meta$_{p}olicy_{s}earchDocumentation$

Dennis Lee, Ignasi Clavera, Jonas Rothfuss

Sep 06, 2019

Contents:

Despite recent progress, deep reinforcement learning (RL) still relies heavily on hand-crafted features and reward functions as well as engineered problem specific inductive bias. Meta-RL aims to forego such reliance by acquiring inductive bias in a data-driven manner. A particular instance of meta learning that has proven successful in RL is gradient-based meta-learning.

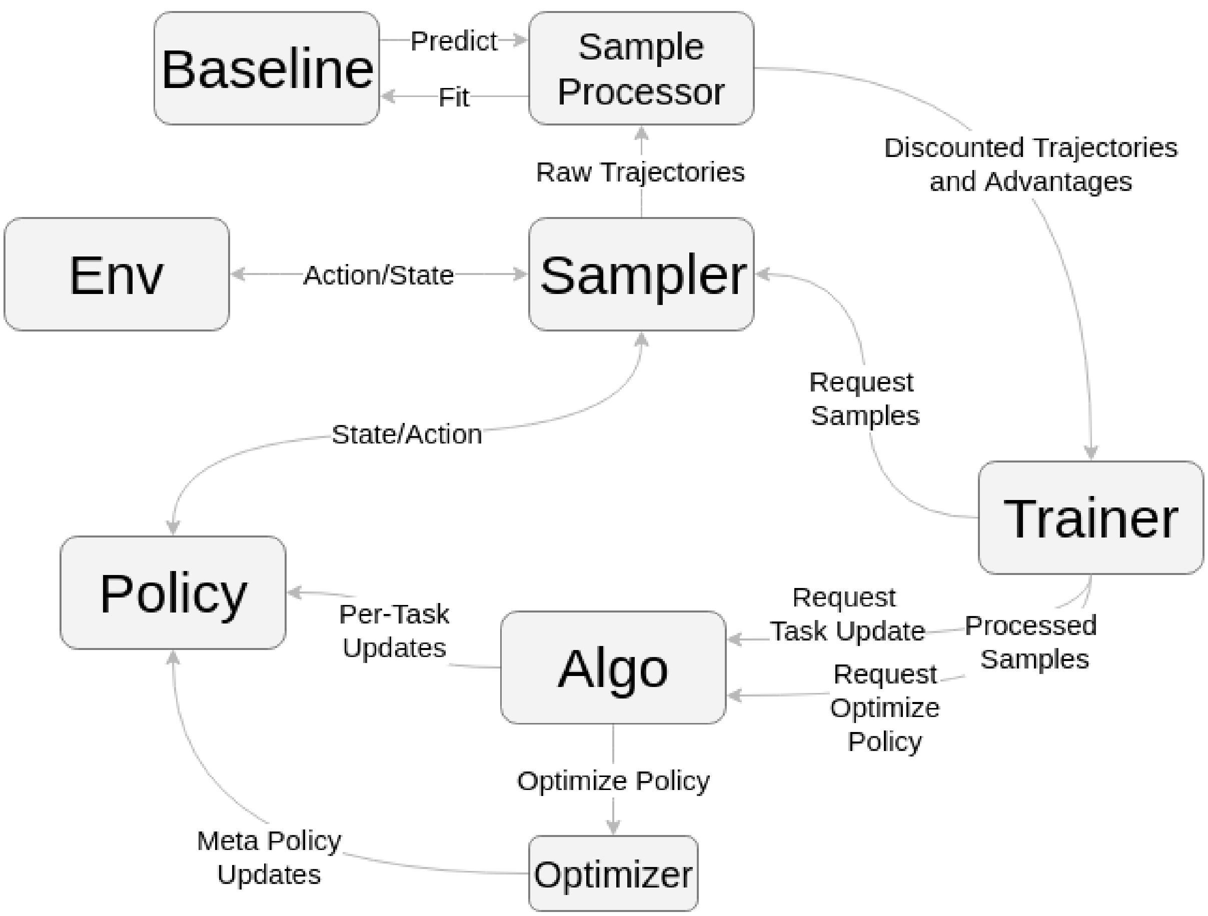
The code repository provides implementations of various gradient-based Meta-RL methods including

