NSF ITR Robot-PFC
Working Memory Toolkit Specification

David C. Noelle
Joshua L. Phillips
10/28/2004

General Information

This document describes the basic framework and interface for the working memory toolkit This working memory toolkit will consist of a set of classes and functions written in ANSI C++ which will then be used to implement a working memory system that is inspired by current, biologically-based models and theories of human cognition and working memory. This system will be able to take information about the environment (provided by the rest of the robot's subsystems) and discern if this information should or should not be maintained within a limited-capacity working memory store. More importantly, the system will be able to learn how to make these decisions (to store or not to store information) over time without explicit aid or programming. The system will instead rely on simple reward information, which is a function of task performance, to learn how to appropriately update the contents of the working memory.

Memory traces or *chunks* will be provided to the system via pointers to arbitrary C++ data structures. The system will be capable of storing a limited number of chunks in its working memory *store*. The capacity of this working memory store must be specified by the user. The user must also specify a function which encodes all of the relevant features within a chunk into a vector of real values. Another function should also be provided that encodes relevant features of the current system state and/or the environment in a similar manner (a vector of real values) and also provides a list of candidate chunks. The working memory system will then evaluate the utility of these various chunks given the current system state and/or the state of the environment. The user must also provide a function that handles memory management for a chunk after it is no longer needed by the working memory system (for example, freeing dynamic memory or caching extraneous chunk information.) Finally, the user must provide a function that provides instantaneous reward information (a real value). This reward information will be used by the working memory system to learn (after multiple training trials) when to update the contents of the working memory store in a task-appropriate manner.

WorkingMemory class

Description

The WorkingMemory class contains most of the interfaces that the user will need to make use of. This class essentially contains all of the working memory traces currently being maintained and also specifies all of the functions for accessing and updating the working memory store. A single instantiation of this class will create all of the instantiations of the other objects necessary for proper working memory function. Thus, the user will only need to provide the reward, state, and chunk evaluation functions mentioned above to this class and this class will create any other components required to process chunk information. However, the user will need to provide candidate chunks to the system and will use the Chunk class to accomplish this task.

Data Members

Visibility	Type	Name	Description
Private	Chunk*[]	working_memory_store	Array of chunks stored in working memory.
Private	int	number_of_chunks	Total size of the working memory (maximum number of chunks that can be stored in working_memory_store.)
Private	int	number_of_active_chunks	Number of chunks currently held in working memory.
Private	void*	state_data_structure	Pointer to the data structure containing the current state's information.
Private	ActorNetwork*	actor_network	Neural network architecture for the action selection network.
Private	CriticNetwork*	critic_network	Neural network architecture for the evaluation of the current situation.
Private	StateFeatureVector	state_features	Vector object that contains the processed state information.
Private	ChunkFeatureVector*[]	chunk_features	Vector objects that contain the processed chunk information.
Private	AggregateFeatureVector	aggregate_features	Vector object that can be processed by the actor and critic networks.
Private	double (*) (WorkingMemory&)	reward_function	Function for getting instantaneous reward information from the user.
Private	void (*) (FeatureVecto&*, WorkingMemory&)	translate_state	Function for translating the current state information into feature vector form.
Private	void (*)(FeatureVector&, Chunk&, WorkingMemory&)	translate_chunk	Function for translating chunks into feature vector form.

Visibility	Type	Name	Description
Private	void (*)(Chunk&)	delete_chunk	Function for deleting arbitrary chunk information before Chunk disposal.
Private	int	episode_time	Time step for the current episode.
Private	bool	use_actor	Flag specifying whether to use the actor network to choose actions.
Private	OR_CODE	the_or_code	Value specifying whether to use “OR” coding of the chunk feature vectors when processing states and chunks.
Private	double	last_reward	Value of reward on last time step.
Private	double	exploration_percentage	Percentage of the time that the WorkingMemory system performs a memory update or release that is contrary to the one advised by the critic network. (5% by default.)

