LUKE – learning underlying knowledge of experts – User Manual

T. Makino / M. Shiro

March 31, 2014

1 Introduction

LUKE is software [1] used to perform apprenticeship learning [2] for environment model parameters efficiently. Apprenticeship learning for environment model parameters is inverse reinforcement learning generalized into model parameter estimation. Users are required to give a sequence of actions supposedly performed by experts, a sequence of observed states, and some credible prior distributions. LUKE allows you to estimate the reward function, the transition matrix, and the observation matrix from them.

In standard reinforcement learning [4], the state of an environment is estimated based on rewards set externally as well as your own actions and the resultant rewards. Specifically, the model is updated so as to maximize the expected sum of rewards. In many cases, however, the reward function is unknown. Conversely, the problem of estimating the reward function from an observation sequence and an action sequence can be framed. This is inverse reinforcement learning [3]. Apprenticeship learning is a generalized problem of inverse reinforcement learning, and allows you to estimate all of the transition matrix, the observation matrix, and the reward function from a given observation sequence and action sequence. This assumes that a sequence of observed actions have been performed by experts familiar with the environment model.

This manual describes apprenticeship learning for environment model parameters, which is the basis of LUKE, and then describes how to install and use it.

2 Apprenticeship Learning for Environment Model Parameters

For details about apprenticeship learning, refer to the references [5, 2]. The minimum required definitions are described here.

The partially observable environment (POMDP) $\mathcal{P}$ is expressed by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{Z}, T, O, \mathbf{b}^{(0)}, R, \gamma \rangle$.

- $\mathcal{S}$ is a set of S states.
- $\mathcal{A}$ is a set of A actions.
- $\mathcal{Z}$ is a set of Z observations.
- $T = \langle T^{a_1}, \ldots, T^{a_A} \rangle$ is a state transition matrix of A matrixes with a size of $S \times S$, and the elements of $T^a = (t^a_{s's})$ or $t^a_{s's} = P(s'|s, a)$ indicates the probability that the action a performed in the state s results in the state s'.
- $O = \langle O^{a_1}, \ldots, O^{a_A} \rangle$ is an observation matrix of A matrixes with a size of $Z \times S$, and the elements of $O^a = (o^a_{zs})$ or $o^a_{zs} = P(z|s, a)$ indicates the probability that the observation z is obtained when the action a results in the state s.
- $R = \langle R^{a_1}, \ldots, R^{a_A} \rangle$ is a reward matrix of A matrixes with a size of $S \times S$, and the elements of $R^a = (r^a_{s's})$ or $r^a_{s's}$ indicates the reward value when the action a performed in the state s results in the state s'.
- $\mathbf{b}^{(0)}$ is the vector of the length S and indicates the probability distribution of the initial state.
- $\gamma \in [0, 1)$ is a discount rate.

In the POMDP, the purpose of the learning agent is to find the policy π that maximizes the expected value of the sum of discounted rewards $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^{\mathrm{T}} r_{a_t s_t}]$. A policy is the action probability distribution when the belief $\mathbf{b}$ is given. The belief here is the vector that indicates the current state probability distribution in the environment.

Let's see the POMDP parameterized with some unknown parameters θ or POMDP $\mathcal{P}_\theta = \langle \mathcal{S}, \mathcal{A}, \mathcal{Z}, T_\theta, O_\theta, \mathbf{b}^{(0)}_\theta, R_\theta, \gamma \rangle$. Assume that the learning agent only knows the prior distribution $p(\theta)$, but experts know the true environment parameter θ^*, and the optimum value $Q^*(\mathbf{b}, a)$ resulting from action on that environment $\mathcal{P}_{\theta^*}$.

Assume that experts select an action based on the soft-max policy with the optimum value. Thus, the following holds:

$$\pi^*(a|\mathbf{b}) = \frac{max_{n:a_n=a} \exp(\beta Q^*(\mathbf{b}, a))}{\sum_{a' \in A} \exp(\beta Q^*(\mathbf{b}, a'))} \tag{1}$$

β is the inverse temperature parameter to control the optimality of action selection.