- ProMP: Proximal Meta-Policy Search (Rothfuss et al., 2018)
- MAML: Model Agnostic Meta-Learning (Finn et al., 2017)
- E-MAML: Exploration MAML (Al-Shedivat et al., 2018, Stadie et al., 2018)

The code was written as part of ProMP. Further information and experimental results can be found on our website. This documentation specifies the API and interaction of the algorithm's components. Overall, on iteration of gradient-based Meta-RL consists of the followings steps:

1. Sample trajectories with pre update policy
2. Perform gradient step for each task to obtain updated/adapted policy
3. Sample trajectories with the updated/adapted policy
4. Perform a meta-policy optimization step, changing the pre-updates policy parameters

This high level structure of the algorithm is implemented in the Meta-Trainer class. The overall structure and interaction of the code components is depicted in the following figure:



CHAPTER 1

Meta-Policy Search

1.1 Baselines

1.1.1 Baseline (Interface)

class meta_policy_search.baselines.**Baseline**

Reward baseline interface

fit(*paths*)

Fits the baseline model with the provided paths

Parameters paths – list of paths

get_param_values()

Returns the parameter values of the baseline object

log_diagnostics(*paths, prefix*)

Log extra information per iteration based on the collected paths

predict(*path*)

Predicts the reward baselines for a provided trajectory / path

Parameters path – dict of lists/numpy array containing trajectory / path information such as "observations", "rewards", …

Returns: numpy array of the same length as paths["observations"] specifying the reward baseline

set_params(*value*)

Sets the parameter values of the baseline object

Parameters value – parameter value to be set

1.1.2 Linear Feature Baseline

class meta_policy_search.baselines.**LinearFeatureBaseline**(*reg_coeff=1e-05*)

Linear (polynomial) time-state dependent return baseline model (see. Duan et al. 2016, "Benchmarking Deep

Reinforcement Learning for Continuous Control", ICML)

Fits the following linear model

reward = b0 + b1*obs + b2*obs^2 + b3*t + b4*t^2+ b5*t^3

Parameters **reg_coeff** – list of paths

fit (*paths*, *target_key='returns'*)
Fits the linear baseline model with the provided paths via damped least squares

Parameters

- **paths** (*list*) – list of paths
- **target_key** (*str*) – path dictionary key of the target that shall be fitted (e.g. "returns")

get_param_values (***tags*)
Returns the parameter values of the baseline object

Returns numpy array of linear_regression coefficients

log_diagnostics (*paths*, *prefix*)
Log extra information per iteration based on the collected paths

predict (*path*)
Abstract Class for the LinearFeatureBaseline and the LinearTimeBaseline Predicts the linear reward baselines estimates for a provided trajectory / path. If the baseline is not fitted - returns zero baseline

Parameters **path** (*dict*) – dict of lists/numpy array containing trajectory / path information such as "observations", "rewards", . . .

Returns numpy array of the same length as paths["observations"] specifying the reward baseline

Return type (np.ndarray)

set_params (*value*, ***tags*)
Sets the parameter values of the baseline object

Parameters **value** – numpy array of linear_regression coefficients

1.1.3 LinearTimeBaseline

class meta_policy_search.baselines.**LinearTimeBaseline** (*reg_coeff=1e-05*)
Linear (polynomial) time-dependent reward baseline model

Fits the following linear model

reward = b0 + b3*t + b4*t^2+ b5*t^3

Parameters **reg_coeff** – list of paths

fit (*paths*, *target_key='returns'*)
Fits the linear baseline model with the provided paths via damped least squares

Parameters

- **paths** (*list*) – list of paths
- **target_key** (*str*) – path dictionary key of the target that shall be fitted (e.g. "returns")

get_param_values (***tags*)
Returns the parameter values of the baseline object

Returns numpy array of linear_regression coefficients

log_diagnostics (*paths*, *prefix*)
Log extra information per iteration based on the collected paths

predict (*path*)
Abstract Class for the LinearFeatureBaseline and the LinearTimeBaseline Predicts the linear reward baselines estimates for a provided trajectory / path. If the baseline is not fitted - returns zero baseline

Parameters **path** (*dict*) – dict of lists/numpy array containing trajectory / path information such as "observations", "rewards", . . .

