LIBPG Quick API Reference, Sept 06

Controller

nil

void getAction(& obs, & action, computeGrad) = 0
void getMostProbAction (obs, action)
void getInputDim() = 0
void getNumParams() = 0
void discountTrace() = 0;
void setDiscount(double discount) = 0;
void accumulateGrad(reward, newObs) =0
void resetGrad() { NYI }
void computeDirection(int steps) { NYI };
void instantStep(Vector& reward) {NYI};
void batchStep() { NYI };
void setStepSize(double stepSize) = 0;
void resetTrace() = 0;
void resetParams() = 0;
void randomizeParams(maxRand) { NYI }
void getMaxParam() = 0;
void write(std::ostream& o) { NYI };
void read(std::istream& o) { NYI };
void reduce(Vec, Approximator::StatsEnum s) { NYI };
scatter(Vect, Approximator::StatsEnum s)

Approximator

int outputs (#action or #parameter)
int inputs; (dim of observation vector)
enum StatsEnum { PARAMS, GRADS }

int getNumParams() = 0;
void doApprox(Observation, Vec) = 0;
void feedbackGrad(Observation, Vec) = 0;
void discountTrace() = 0;
void setDiscount(double discount) = 0;
void instantStep(double reward) { NYI };
void setStepSize(double stepSize) { NYI };
void resetTrace() = 0;
void resetParams() = 0;
void randomizeParams(double maxRand) { NYI };
double getMaxParam() = 0;
void write(std::ostream& o) { NYI };
void read(std::istream& o) { NYI };
void batchStep() { NYI };
void accumulateGrad(double reward, Observation) { NYI };
void resetGrad() { NYI };
void computeDirection(int steps) { NYI };
void reduce(Vector& v, Approximator::StatsEnum s) { NYI };
void scatter(Vector& v, Approximator::StatsEnum s) { NYI };

BasicController

Approximator* approx
Vector dist

BasicController(Approximator* approx)
All virtual functions of Controller

BinaryController

nil

BinaryController(Approximator* approx);
getAction(& obs, Vec& action, bool computeGrad = true);
getMostProbAction(& obs, Vec& action);

FactoredController

typedef std::vector<Controller*> Controllers;
Controllers controllers;
bool splitObs;
Vector dummyAction;
Vector subReward;
bool localRewards;

FactoredController(Controllers controllers, bool splitObs, bool localRewards);
virtual ~FactoredController() {}
All virtual functions of **Controller**

LookupTable

int observations;
Matrix params;
Matrix trace;
double discount;

LookupTable(int obs, int outputs);
~LookupTable() {};
All virtual functions of **Approximator**

CyclicPolicyBias

int parameters;
Approximator* approx;
int stepsPerControl; // How many consecutive steps to add bias to a particular control
int cycleTime; // Total length of policy cycle
double bias; // How much to add to outputs

CyclicPolicyBias(Approximator*, int cycleTime, double bias);
All virtual functions of **Approximator**

Simulator

int outputs (#action or #parameter)
int inputs; (dim of observation vector)
enum StatsEnum { PARAMS, GRADS }

virt int getMaxEpisodeLength() {return 0;}
virtual int getObsRows() = 0;
virtual int getObsCols() = 0;
virtual int getAgents() = 0;
virtual int getActionDim() = 0;
virtual int getRewardDim() = 0;
virtual void getReward(Vec& rewards) = 0
virtual void getObservation(Obs& obs) = 0
virtual int doAction(Vector& action) = 0

Sampler

static int discrete(Vec& pdf, double cdf)

virt int getMaxEpisodeLength() {return 0;}
virtual int getObsRows() = 0;
virtual int getObsCols() = 0;
virtual int getAgents() = 0;
virtual int getActionDim() = 0;
virtual int getRewardDim() = 0;
virtual void getReward(Vec& rewards) = 0
virtual void getObservation(Obs& obs) = 0
virtual int doAction(Vector& action) = 0

Observation

Matrix features
Vector eligible
int agent
int steps

Observation(int rows, int cols, int agents)
Observation() {}; // Allow blank
void init(int rows, int cols, int agents);

RLAlg

Vector baseline;
bool useAutoBaseline;
double stepSize;
int maxSteps;
int epochsBetweenStepUpdates;
double threshold;
double increaseFactor;
double decreaseFactor;
bool doSaves;
char* saveName;
Controller* controller;
Simulator* simulator;
Observation obs;

RLAlg() { };
RLAlg(Controller* controller, Simulator* simulator, double discount, double stepSize);
virtual ~RLAlg() {};
virtual double doSteps(Vector& totalRewards, int steps, bool learn) = 0;
virtual double evaluate(Vector&totalRewards, int steps);
virtual double evaluateAndPrint(int stepsPerEpoch, int maxSteps);
virtual void resetLearning();
virtual double learn(int stepsPerEpoch, int maxTime=0, int maxSteps=0, double maxValue=0);
virtual double learnCore(int stepsPerEpoch, int& totalSteps);
virtual void saveBest(char* fname);
virtual void write(char* fname);
virtual void read(char* fname);
virtual void printPerformanceInfo(bool printTitles, int steps, double avgReward, double maxParam, int seconds);
virtual void checkStepSize(double curr, double best, double last);

OLPomdp

nil

OLPomdp() {};
OLPomdp(Controller* controller, Simulator* simulator, double discount, double stepSize);
double doSteps(Vector& totalReward, int steps, bool learn);

Controller

ManyToOneController

typedef std::vector<Approximator*> Approximators;
Approximators approximators;
bool splitObs //each controller get own obs
bool localRewards // reward per controller?
Vector dummyDist;
Vector dist;

ManyToOneController(Approximators approximators, bool splitObs, bool localRewards);
virtual ~ManyToOneController() {};
All virtual functions of **Controller**