Therefore, apprenticeship learning for environment models performed on LUKE can be formulated as the problem of finding the parameter $\hat{\theta}$ that maximizes its posterior distribution $p(\theta|D)$ (and the optimum policy for $\mathcal{P}_{\hat{\theta}}$) when the demonstration of L steps $D = (a^1, z^1, \ldots, a^L, z^L)$ by experts is given.

LUKE supports the following three operation modes:

- Demonstration creation mode (GEN): In this mode, you can enter the POMDP $\mathcal{P}$ with known parameters and then calculate the optimum solution $\pi_{\mathcal{P}}$ for the POMDP and output the demonstration D. This mode is originally unnecessary, but is used to check model or parameter settings before actual demonstrations are used.

- Environment learning mode (IOHMM): In this mode, you can enter the parameterized POMDP $\mathcal{P}_\theta$, the prior distribution $p(\theta)$, and the demonstration D resulting from selecting an appropriate action, and then calculate $\hat{\theta}$ without making an assumption about action selection. This mode does not use apprenticeship learning and corresponds to conventional environment learning.

- Apprenticeship learning mode (APPL): In this mode, you can enter the parameterized POMDP $\mathcal{P}_\theta$, the prior distribution $p(\theta)$, and the demonstration D by experts, and then calculate $\hat{\theta}$ on the assumption that experts take action based on the optimum solution for the environment.

3 Installation

3.1 Required systems and libraries

If you are running Ubuntu 13.10 or later, all the necessary systems and libraries other than CPLEX can be installed by running the following command:
`apt-get install python g++-4.8 libboost1.53-dev libtbb-dev libnlopt-dev graphviz`

- Linux 3.2 or later (distribution, 32- or 64-bit version)
  At present, Cygwin is not supported as it cannot compile intel tbb. It may run on Visual C++ on Windows (Until recently, it was running, but compilation can now no longer be performed because an internal error in Visual C++ is caused. It may be able to be run if a way of avoiding the problem is found).

- GCC version 4.8 or later (Pay attention to the required version.)
  To build gcc on your system, install the necessary software (libgmp-dev libmpc-dev libmpfr-dev liblibisl-dev libcloog-isl-dev), run the command `./configure --prefix=/usr/local --enable-languages=c,c++,lto` to create a config file, and then run the command `make -j12` or `sudo make install`. Specify the number of cores of the computer where compilation is performed for the option `j`.

- Boost version 1.53 or later (Pay attention to the required version.)
  If your Boost is not the latest version, run any one of the following commands to perform compilation:
  `./bootstrap.sh --prefix=/usr/local`
  `./b2 variant=debug install -j12`

`./b2 variant=release install -j12`
Specify the number of cores of the computer where compilation is performed for the option `j`. Boost is a template library, but LUKE cannot be run just by copying the header.

- Python
  This is necessary to run waf. Most systems have it installed as standard. If your system has not it installed, install it.

- Intel tbb
  Obtain the library from `http://threadingbuildingblocks.org/download` and install it.

- nlopt version 2.3 or later
  Obtain the library from `http://ab-initio.mit.edu/wiki/index.php/NLopt` and install it. To install it, execute `./configure` and then `make` and `make install` in this order.

- CPLEX: The current version of LUKE uses CPLEX from IBM to solve linear programming problems. If you are a staff member of an educational institution, you can use it free of charge by registering with the IBM Academic Initiative. This is used only to solve linear programming problems, and it should not be difficult to use free software instead of CPLEX. If CPLEX has not been installed, LUKE attempts to run without using it (but, this attempt has not been successful so far).