Returns numpy array of the same length as paths["observations"] specifying the reward baseline

Return type (np.ndarray)

set_params (*value*, ***tags*)
Sets the parameter values of the baseline object

Parameters **value** – numpy array of linear_regression coefficients

1.2 Environments

1.2.1 MetaEnv (Interface)

class meta_policy_search.envs.base.**MetaEnv** (**args*, ***kwargs*)
Wrapper around OpenAI gym environments, interface for meta learning

get_task ()
Gets the task that the agent is performing in the current environment

Returns task of the meta-learning environment

Return type task

log_diagnostics (*paths*, *prefix*)
Logs env-specific diagnostic information

Parameters

- **paths** (*list*) – list of all paths collected with this env during this iteration
- **prefix** (*str*) – prefix for logger

sample_tasks (*n_tasks*)
Samples task of the meta-environment

Parameters **n_tasks** (*int*) – number of different meta-tasks needed

Returns an (n_tasks) length list of tasks

Return type tasks (list)

set_task (*task*)
Sets the specified task to the current environment

Parameters **task** – task of the meta-learning environment

1.3 Meta-Algorithms

1.3.1 MAML-Algorithm (Interface)

class meta_policy_search.meta_algos.**MAMLAlgo**(*policy*, *inner_lr=0.1*, *meta_batch_size=20*, *num_inner_grad_steps=1*, *trainable_inner_step_size=False*)

Bases: meta_policy_search.meta_algos.base.MetaAlgo

Provides some implementations shared between all MAML algorithms

Parameters

- **policy** (Policy) – policy object
- **inner_lr** (*float*) – gradient step size used for inner step
- **meta_batch_size** (*int*) – number of meta-learning tasks
- **num_inner_grad_steps** (*int*) – number of gradient updates taken per maml iteration
- **trainable_inner_step_size** (*boolean*) – whether make the inner step size a trainable variable

build_graph()

Creates meta-learning computation graph

Pseudocode:

```
for task in meta_batch_size:
    make_vars
    init_dist_info_sym
for step in num_grad_steps:
    for task in meta_batch_size:
        make_vars
        update_dist_info_sym
set objectives for optimizer
```

make_vars(*prefix=”*)

Parameters **prefix** (*str*) – a string to prepend to the name of each variable

Returns a tuple containing lists of placeholders for each input type and meta task

Return type (tuple)

optimize_policy(*all_samples_data*, *log=True*)

Performs MAML outer step for each task

Parameters

- **all_samples_data** (*list*) – list of lists of lists of samples (each is a dict) split by gradient update and meta task
- **log** (*bool*) – whether to log statistics

Returns None

1.3.2 ProMP-Algorithm

class meta_policy_search.meta_algos.**ProMP**(**args, name='ppo_maml', learning_rate=0.001, num_ppo_steps=5, num_minibatches=1, clip_eps=0.2, target_inner_step=0.01, init_inner_kl_penalty=0.01, adaptive_inner_kl_penalty=True, anneal_factor=1.0, **kwargs*)

Bases: meta_policy_search.meta_algos.base.MAMLAlgo

ProMP Algorithm

Parameters

- **policy** (Policy) – policy object
- **name** (*str*) – tf variable scope
- **learning_rate** (*float*) – learning rate for optimizing the meta-objective
- **num_ppo_steps** (*int*) – number of ProMP steps (without re-sampling)
- **num_minibatches** (*int*) – number of minibatches for computing the ppo gradient steps
- **clip_eps** (*float*) – PPO clip range
- **target_inner_step** (*float*) – target inner kl divergence, used only when adaptive_inner_kl_penalty is true
- **init_inner_kl_penalty** (*float*) – initial penalty for inner kl
- **adaptive_inner_kl_penalty** (*bool*) – whether to used a fixed or adaptive kl penalty on inner gradient update
- **anneal_factor** (*float*) – multiplicative factor for annealing clip_eps. If anneal_factor < 1, clip_eps <- anneal_factor * clip_eps at each iteration
- **inner_lr** (*float*) – gradient step size used for inner step
- **meta_batch_size** (*int*) – number of meta-learning tasks
- **num_inner_grad_steps** (*int*) – number of gradient updates taken per maml iteration
- **trainable_inner_step_size** (*boolean*) – whether make the inner step size a trainable variable

build_graph()

