

Machine Learning Explorer™ Eval Kit Specification

INSTALLATION

To install, unzip mle.zip with the option turned on to reserve the subdirectory structure. Mle.h should appear in FreeMLE/Include and Mle.lib should appear in FreeMLE/Lib. A demo application is available in FreeMLE/Demo.

The demo can be compiled with Microsoft® Visual C++ 5.0. After the project file is loaded, the user should perform the following to set up the directories for include and library files.

- Go to Tools->Options, select the directories tab. Select the “Include files” under “Show directories for:”, add “FreeMLE/Include”.
- Go to Tools->Options, select the directories tab. Select the “Library files” under “Show directories for:”, add “FreeMLE/Lib”.

Select Build->Build Demo.exe and the demo is done.

To appreciate the demo, under the menu item “Learn”, select “Explore” first, then “Associate”, and then “Optimize”. The Boxer on the right hand side of the drawing will learn to hit its opponent as efficiently as possible.

EVAL KIT OVERVIEW

Machine Learning Explorer™ Eval Kit includes the key component of Machine Learning Explorer™ SDK, the MLEngine. MLEngine is a general purpose AI engine suited for controlling simulated autonomous systems with discrete dynamics. The sensory states of the system under MLEngine’s control should be expressed as a linear array of discrete states ranging from 0 to $\text{dim_state} - 1$. Similarly, the actuator commands should be expressed as an array of discrete states ranging from 0 to $\text{dim_action} - 1$.

For the Eval Kit, the maximum number of sensory states allowed is 32. The maximum number of action states is 8.

INTRODUCTION

Machine Learning Explorer™ (MLE) is a set of algorithms designed to control simulated autonomous systems (automata, agents etc.). MLE will synthesize, in real time, a feedback controller that will drive the system under its control towards achieving a pre-defined goal. As a result, an integrated system with MLE built-in will appear to learn. The autonomous system (called the system thereafter) under the control of MLE needs to be equipped with a number of simulate sensors and actuators. The system should also interact with a dynamic environment through exchange of stimuli. MLE performs the control by taking sensory feedback and issuing commands to the actuators.

MLE combines sensory encoding, system identification and optimal control into one integrated solution. MLE is non-model based, which means it does not need to have any prior knowledge of the system under its control. As an integral part of the learning process, MLE will perform statistical system identification before it is able to synthesize an optimal controller. For system identification, MLE relies on information gathered from the sensors and records of its actuator commands. MLE is therefore a general purpose AI engine that can be applied to a wide variety of control applications.

To specify a goal for the integrated system, the application can take a subset of the system sensory channels and define the goal in terms of desired sensor values or sensory expectations. For example, if the objective of a robot is to grab an object with its end effector, the goal can be defined as a sensory state such that the contact sensor on the end effector signals a touch input.

The fact that a goal can only be expressed as sensory expectations implies that in order for a goal to be achieved, the system has to be equipped with sensory capabilities to tell whether it is in a state that corresponds the goal or not.

THE LEARNING PROCESS

Once MLE is integrated with an autonomous system, MLE carries out the following steps to complete the learning process. During each phase, an indicator will be available. The indicator can be used to tell to what extent the process has reached its intended objective.

Explore

During this phase, MLE will drive the system to interact freely with its environment. MLE essentially passes out random action command and monitors/remembers the sensory input.

Associate

During this phase, MLE will associate sensory and action information and build an internal representation of the system under its control. This is essentially a non-model based system identification process.

A parameter called association entropy can be obtained from MLE during the association phase. This value should go down as the association phase is being carried out. The smaller this value is, the more deterministic MLE has become while building a representation of the system under its control.

Optimize

This is the phase where MLE will synthesize a strategy to control the system and drive it to achieve a user-defined goal. The strategy is essentially a sensory feedback controller that determines how the system should interact with its environment by reacting to sensory inputs.

A parameter called optimization entropy can be returned from the MLE during the optimization phase. This value should go down as the optimization phase is being carried out. The smaller this value is, the more deterministic the synthesized strategy is.

Machine Learning Explorer™ API

Class MLEngine

Synopsis

#include "Mle.h"

Description

This is the class that implements MLEngine - a general purpose AI Engine for games.