3.2 Recommended library

- graphviz
  This is a tool to visualize estimation results. The results of LUKE are given as a `.dot` file. For a Debian distribution, it can be installed by running the command `apt-get install graphviz`. Alternatively, obtain it from `http://www.graphviz.org/content/fsm` and then compile it. Use it by running a command like `dot -Tpng nodes.dot > results.png`.

3.3 Installing LUKE

- Download and expand the package [1].

- Compilation

  1. It is `trunk` in `pomdp`: `cd pomdp/trunk`
  2. If the C++ compiler and necessary libraries are not placed in the default locations, set the environment variables.
     `export CXX=/usr/local/bin/g++` (bash)
     `export LD_LIBRARY_PATH=/usr/local/lib:/usr/local/lib64` (bash)

[1] In the future, it will be distributed as a tarball, but currently, it is distributed through subversion. Obtain `svn checkout https://dav.snowelm.com:4443/pomdp/`.

3. Specify the location of each library as an argument and create a config file:
   `./waf configure`
4. Build them:
   `./waf release`

- Executing a sample:
  Run a command
  `build/release/src/bayesmain --model=tiger.model --expertmodel=experttiger.model --task=APPL_EM`. `--task=APPL_EM` means to perform apprenticeship learning. If graphviz has been installed, you can see the execution result by running the command `dot -Tpng nodes.dot > nodes.png`.

Many problems occur during the process of creating a config file. Especially, if the libraries are not placed in the default locations, their locations must be specified explicitly. You may be able to find the cause of a problem by checking `build/config.log` created.

- `--boost-libs`=Location of the Boost library
- `--boost-includes`=Location of the header file of Boost
- `--cplex`=Location of CPLEX
- `--tbb`=Location of TBB

The author created a config file as follows:
`./waf configure --cplex=/usr/local/lib/cplex --tbb=/usr/local/lib/tbb`

4 Setting and Execution

4.1 Execution and command line arguments

When compilation is performed successfully, an executable file named `build/release/src/bayesmain` is created. Then, create a model file according to the format described later, give `bayesmain` command line arguments, and execute it. Arguments are described below. Give arguments in the following format: `--argumentname=parameter`. Arguments may be given in any order.

4.1.1 Basic argument

Specify the operation mode by the argument `--task`.

- `LOAD_ONLY`: Only loads specified files. This mode is used to check syntax.
- `GEN_DEMO`: Creates time-series data based on a given expert model. Specify `--expertmodel` and `--demolength`, respectively. This mode is selected by default.

- `APPL_EM`: Performs apprenticeship learning.
- `IOHMM_EM`: Solves input/output hidden Markov models. This mode corresponds to the environment learning mode mentioned before.

4.1.2 Other arguments

Some of the`task`: A brief list of arguments and the default values can be shown by specifying `--help` as an argument.

- `--model`: Specify a template model file.
- `--demodata`: Specify a time-series data file of expert actions.
- `--algo`: Specify an algorithm used for optimization[2]. Specify a negative value to use gradient data. For details, refer to `http://ab-initio.mit.edu/wiki/index.php/NLopt_`

  0 Nelder-Mead Simplex method
  1 BOBYQA method
  2 COBYLA method
  3 Principal AXIS method
  4 Subplex-based method
  5 Controlled Random Search method
  6 Improved Stochastic Ranking Evolution Strategy method
  −1 Moving Asymptotes method
  −3 SLSQP method
  −4 Low-storage BFGS method
  −5 Preconditioned truncated Newton method
  −6 Shifted limited-memory variable-metric, Rank1
  −7 Shifted limited-memory variable-metric, Rank2

  1 or the BOBYQA method is selected by default.
- `--skiptest`: Specify whether to skip a model test before actual execution. Specify "1" to skip a model test.
- `--verbosity`: Specify output redundancy. Specify "2" to debug the model.
- `--test_length`: Specify the length of test time series for evaluating learning results after the completion of apprenticeship learning (1000000).
- `--beta`: Specify an inverse temperature parameter for the normalized exponential function in creating a model internally from expert data.