Creates the computation graph

make_vars(*prefix="*)

Parameters **prefix** (*str*) – a string to prepend to the name of each variable

Returns a tuple containing lists of placeholders for each input type and meta task

Return type (tuple)

optimize_policy(*all_samples_data, log=True*)

Performs MAML outer step

Parameters

- **all_samples_data** (*list*) – list of lists of lists of samples (each is a dict) split by gradient update and meta task
- **log** (*bool*) – whether to log statistics

Returns None

1.3.3 TRPO-MAML-Algorithm

class meta_policy_search.meta_algos.**TRPOMAML**(**args, name='trpo_maml', step_size=0.01, inner_type='likelihood_ratio', exploration=False, **kwargs*)

Bases: `meta_policy_search.meta_algos.base.MAMLAlgo`

Algorithm for TRPO MAML

Parameters

- **policy** (`Policy`) – policy object
- **name** (*str*) – tf variable scope
- **step_size** (*int*) – trust region size for the meta policy optimization through TPRO
- **inner_type** (*str*) – One of 'log_likelihood', 'likelihood_ratio', 'dice', choose which inner update to use
- **exploration** (*bool*) – whether to use E-MAML or MAML
- **inner_lr** (*float*) – gradient step size used for inner step
- **meta_batch_size** (*int*) – number of meta-learning tasks
- **num_inner_grad_steps** (*int*) – number of gradient updates taken per maml iteration
- **trainable_inner_step_size** (*boolean*) – whether make the inner step size a trainable variable

build_graph()

Creates the computation graph

make_vars(*prefix=''*)

Parameters **prefix** (*str*) – a string to prepend to the name of each variable

Returns a tuple containing lists of placeholders for each input type and meta task

Return type (tuple)

optimize_policy(*all_samples_data, log=True*)

Performs MAML outer step

Parameters

- **all_samples_data** (*list*) – list of lists of lists of samples (each is a dict) split by gradient update and meta task
- **log** (*bool*) – whether to log statistics

Returns None

1.3.4 VPG-MAML-Algorithm

class meta_policy_search.meta_algos.**VPGMAML**(**args, name='vpg_maml', learning_rate=0.001, inner_type='likelihood_ratio', exploration=False, **kwargs*)

Bases: `meta_policy_search.meta_algos.base.MAMLAlgo`

Algorithm for PPO MAML

Parameters

- **policy** (Policy) – policy object
- **name** (*str*) – tf variable scope
- **learning_rate** (*float*) – learning rate for the meta-objective
- **exploration** (*bool*) – use exploration / pre-update sampling term / E-MAML term
- **inner_type** (*str*) – inner optimization objective - either log_likelihood or likelihood_ratio
- **inner_lr** (*float*) – gradient step size used for inner step
- **meta_batch_size** (*int*) – number of meta-learning tasks
- **num_inner_grad_steps** (*int*) – number of gradient updates taken per maml iteration
- **trainable_inner_step_size** (*boolean*) – whether make the inner step size a trainable variable

build_graph()

Creates the computation graph

make_vars(*prefix=''*)

Parameters **prefix** (*str*) – a string to prepend to the name of each variable

Returns a tuple containing lists of placeholders for each input type and meta task

Return type (tuple)

optimize_policy(*all_samples_data*, *log=True*)

Performs MAML outer step

Parameters

- **all_samples_data** (*list*) – list of lists of lists of samples (each is a dict) split by gradient update and meta task
- **log** (*bool*) – whether to log statistics

Returns None

1.4 Optimizers

1.4.1 Conjugate Gradient Optimizer

class meta_policy_search.optimizers.**ConjugateGradientOptimizer**(*cg_iters=10*, *reg_coeff=0*, *subsample_factor=1.0*, *backtrack_ratio=0.8*, *max_backtracks=15*, *debug_nan=False*, *accept_violation=False*, *hvp_approach=<meta_policy_search.op object>*)

Bases: `meta_policy_search.optimizers.base.Optimizer`

Performs constrained optimization via line search. The search direction is computed using a conjugate gradient algorithm, which gives x = A^{-1}g, where A is a second order approximation of the constraint and g is the gradient of the loss function.