MLEngine

Synopsis

MLEngine (int dim_state, int dim_action, int order)

Description

This will create an instance of MLEngine. The calling program needs to specify the dimension of the sensory state space, i.e. the number of states in the sensory space through dim_state, and the dimension of action space, i.e. the number of actions through dim_action. Additionally, the caller needs to specify the order of this instance of MLEngine. The order number will determine how far MLEngine predicts ahead.

Synopsis

MLEngine (char *filename)

By calling this constructor, the calling program can instantiate a MLEngine with parameters stored in a file written by MLEngine. See SaveMleState ().

~MLEngine

Synopsis

~MLEngine()

Description

This is the standard destructor for MLEngine. It deletes all the memory allocated for MLEngine.

SaveMleState

Synopsis

int SaveMleState(char *filename)

Description

The user program can call this function to save all the content of the current instance of MLEngine to a file specified by filename.

On success, SaveMleSate returns 0; otherwise, it returns -1.

Perform

Synopsis

```
int Perform (int s);
```

Description

When perform is called the application passes the sensory state *s* to MLEngine and MLEngine will evaluate the sensory input and return an actuator command. The command is an integer ranging from 0 to *dim_action* - 1. If the Engine has already gone through Optimization, the feedback strategy performed by Perform will drive the system under its control towards the maximum probability of achieving the state defined by the SetGoal.

Explore

Synopsis

```
int Explore(int s);
```

This function should be called repeatedly when the system explores its environment. By calling this function in each control loop, the sensory state information needs to be passed to MLEngine. MLEngine will return an action command to the system under its control. The result of the explore phase will be written to a temporary file.

Associate

Synopsis

```
void Associate();
```

Description

This function should be called repeatedly during the association phase. As a result, the Engine will perform system identification and build an internal representation of the system under its control. The extend to which the objective of association is accomplished can be monitored by calling GetAssociationEntropy ().

GetAssociationEntropy

Synopsis

```
double GetAssociateEntropy();
```

Description

By calling this function, the engine will return the current association entropy. The association entropy indicates to what extent the system identification process has completed. Without any system identification, the engine returns a value equal to 1. As the system identification is carried out, the association entropy will approach 0. In a lot of cases however, if the control system and its environment is not fully deterministic, the association may never get to 0. It will stabilize at some positive value between 0 and 1 instead.

SetGoal

Synopsis

```
void SetGoal(int goal);
```

Description

The caller can choose a sensory state and assign it as the goal. An integer that corresponds to goal states needs to be passed into the function.

Synopsis

```
void SetGoal(int* goal, int num_goals);
```

Description

The caller can choose one or more sensory states and assign them as the goal. An array of states goal needs to be passed into the function. The caller needs to specify the number of goals through num_goals as well.

Optimize

Synopsis

```
void Optimize();
```

Description

This function should be called repeatedly during the optimization phase. As a result, the Engine will optimize the feedback strategy so as to maximize the probability for the system to achieve the goal defined by SetGoal. The extent to which the optimization phase is finished can be monitored by calling GetOptimizationEntropy() and GetEstimatedGoalProbability().

GetOptimizeEntropy

Synopsis

```
double GetOptimizeEntropy();
```

Description

By calling this function, the engine will return the current optimization entropy. The optimization entropy indicates to what extent the optimization process has completed. Without any optimization, the engine returns a value equal to 1. As the system identification is carried out, the optimization entropy will approach 0. In many cases however, if the optimal control strategy is one that is not fully deterministic, the optimization entropy may never get to 0. It will stabilize at some positive value between 0 and 1 instead.

GetEstimatedGoalProbability

Synopsis

```
double GetEstimatedGoalProbability();
```

Description

By calling this function, the engine will return the estimated long-term probability for the system to achieve the defined goal. As discussed earlier, MLEngine itself cannot guarantee that the control system will perform and accomplish the user defined the goal, it only optimizes the strategy so that the probability to achieve the goal is the best. It is possible that certain goals are hard to achieve because of many other constraints. MLEngine does give an estimation of the probability that the defined the goal can be achieved under the control of a MLEngine while it performs the optimization. As the optimization phase is carried out, the optimization entropy will decrease while the estimated goal probability will increase.

SetAssociationRate

Synopsis

```
void SetAssociationRate(double r)
```

Description

The parameter association rate determines the rate at which MLEngine converges during the association phase. The default is 0.1. The bigger the association rate, the faster MLEngine will converge initially. With big association rate on the other hand, MLEngine will be less stable. The association entropy will fluctuate more. When the association rate is small, MLE will be able to stabilize better when it has built an internal representation of the system under its control.