[2]The argument is passed internally to the NLopt library.

- `--stepsize`: Specify a step size by which parameter values are changed for optimization[3].
- `--xtimeout`: Specify a length of time given to the parameter solver for overall optimization. Specify the time in seconds. The default value is 86400 or 24 hours.
- `--xtol`: Specify a tolerance level in optimizing parameters. If this level is not exceeded, it is assumed that parameters were optimized.
- `--ftol`: Specify a relative tolerance for function values.
- `--ptimeout`: Specify a length of time given to the POMDP solver for a single internal iteration of optimization. Specify the time in seconds. The default value is 600 or 10 minutes. If the length of time specified here is too short, the POMDP solver may provide unstable solutions, preventing you from arriving at the optimum solution.
- `--ptol`: Specify a tolerance for values in the POMDP solver.
- `--delta`: Specify a width considered not to make a difference between probability values. The default value is 10^{-5}.
- `--seed`: Specify a seed for random numbers.
- `--transfer_solution`: If "1" is specified, the POMDP solver uses the previous solution to increase the processing speed.
- `--single`: Solves problems using single-precision real. This may reduce memory used, increasing the processing speed.
- `--ndemos`: Specify how many pieces of internally created time series should be output when expert data is given in model format.
- `--demo_length`: Specify a time series length when expert data is given in model format and converted internally into time-series data. The default value is 100.
- `--expertmode`: Specify an expert model (model obtained after sufficient learning). Do not specify a demo file.
- `--outfile`: Specify an output file. In `GEN_DEMO` mode, a demo file is output. If an output file is not specified, the results are output to the standard output. (NULL)

[3]The argument is passed internally to the NLopt library.

4.2 Writing a demo file

A demo file contains time-series data of expert actions and each line has the following structure. A blank line may be put between these lines. Characters following "`%`" up to the line end are considered as comments.

`Action, observation`

Actions and observations are written in the same notation as in model files.

4.3 Writing a model file

The syntax for model files is similar to MATLAB, but note that matrix indexes start at 0. A statement ends with "`;`." Characters following "`%`" up to the line end are commends. The complicated method of writing a model file is described here by taking an apprenticeship learning version of the tiger problem famous in the partially observable Markov model (POMDP) as an example.

The original tiger problem assumes the situation where two doors are in front of the agent's eyes and there is a room on the other side of each door. There is a tiger in one room, and money in another room. The agent does not know in which room a tiger is and can select an action to take from among three options: open the right door (`open_right`), open the left door (`open_left`), or listen to the sound from behind the door (`listen`). If the room the agent selected has a tiger, he is eaten by it. Thus, a negative reward is set. If the room the agent selected has money, a positive reward is set. If the agent selects to listen to the sound, a slightly negative reward is set. The agent optimizes actions so as to maximize the expected reward and learns to open the door of the room with no tiger.