Methods

Visibility	*Return Type*	*Name*	*Arguments*	*Description*
Public	None	WorkingMemory	wm_size : int state_feature_vector_size : int chunk_feature_vector_size : int user_state_data_structure : void* user_reward_function : double (*)(WorkingMemory&) user_state_function : void (*)(FeatureVector&, WorkingMemory&) user_chunk_function : void (*)(FeatureVector&, Chunk&, WorkingMemory&) user_delete_function: void (*) (Chunk&) using_actor : bool or_code : OR_CODE	Constructor The wm_size is the maximum number of chunks that can be stored in the working memory system. The state_feature_vector and chunk_feature_vector parameters are the size of the FeatureVectors that are needed by the user_state_function and user_chunk_function functions, respectively. The using_actor flag tells the working memory system whether to use an actor network for selecting chunks for storage. The using_or_code flag tells the working memory system whether to evaluate the “OR” of all chunks or just take them individually when selecting them for storage. Enumeration defined with FeatureVector class.
Public	None	~WorkingMemory	None	Destructor
Public	None	WorkingMemory	WorkingMemory&	Copy Constructor
Public	None	=	None	Assignment Operator
Public	int	getWorkingMemorySize	None	Returns the total number of chunks that the working memory system can hold at any one time.
Public	int	getNumberOfChunks	None	Returns the number of chunks currently stored in the working memory store.

Visibility	***Return Type***	***Name***	***Arguments***	***Description***
Public	Chunk	getChunk	chunk_number : int	Retrieves the specified chunk from working memory.
Public	void*	getStateDataStructure	None	Retrieves the user-provided data structure that contains all relevant information about the current state.
Public	bool	newEpisode	clear_memory : bool	Resets the working memory class's time-step clock to zero. It should be used at the end of every training “episode”. (It cues the working memory class that this is the final state of the episode.) A provided flag tells whether to clear all of the chunks out of working memory. This function returns true if the new episode could be properly initialized, false otherwise. (Reset the environment/state information **BEFORE** calling this function, as it initializes the WM system using the current state configuration as a start-state.)
Public	int	getEpisodeTime	None	Returns the current time step as recorded by the working memory clock. Clock time starts at zero.
Public	bool	checkForTickCall	tolerance : double	This function makes a comparison between the current state representation and the one for the last time step to see if any significant changes have occurred. If there has been a significant change then it returns true, false otherwise. The comparison is made by calculating the Euclidean distance between the two state vectors. The provided tolerance will be used to gage the difference being significant. This function can be used to gage when to call tickEpisodeClock (it might not need to be called if this function returns false.)
Public	int	tickEpisodeClock	candidate_chunks : list<Chunk>&	This function performs several different critical functions. First, it calls the user-provided state feature extraction function, then it calls the user-provided chunk feature extraction function for each chunk in the provided list, and, finally, the actor-critic networks are consulted and relevant chunks are added to the working memory store. This essentially is the function that performs all of the learning and memory updating. It then increments the working memory clock and returns the new clock value. The clock will not increment if there was a problem while processing working memory.

Visibility	***Return Type***	***Name***	***Arguments***	***Description***
Public	bool	isUsingActor	None	Checks to see if the WorkingMemory class is using an actor network to process chunks.
Public	OR_CODE	getORCode	None	Returns the OR_CODE value corresonding to the OR_CODE the WorkingMemory class is using.
Public	bool	setORCode	or_code : OR_CODE	Sets the OR_CODE value for the WorkingMemory class.
Public	bool	saveNetwork	filename : string	Writes the connection weight values of the neural networks to the file, filename.
Public	bool	loadNetwork	filename : string	Reads the connection weight values for the neural networks from the file, filename.
Public	double	getEplorationPercentage	None	Returns the current memory updating exploration percentage being used by the system.
Public	bool	setExplorationPercentage	val : double	Sets the current memory exploration percentage. Returns true on success and false otherwise.
Public	CriticNetwork*	getCriticNetwork	None	Returns a pointer to the CriticNetwork object that the WorkingMemory object has created for learning the value of its working memory contents.
Public	ActorNetwork*	getActorNetwork	None	Returns a pointer to the ActorNetwork object that the WorkingMemory object has created for learning when to add a chunk to the working memory system.
Private	bool	generateChunkFeatureVectors	None	Takes the current contents of working memory as resides in the working_memory_store and generates the feature vector representations (storing them in chunk_features.)

Possible user-provided function prototypes:

```
double reward_function(WorkingMemory&);
void state_translation_function(FeatureVector&, WorkingMemory&);
void chunk_translation_function(FeatureVector&, Chunk&, WorkingMemory&);
void chunk_deletion_function(Chunk&);
```

Notes on function prototypes:

These can be renamed to anything desired, but the return type and argument lists should remain as specified. The WorkingMemory& in all functions will be the working memory system itself. Thus, any state and chunk information in the WorkingMemory class can be accessed within the function via this pointer. For the state and chunk translation functions, the FeatureVector& is a vector of the appropriate size as provided to the Working Memory class constructor. Simply fill in the vector with the appropriate values and it will be used by the working memory system when the function returns. The Chunk& in the chunk translation function is the actual chunk that needs to be translated. The chunk deletion function should properly handle the deletion of any data held by a chunk that is passed to the function. This function will be called when the working memory system decides that a particular chunk is not needed in working memory. Thus, the chunk deletion function must manage the deletion of any dynamic memory or other necessary memory management tasks. Any relevant type information should be provided to all chunks passed to the system so that it will be available within the chunk translation function when the WorkingMemory class calls these functions.