SetOptimizationRate

Synopsis

```
void SetOptimizationRate(double r)
```

Description

The parameter optimization rate determines the rate at which MLEngine converges during the optimization phase. The default is 0.1. The bigger the

optimization rate, the faster MLEngine will converge initially. With big optimization rate on the other hand, MLEngine will be less stable. The optimization entropy will fluctuate more. When the optimization rate is small, MLE will be able to stabilize better when it has synthesized a feedback strategy to achieve the goal set by SetGoal.

UnlearnAssociation

Synopsis

```
void UnlearnAssociation();
```

Description

Calling this function will erase all the information content stored in the MLEngine. This is useful when there is a need for the MLEngine to learn a new strategy under a new environment condition.

In a real application, after MLEngine has already learned a strategy, the way the system behaves and interacts with its environment may change. If learning continues as usual, MLEngine will take time to unlearn the old strategy and then go on to learn a new strategy. By calling unlearn under this kind of condition, the whole process becomes faster. Calling this function will also reset association and optimization entropy, making it easier to control the transition between phases.

To determine if there is a need to unlearn, the application can monitor the average goal probability of and compare that to the estimated goal probability. If the average drop far below the estimation, it is probably time to unlearn and learn again.

```
void UnlearnOptimization()
```

Calling this function will erase the optimized feedback strategy. This is useful when there is a new goal and therefore MLEngine needs to learn a new strategy.

Q&A

1. What is Machine Learning Explore, how is it different from other AI engines such as a hierarchical finite state machine (HFSM)?

Machine Learning Explorer (MLE) is an integrated solution that combines statistical system identification and optimal control. It is designed for controlling autonomous systems through interface with sensors and actuators. MLE does not assume any prior knowledge about the system under its control and synthesizes an optimal control strategy at run time.

Finite state machines need to be designed off-line. Whoever does that needs to understand the dynamics of the system under its control. MLE can synthesize control strategies at run-time.

2. Since MLE is adaptive and capable of self-learning, is there any relationship between MLE and artificial neural network (NN) models?

Yes, we took many inspirations from neural net research, especially those with recurrent architecture (e.g. Hopfield type NN model) and stochastic dynamics (e.g. Boltzmann Machine). The advantage of these architectures is such that they are "active" and capable of generating temporal sequence of motor commands by themselves. The output is a stochastic function of both input and time.

Our algorithm shares with neural net model at pure mathematical level. We did not simulate the dynamics of neurons and synapses. In other words, we took the mathematics that makes the neural net models work and optimize that for conventional microprocessors.

3. When you say that your algorithm learns, what exactly does it learn from? Do you have to teach it?

Once MLE is integrated with an autonomous system through sensor and actuator interface, the user can and specify a goal. This goal has to be a combination of desired sensor values. The learning process will synthesize a feedback strategy from sensors to actuators so as to maximize the probability for the system to achieve the state defined by the goal.

Although you have to let MLE know what its goal is, you do not have to "teach" MLE specific motor sequences to achieve that goal. MLE is an unsupervised learning machine. In fact, MLE can generate motor sequences that have never been tried out before during the explore phase. It is required however that MLE has tried all the basic elements of such sequences during the explore phase.

Because the goal or control objective has to be specified in terms of desired sensory values, the system can only learn to accomplish a state its sensors can discriminate. For example, a bird can never learn to catch a target if it cannot see the target (that is, not knowing where the target is through sensors).

6. How adaptive can a learned system becomes, is this quality related to the explore/training phase?

In order for a system that is under the control of MLE to develop an adaptive strategy, it has to have sensors that provide real time feedback about its interaction with its environment. It also has to be able to interact freely with its environment when it explores its environment.

During the explore phase, if the condition of the environment undergoes changes and the control system has the sensory capability to reflect that, an adaptive strategy will be synthesized. The synthesized strategy will ensure that the system adapt in real time later so as to maintain a certain level of performance in spite of environment changes.

On the other hand, if the environment undergoes novel changes that has never happened before, the control system performance will degrade gracefully if such changes can be detected by existing sensors. In some case when environment changes are both significant and unaccounted by the existing sensors, the control system has to learn again in order to improve. The application program can compare the average probability of achieving the goal with the estimated probability returned from MLE engine to determine if there is a need to relearn.