In apprenticeship learning, actions of experts familiar with a problem model (states and state transition) and the true environment settings (the probability of the room having a tiger, the probability of mishearing the sound, the loss suffered when the door of the room with a tiger is opened, etc.) and the observations of the results of those actions are given as time-series data (demonstrations). Given below is a model file for this problem:

```
%Gives three options as an action set: open the left door, open the right door,
%and listen to the sound.
[open_left, open_right, listen] = ActionSet;

%Gives two possible states as a state set: a tiger may be in the left room or in
%the right room.
[tiger_left tiger_right] = StateSet;

%Gives two possible types of observation as an observation set: roars may come
%from behind the left door or the right door.
[grawl_left, grawl_right] = ObservationSet;

%Defines the parameter badreward and specifies that it follows a normal distribution.
```

```
badreward = Normal( -50, 50 );

%Defines the parameters InitProb, ListenLeft, and ListenRight and specifies that
%they follow a beta distribution.
InitProb = Beta( 3, 3 );
ListenLeft = Beta( 5, 3 );
ListenRight = Beta( 5, 3 );

%Init is a reserved word. Gives an array of initial states as a column vector.
Init = [InitProb ; 1-InitProb];

%Trans is a reserved word. Gives state transition matrixes. eye(2) is a 2x2 identity mat
Trans(:,:,open_left) = [Init Init ];
Trans(:,:,open_right) = [Init Init ];
Trans(:,:,listen)    = eye(2);

%Obser is a reserved word. Gives observation matrixes. ones(2,2) is a 2x2 matrix
%where all the elements are 1.
%allot is a reserved function to allot the remaining value equally to rem_.
Obser(:,:,open_left) = ones(2,2) * 0.5;
Obser(:,:,open_right) = 0.5;
Obser(:,:,listen)    = allot([ListenLeft 1-ListenRight ; PlaceHolder PlaceHolder] );

%Rewar is a reserved word. Gives reward matrixes to estimate.
Rewar(:, :, listen) = -1;
Rewar(:, tiger_left, open_left) = badreward;
Rewar(:, tiger_right, open_right) = badreward;
Rewar(:, tiger_right, open_left) = +10;
Rewar(:, tiger_left, open_right) = 10;

%Discount is a reserved word to give a discount rate.
Discount = 0.75;
```

Reserved words, variables, and built-in functions are listed and summarized below. There are three reserved words:

- `ActionSet`: Action set
- `StateSet`: State set
- `ObservationSet`: Observation set

These reserved words can be called twice or more to specify a two- or more dimensional space. Let's see the following example:
`[s11, s12, s13] = StateSet; [s21, s22, s23] = StateSet;`

In this example, states are expressed by a pair of values, and each set has three values. In other words, there are nine possible state values ($s11 + s21, s11 + s22, s11 + s23, s12 + s21, s12 + s22, s12 + s23, s13 + s21, s13 + s22, s13 + s23$). Actually, the values $0, \ldots, 8$ are assigned to them. States, actions, and observations may be referenced by the respective names defined here or the corresponding integers. The same applies to demonstration data files.
There are five reserved variable names:

- `Init`: Initial state $\mathbf{b}_0(s)$
- `Trans`: State transition matrix $p(s'|s, a)$
- `Obser`: Observation matrix $p(z|s, a)$
- `Rewar`: Reward matrix $R(s', s, a)$
- `Discount`: Discount rate γ

Especially, note that `Trans`, `Obser`, and `Rewar` are three-dimensional matrixes. LUKE allows you to create any matrix slices according to the notation of MATLAB. For example, `Obser(:,:,open_left)` represents a slice where the third element of `Obser` is `open_left` (overall matrix of the first and second elements). In addition, any variable name other than these reserved variable names can be defined and used by placing it in the left-hand side of `=`.
The following two probability distributions are used:

- `Normal(`μ`,`σ`)`: Normal distribution. This is a normal conjugate prior distribution and is often used as the initial value of a continuous variable.
- `Beta(`α`,`β`)`: Beta distribution. This is a binomial conjugate prior distribution and is often as the initial value of a random variable. A multinomial distribution can be expressed by combining beta distributions.

These must be assigned to a variable before they can be used. These are Bayesian probability distributions to express the uncertainty of an environment, and cannot be used to describe the probabilistic behavior of an environment. For example, the reward distribution of the results of a gamble cannot be expressed (you must divide the next state and assign different reward values to them).
A built-in function is mainly used as an auxiliary function to write a program briefly, and the following built-in functions are implemented:

- `allot(rem_)`: Equally allots probability. This function adjusts the value of the placeholder `PlaceHolder` (the short form _ may also be used) in order that the sum of each column of the matrix becomes 1. This calculation is performed on a column-by-column basis.

$$\text{allot} \begin{pmatrix} 0.3 & 0.2 & 0.5 \\ 0 & \_ & 2 \times \_ \\ \_ & \_ & 3 \times \_ \end{pmatrix} = \begin{pmatrix} 0.3 & 0.2 & 0.5 \\ 0 & 0.4 & 0.2 \\ 0.7 & 0.4 & 0.3 \end{pmatrix}$$