Parameters

- **cg_iters** (*int*) – The number of conjugate gradients iterations used to calculate A^-1 g
- **reg_coeff** (*float*) – A small value so that A -> A + reg*I
- **subsample_factor** (*float*) – Subsampling factor to reduce samples when using "conjugate gradient. Since the computation time for the descent direction dominates, this can greatly reduce the overall computation time.
- **backtrack_ratio** (*float*) – ratio for decreasing the step size for the line search
- **max_backtracks** (*int*) – maximum number of backtracking iterations for the line search
- **debug_nan** (*bool*) – if set to True, NanGuard will be added to the compilation, and ipdb will be invoked when nan is detected
- **accept_violation** (*bool*) – whether to accept the descent step if it violates the line search condition after exhausting all backtracking budgets
- **hvp_approach** (*obj*) – Hessian vector product approach

build_graph(*loss*, *target*, *input_ph_dict*, *leq_constraint*)

Sets the objective function and target weights for the optimize function

Parameters

- **loss** (*tf_op*) – minimization objective
- **target** (Policy) – Policy whose values we are optimizing over
- **inputs** (*list*) – tuple of tf.placeholders for input data which may be subsampled. The first dimension corresponds to the number of data points
- **extra_inputs** (*list*) – tuple of tf.placeholders for hyperparameters (e.g. learning rate, if annealed)

- **leq_constraint** (*tuple*) – A constraint provided as a tuple (f, epsilon), of the form f(*inputs) <= epsilon.

constraint_val (*input_val_dict*)
Computes the value of the KL-divergence between pre-update policies for given inputs

Parameters

- **inputs** (*list*) – inputs needed to compute the inner KL
- **extra_inputs** (*list*) – additional inputs needed to compute the inner KL

Returns value of the loss

Return type (float)

gradient (*input_val_dict*)
Computes the gradient of the loss function

Parameters

- **inputs** (*list*) – inputs needed to compute the gradient
- **extra_inputs** (*list*) – additional inputs needed to compute the loss function

Returns flattened gradient

Return type (np.ndarray)

loss (*input_val_dict*)
Computes the value of the loss for given inputs

Parameters

- **inputs** (*list*) – inputs needed to compute the loss function
- **extra_inputs** (*list*) – additional inputs needed to compute the loss function

Returns value of the loss

Return type (float)

optimize (*input_val_dict*)
Carries out the optimization step

Parameters

- **inputs** (*list*) – inputs for the optimization
- **extra_inputs** (*list*) – extra inputs for the optimization
- **subsample_grouped_inputs** (*None or list*) – subsample data from each element of the list

1.4.2 MAML First Order Optimizer

class meta_policy_search.optimizers.**MAMLFirstOrderOptimizer**(*tf_optimizer_cls=<class 'tensorflow.python.training.adam.AdamOptimizer'>, tf_optimizer_args=None, learning_rate=0.001, max_epochs=1, tolerance=1e-06, num_minibatches=1, verbose=False*)

Bases: `meta_policy_search.optimizers.base.Optimizer`

Optimizer for first order methods (SGD, Adam)

Parameters

- **tf_optimizer_cls** (*tf.train.optimizer*) – desired tensorflow optimzier for training
- **tf_optimizer_args** (*dict or None*) – arguments for the optimizer
- **learning_rate** (*float*) – learning rate
- **max_epochs** – number of maximum epochs for training
- **tolerance** (*float*) – tolerance for early stopping. If the loss fucntion decreases less than the specified tolerance
- **an epoch, then the training stops.** (*after*) –
- **num_minibatches** (*int*) – number of mini-batches for performing the gradient step. The mini-batch size is
- **size//num_minibatches.** (*batch*) –
- **verbose** (*bool*) – Whether to log or not the optimization process

build_graph(*loss, target, input_ph_dict*)