Chunk class

Description

This class holds onto a particular memory trace or chunk so that it can be transferred in and out of the WorkingMemory class's store. An informal amount of type information can be associated with the chunk in order to later identify the type of data it represents. This information is stored in a simple string format to be used in any way the user desires. A list of Chunk class objects will need to be made by the user and sent to the WorkingMemory class when calling tickEpisodeClock (as shown above).

Data Members

Visibility	***Type***	***Name***	***Description***
Private	void*	chunk_data	Pointer to chunk data.
Private	string	chunk_info	String tag containing some kind of user-specified type information.

Methods

Visibility	***Return Type***	***Name***	***Arguments***	***Description***
Public	None	Chunk	data : void* type : string	Constructor Creates a chunk class with the specified data and type information.
Public	None	Chunk	Chunk&	Copy Constructor
Public	None	=	None	Assignment Operator
Public	void*	getData	None	Returns a pointer to the chunk data.
Public	void	setData	data : void*	Set the current data pointer for this chunk class.
Public	string	getType	None	Returns the type string provided to the chunk class.
Public	void	setType	type : string	Sets the type string for the chunk.

CriticNetwork class

Description

This class holds all of the details of the critic network which will be used to approximate the value of storing or removing chunks from working memory. It is the workhorse of a TD learning system. It may or may not be used in conjunction with an ActorNetwork to learn how to update the contents of working memory. Using only a critic network will allow faster learning, but it will lead to slower working memory updates (this will probably not be an issue.)

Data Members

Visibility	***Type***	***Name***	***Description***
Private	Layer*	input_layer	This is a layer of neural units that will be provided with the representation from an AggregateFeatureVector.
Private	TDLayer*	critic_layer	This is a layer containing a single neural unit which will output the value of the AggregateFeartureVector representation within the input_layer and it also evaluates the change in expected reward (TD error) that occurs between two AggregateFeatureVectors (one at each time step.)
Private	Layer*	bias_layer	This is a layer containing a single neural unit whose activation will always be 1. Any weights leading from this layer to another will be the bias weights for the receiving units.
Private	TDFullForwardProjection*	input_to_critic_projection	This is a set of connection weights from the input_layer to the critic_layer that fully connects the two layers. These weights will be adjusted according to the TD error computed by the critic_layer.
Private	TDFullForwardProjection*	bias_critic_projection	This projection contains the bias weight for the critic unit.
Private	SimpleLinearActivationFunction*	slaf	This is the activation function that will be used by the units in the network.

Methods

Visibility	***Return Type***	***Name***	***Arguments***	***Description***
Public	None	CriticNetwork	input_layer_size : int	Constructor Creates a critic network with an input layer of the specified size.

Visibility	*Return Type*	*Name*	*Arguments*	*Description*
Public	None	~CriticNetwork	None	Destructor
Public	None	CriticNetwork	CriticNetwork&	Copy Constructor
Public	None	=	None	Assignment Operator
Public	TDLayer*	getCriticLayer	None	Returns a pointer to the critic_layer. (This is used for creating the actor network, which needs to have access to the critic_layer to process the TD error.)
Public	bool	clearEligibilityTraces	None	Clears the eligibility traces for the critic network.
Public	bool	initializeWeights	rng : RandomNumber Generator&	Initializes the projection with weight values as specified by the RandomNumberGenerator.
Public	double	processVector	features : FeatureVector&	Processes the vector representation through the network, then returns the value of the processed vector.
Public	double	processVectorAsNext TimeStep	features : FeatureVector& reward_for_curr ent_time_step : double	Processes the given vector through the network as the next state. The last time step's representation is the same as that used in the last processVector or processVectorAsNextTimeStep call. The reward information for the current state must be provided as well. Then the function returns the value of the current vector.
Public	double	processFinalTimeSte p	reward_for_curr ent_time_step : double	Uses the last vector processed via processVectorAsNextTimeStep (preferably) as the final state. This case absorbs the provided reward information. The clearEligibilityTraces method should normally be used immediately after calling this method. (However, this choice is left to the network user.)
Public	bool	writeWeights	file_stream : ofstream&	Writes the weights of the network out to the provided file stream. The stream should be ready to receive the values for writing. The function only returns false if the network is not yet initialized.
Public	bool	readWeights	file_stream : ifstream&	Reads the weights of the network in from the provided file stream. The stream should be ready for reading. The vector sizes of the network weights being used should be the same as this network's vectors. The function only returns false if the network is not yet initialized.
Public	bool	setLearningRate	value : double	Sets the learning rate for the network.
Public	double	getLearningRate	None	Returns the value of the learning rate being used by the network.