- `conv2(M, N)`: Returns the convolution of two given sparse matrixes M, N. By specifying a transition matrix for the state set S_1 in M and a transition matrix for the state set S_2 in N and performing convolution, the transition matrix for the state set $S_1 \times S_2$ can be obtained. State spaces that interact with each other only under special conditions can be easily described by performing the convolution of a matrix containing placeholders, making some changes, and then calling the function `allot`.
- `conv3(M, N)`: Returns the three-dimensional convolution of two given sparse matrixes M, N.
- `carteset(s1, s2)`: Creates the direct product of set 1 and set 2.
- `expand(m, v1, v2 ...)`: Shifts the matrix m by the distances specified for the vectors with an index v1, v2, ... and calculates the sum. If two vectors with an index are specified, the result is equal to the convolution of the matrix m and the matrix `temp(v1,v2) = 1;` (or `conv2(m, temp)`). This function is used to expand a reward function defined for subspace 1 in a direction toward subspace 2.

In addition, the following functions of MATLAB can be used:

- `eye(n)`: Returns an identity matrix with a size of $n \times n$.
- `ones(n1, ...)`: Returns a matrix with a specified size where all the elements are 1.
- `zeros(n1, ...)`: Returns a matrix with a specified size where all the elements are 0.
- `length(v)`: Returns a vector length.
- `size(m)`: Returns the size of a matrix as a vector.
- `horzcat(m, n)`: Connects matrixes in a horizontal direction. You may also type "`[m, n]`" or "`[m n]`."
- `vertcat(m, n)`: Connects matrixes in a vertical direction. You may also type "`[m; n]`."
- `cat3(m, n)`: Connects matrixes in a three-dimensional direction.
- `display(x)`: Shows the content of x.

4.4 Output format

When execution is finished, a nodes.dot file is created. By processing this file with graphviz, the results can be obtained as a PNG image or other forms. The execution log is output to a file specified by `--outfile` or to the standard output. A list of settings, sample values initialized with random numbers, and

the given model are output first, and then `Solve` and `Calculate FIB` are output and followed by a state.

```
@Iteration Time(s) #Policy #Belief #BelHash #node/#free UpperBound LowerBound     Gap
@       16  15.72       2     481   59328 59330/    0    -16.845    -17.069    0.22405
@       32  23.63       2     581   70732 70734/    0    -16.848    -17.069    0.22115
@       48  23.72       2     574   70133 70135/    0    -16.848    -17.069    0.22115
@       59  28.56       2     493   70133 70216/    0    -17.025    -17.069   0.044227
@       60  32.92       2     493   70133 70216/    0    -17.031    -17.069   0.037939
```

In addition, the state log likelihood is shown in the form of `Finish solve, loglike=-***` and these are repeated.

5 Contact Address

Takaki Makino `t-luke@snowelm.com`

Acknowledgements

This research was subsidized by the Grants-in-Aid for Scientific Research (25730128) and the Funding Program for World-Leading Innovative R&D on Science and Technology (FIRST, Aihara Innovative Mathematical Modelling Project) designed by the Council for Science and Technology Policy through the Japan Society for the Promotion of Science.

References

[1] Takaki Makino, Masanori Shiro, and Kazuyuki Aihara. Effcient model-parameter inverse reinforcement learning program with parametric modeling of partially observable environments. pages 2H1–1, 2014.

[2] Takaki Makino and Johane Takeuchi. Apprenticeship learning for model parameters of partially observable environments. In *Proc. of the 29th International Conference on Machine Learning (ICML)*, pages 1495–1502, July 2012.

[3] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 663–670, San Francisco, CA, USA, 2000.

[4] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[5] , , and . . In *28*, pages 2H1–1. 2014.