Sets the objective function and target weights for the optimize function

Parameters

- **loss** (*tf_op*) – minimization objective
- **target** (Policy) – Policy whose values we are optimizing over
- **input_ph_dict** (*dict*) – dict containing the placeholders of the computation graph corresponding to loss

loss(*input_val_dict*)

Computes the value of the loss for given inputs

Parameters **input_val_dict** (*dict*) – dict containing the values to be fed into the computation graph

Returns value of the loss

Return type (float)

optimize(*input_val_dict*)

Carries out the optimization step

Parameters **input_val_dict** (*dict*) – dict containing the values to be fed into the computation graph

Returns (float) loss before optimization

1.5 Policies

1.5.1 Policy Interfaces

class meta_policy_search.policies.**Policy**(*obs_dim*, *action_dim*, *name='policy'*, *hidden_sizes=(32, 32)*, *learn_std=True*, *hidden_nonlinearity=<function tanh>*, *output_nonlinearity=None*, ***kwargs*)

Bases: meta_policy_search.utils.serializable.Serializable

A container for storing the current pre and post update policies Also provides functions for executing and updating policy parameters

Note: the preupdate policy is stored as tf.Variables, while the postupdate policy is stored in numpy arrays and executed through tf.placeholders

Parameters

- **obs_dim** (*int*) – dimensionality of the observation space -> specifies the input size of the policy
- **action_dim** (*int*) – dimensionality of the action space -> specifies the output size of the policy
- **name** (*str*) – Name used for scoping variables in policy
- **hidden_sizes** (*tuple*) – size of hidden layers of network
- **learn_std** (*bool*) – whether to learn variance of network output
- **hidden_nonlinearity** (*Operation*) – nonlinearity used between hidden layers of network
- **output_nonlinearity** (*Operation*) – nonlinearity used after the final layer of network

build_graph()

Builds computational graph for policy

distribution

Returns this policy's distribution

Returns this policy's distribution

Return type (Distribution)

distribution_info_keys(*obs*, *state_infos*)

Parameters

- **obs** (*placeholder*) – symbolic variable for observations
- **state_infos** (*dict*) – a dictionary of placeholders that contains information about the

- **of the policy at the time it received the observation** (*state*) –

Returns a dictionary of tf placeholders for the policy output distribution

Return type (dict)

distribution_info_sym(*obs_var*, *params=None*)
Return the symbolic distribution information about the actions.

Parameters

- **obs_var** (*placeholder*) – symbolic variable for observations
- **params** (*None or dict*) – a dictionary of placeholders that contains information about the
- **of the policy at the time it received the observation** (*state*) –

Returns a dictionary of tf placeholders for the policy output distribution

Return type (dict)

get_action(*observation*)
Runs a single observation through the specified policy

Parameters observation (*array*) – single observation

Returns array of arrays of actions for each env

Return type (array)

get_actions(*observations*)
Runs each set of observations through each task specific policy

Parameters observations (*array*) – array of arrays of observations generated by each task and env

Returns

array of arrays of actions for each env (meta_batch_size) x (batch_size) x (action_dim) and array of arrays of agent_info dicts

Return type (tuple)

get_param_values()
Gets a list of all the current weights in the network (in original code it is flattened, why?)

Returns list of values for parameters

Return type (list)

get_params()
Get the tf.Variables representing the trainable weights of the network (symbolic)

Returns a dict of all trainable Variables

Return type (dict)

likelihood_ratio_sym(*obs*, *action*, *dist_info_old*, *policy_params*)
Computes the likelihood p_new(obs|act)/p_old ratio between

Parameters

- **obs** (*tf.Tensor*) – symbolic variable for observations
- **action** (*tf.Tensor*) – symbolic variable for actions

- **dist_info_old** (*dict*) – dictionary of tf.placeholders with old policy information
- **policy_params** (*dict*) – dictionary of the policy parameters (each value is a tf.Tensor)

Returns likelihood ratio

Return type (tf.Tensor)

log_diagnostics (*paths*)
Log extra information per iteration based on the collected paths

log_likelihood_sym (*obs*, *action*, *policy_params*)
Computes the log likelihood p(obs|act)