Visibility	*Return Type*	*Name*	*Arguments*	*Description*
Public	bool	setLearningRate	value : double	Sets the learning rate for the network. Returns false on failure, true otherwise.
Public	double	getGamma	None	Returns the current reward discount rate (gamma) used by the network.
Public	bool	setGamma	value : double	Sets the reward discount rate (gamma) for the network. Returns false on failure, true otherwise.
Public	double	getLambda	None	Returns the eligibility trace discount rate (lambda) used by the network.
Public	bool	setLambda	value : double	Sets the eligibility trace discount rate (lambda) for the network. Returns false on failure, true otherwise.

ActorNetwork class

Description

This class holds all of the details of the actor network which may or may not be used by the working memory system. It will slow learning to use this system, but results in faster performance once the system is trained-up.

Data Members

Visibility	***Type***	***Name***	***Description***
Private	Layer*	input_layer	This is a layer of neural units that will be provided with the representation from a FeatureVector.
Private	Layer*	output_layer	This is a layer of neural units that will calculate the probabilities of adding or not adding various chunks to the working memory store.
Private	TDFullForwardProjection*	input_to_output_projection	This is a set of connection weights from the input_layer to the output_layer that fully connects the two layers. These weights will be adjusted according to the TD error computed by the output_layer.
Private	int	winning_unit	The last action unit to win probabilistic selection.

Methods

Visibility	***Return Type***	***Name***	***Arguments***	***Description***
Public	None	ActorNetwork	input_layer_size : int critic_layer : TDLayer*	Constructor Creates an actor network with an input layer of the specified size and the projection tied to a TDLayer that is able to provide the appropriate TD error.
Public	None	~ActorNetwork	None	Destructor
Public	None	ActorNetwork	ActorNetwork&	Copy Constructor
Public	None	=	None	Assignment Operator
Public	void	initializeWeights	rng : RandomNumberGenerator&	Initializes the projection with weight values as specified by the RandomNumberGenerator.
Public	int	processVector	features : FeatureVector&	Processes the vector representation through the network and probabilistically chooses the winning action unit and then returns its index.

Visibility	***Return Type***	***Name***	***Arguments***	***Description***
Public	void	updateWeights	None	Updates the weights of the network based on the last representation in place from calling processVector and the current TD error from the TDLayer provided to the constructor.
Public	bool	writeWeights	file_stream : ofstream&	Writes the weights of the network out to the provided file stream. The stream should be ready to receive the values for writing. The function only returns false if the network is not yet initialized.
Public	bool	readWeights	file_stream : ifstream&	Reads the weights of the network in from the provided file stream. The stream should be ready for reading. The vector sizes of the network weights being used should be the same as this network's vectors. The function only returns false if the network is not yet initialized.

FeatureVector class

Description

Basic vector of real values that contains several helper functions for coding values into the vector easily. This is the the class for storing coded representations of a state or chunk.

Enumeration : OR_CODE

- NO_OR
- MAX_OR
- NOISY_OR

Data Members

Visibility	***Type***	***Name***	***Description***
Private	double*	values	Vector of real values for coding values.
Private	int	size	Size of the vector.

Methods

Visibility	***Return Type***	***Name***	***Arguments***	***Description***
Public	None	FeatureVector	vector_size : int	Constructor Creates a real vector of the specified size.
Public	None	~FeatureVector	None	Destructor
Public	None	FeatureVector	FeatureVector&	Copy Constructor
Public	None	=	None	Assignment Operator
Public	int	compare	FeatureVector&	Returns zero if the elements of the provided feature vector match this vector, negative if the provided vector is less than this vector, and positive if the provided vector is greater than this vector. (This comparison is only used for canonical ordering of feature vectors and has no mathematical relevance.)
Public	int	getSize	None	Returns the size of the vector.
Public	double	getValue	position : int	Returns the element of the vector in the specified position.
Public	bool	setValue	position : int value : double	Sets the vector element at the specified position.
Public	void	clearVector	None	Sets the vector elements to their minimum values.

