89

Table 1: Average classification rates of networks used by CMConsumers during control and test phases.

Here, we see that the average classification rates of networks for each CMConsumer during the test phase of the experiment were greater than those during the control phase of the experiment. Thus, these preliminary results support the claim that the use of network merging can help to maintain the classification performance of agents.

Conclusions

This paper presented an extension to the CMT framework for facilitating network merging. The motivation for this extension was to allow agents to potentially improve their learning or classification performance by utilizing composite networks that embody the classification mappings from several distributed and dynamically selected source networks. Using a simulated aerial reconnaissance system, we conducted a simple experiment that showed how the use of network merging could help agents maintain their ability to classify images of military objects.

References

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

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

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

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

Finin, T.; Labrou, Y.; and Mayfield, J. 1997. KQML as an agent communication language. In Bradshaw, J., ed., *Software Agents*. Cambridge, MA: MIT Press.

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

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

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

Pratt, L. 1993. *Transferring Previously Learned Back-Propagation Neural Networks to New Learning Tasks.* Ph.D. Dissertation, Department of Computer Science, Rutgers University.

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

Quirolgico, S.; Canfield, K.; Finin, T.; and Smith, J. 1999. Communicating neural network knowledge between agents in a simulated aerial reconnaissance system. In *First International Symposium on Agent Systems and Applications*, 242–254.

Towell, G., and Shavlik, J. 1991. Extracting refined rules from knowledge-based neural networks. *Machine Learning*.

Weiβ, G. 1996. Adaptation and learning in multiagent systems: Some remarks and a bibliography. In Weiβ, G., and Sen, S., eds., *Adaptation and Learning in Multi-Agent Systems*, 1–21. Springer-Verlag.