Visibility	***Return Type***	***Name***	***Arguments***	***Description***
Public	bool	thermometerCode	start_position : int end_position : int minimum : double maximum : double value : double	This function encodes the value provided in thermometer code within the specified range of vector positions (inclusive) while scaling according to the minimum and maximum provided. The final individual vector values are in the range (0-1).
Public	bool	gaussianCoarseCode	start_position : int end_position : int minimum : double maximum : double variance : double value : double	This function encodes the value provided using a gaussian coarse coding of the specified variance within the specified range of vector positions (inclusive) while scaling according to the minimum and maximum provided. The final individual vector values are in the range (0-1].
Public	bool	speedometerCoarseCode	start_position : int end_position : int minimum : double maximum : double value : double	This function encodes the value provided in standard threshold coarse coding within the specified range of vector positions (inclusive) while scaling according to the minimum and maximum provided. The final vector values are either 0 or 1.
Public	bool	makeORCode	vector : const FeatureVector& type : const OR_CODE	Sets this vector to the “OR” of this feature vector with the provided feature vector. (The vectors must be the same size for this to succeed.)
Private	double	computeGaussian	mean : double variance : double value : double	This function computes the value of a gaussian function with the provided mean and variance.

StateFeatureVector class

Description

This class contains an array of real values that represent the current state of the world as provided by the sensory input systems. It inherits publicly from the FeatureVector class to do this.

Data Members

Visibility	*Type*	*Name*	*Description*
Private	void (*)(FeatureVector&, WorkingMemory&)	translate	Pointer to state translation function.

Methods

Visibility	*Return Type*	*Name*	*Arguments*	*Description*
Public	None	StateFeatureVector	vector_size : int translation_function : void (*)(FeatureVector&, WorkingMemory&)	Constructor Creates a real vector of the specified size and links it with the provided function.
Public	None	StateFeatureVector	StateFeatureVector&	Copy Constructor
Public	None	=	None	Assignment Operator
Public	bool	updateFeatures	wm : WorkingMemory&	Calls the translation function to map the information about the current state into the vector. Returns true if successful, and false otherwise.
Public	bool	setTranslationFunction	translation_function : void (*)(FeatureVector&, WorkingMemory&)	Sets the translation function to the one provided as an argument. Returns true on success and false otherwise.

ChunkFeatureVector class

Description

This class contains an array of real values that represent the information contained in a chunk in real-vector form so that it can be combined with information about the current state. This information can then be evaluated by the critic network and used to make choices about how to maintain working memory via the actor network (or the critic network will do this alone.) This class inherits publicly from the FeatureVector class.

Data Members

Visibility	***Type***	***Name***	***Description***
Private	void (*)(FeatureVector&, Chunk&, WorkingMemory&)	translate	Pointer to chunk translation function.

Methods

Visibility	***Return Type***	***Name***	***Arguments***	***Description***
Public	None	ChunkFeatureVector	vector_size : int translation_function : void (*) (FeatureVector&, Chunk&, WorkingMemory&)	Constructor Creates a real vector of the specified size and links it with the provided function.
Public	None	ChunkFeatureVector	ChunkFeatureVector&	Copy Constructor
Public	None	=	None	Assignment Operator
Public	bool	updateFeatures	single_chunk : Chunk& wm : WorkingMemory&	Calls the translation function to map the information about a chunk into the vector. Returns true if successful, and false otherwise.
Public	bool	setTranslationFunction	translation_function : void (*) (FeatureVector&, Chunk& WorkingMemory&)	Sets the translation function to the one provided as an argument. Returns true on success and false otherwise.

AggregateFeatureVector class

Description

This class allows for the concatenation of a StateFeatureVector with any number of ChunkFeatureVectors into a single vector representation that can then be provided to the actor and critic networks. It inherits publicly from the FeatureVector class.

Data Members

Visibility	***Type***	***Name***	***Description***
Private	int	s_vector_size	The size of the state vector that will be provided.
Private	int	c_vector_size	The size of the chunk vectors that will be provided.
Private	int	number_of_chunks	Number of chucks this vector assumes will be provided for concatenation.

Methods

Visibility	***Return Type***	***Name***	***Arguments***	***Description***
Public	None	AggregateFeatureVector	state_vector_size : int chunk_vector_size : int number_of_chunks_to_append : int	Constructor Creates a real vector of the specified dimensions.
Public	None	AggregateFeatureVector	AggregateFeatureVector&	Copy Constructor
Public	None	=	None	Assignment Operator
Public	bool	updateFeatures	state_vector : StateFeatureVector& chunk_vectors : ChunkFeatureVector*[]	Maps the feature vectors into the aggregated vector. Returns true if successful, and false otherwise.
Private	int	determineNecessarySize	state_size : int chunk_size : int num_chunks : int	Determines the size that an aggregate vector needs to be and checks for inconsistencies in the provided sizes.