Parameters

- **obs** (*tf.Tensor*) – symbolic variable for observations
- **action** (*tf.Tensor*) – symbolic variable for actions
- **policy_params** (*dict*) – dictionary of the policy parameters (each value is a tf.Tensor)

Returns log likelihood

Return type (tf.Tensor)

set_params (*policy_params*)
Sets the parameters for the graph

Parameters **policy_params** (*dict*) – of variable names and corresponding parameter values

class meta_policy_search.policies.**MetaPolicy** (**args*, ***kwargs*)
Bases: meta_policy_search.policies.base.Policy

build_graph ()
Also should create lists of variables and corresponding assign ops

distribution
Returns this policy's distribution

Returns this policy's distribution

Return type (Distribution)

distribution_info_keys (*obs*, *state_infos*)

Parameters

- **obs** (*placeholder*) – symbolic variable for observations
- **state_infos** (*dict*) – a dictionary of placeholders that contains information about the
- **of the policy at the time it received the observation** (*state*) –

Returns a dictionary of tf placeholders for the policy output distribution

Return type (dict)

distribution_info_sym (*obs_var*, *params=None*)
Return the symbolic distribution information about the actions.

Parameters

- **obs_var** (*placeholder*) – symbolic variable for observations
- **params** (*None or dict*) – a dictionary of placeholders that contains information about the
- **of the policy at the time it received the observation** (*state*) –

Returns a dictionary of tf placeholders for the policy output distribution

Return type (dict)

get_action(*observation*)
Runs a single observation through the specified policy

Parameters observation (*array*) – single observation

Returns array of arrays of actions for each env

Return type (array)

get_actions(*observations*)
Runs each set of observations through each task specific policy

Parameters observations (*array*) – array of arrays of observations generated by each task and env

Returns

array of arrays of actions for each env (meta_batch_size) x (batch_size) x (action_dim) and array of arrays of agent_info dicts

Return type (tuple)

get_param_values()
Gets a list of all the current weights in the network (in original code it is flattened, why?)

Returns list of values for parameters

Return type (list)

get_params()
Get the tf.Variables representing the trainable weights of the network (symbolic)

Returns a dict of all trainable Variables

Return type (dict)

likelihood_ratio_sym(*obs, action, dist_info_old, policy_params*)
Computes the likelihood p_new(obs|act)/p_old ratio between

Parameters

- **obs** (*tf.Tensor*) – symbolic variable for observations
- **action** (*tf.Tensor*) – symbolic variable for actions
- **dist_info_old** (*dict*) – dictionary of tf.placeholders with old policy information
- **policy_params** (*dict*) – dictionary of the policy parameters (each value is a tf.Tensor)

Returns likelihood ratio

Return type (tf.Tensor)

log_diagnostics(*paths*)
Log extra information per iteration based on the collected paths

log_likelihood_sym (*obs*, *action*, *policy_params*)
Computes the log likelihood p(obs|act)

Parameters

- **obs** (*tf.Tensor*) – symbolic variable for observations
- **action** (*tf.Tensor*) – symbolic variable for actions
- **policy_params** (*dict*) – dictionary of the policy parameters (each value is a tf.Tensor)

Returns log likelihood

Return type (tf.Tensor)

policies_params_feed_dict
returns fully prepared feed dict for feeding the currently saved policy parameter values into the lightweight policy graph

set_params (*policy_params*)
Sets the parameters for the graph

Parameters **policy_params** (*dict*) – of variable names and corresponding parameter values

switch_to_pre_update ()
Switches get_action to pre-update policy

update_task_parameters (*updated_policies_parameters*)

Parameters

- **updated_policies_parameters** (*list*) – List of size meta-batch size. Each contains a dict with the policies
- **as numpy arrays** (*parameters*) –

1.5.2 Gaussian-Policies

class meta_policy_search.policies.**GaussianMLPPolicy** (**args*, *init_std=1.0*, *min_std=1e-06*, ***kwargs*)
Bases: meta_policy_search.policies.base.Policy

Gaussian multi-layer perceptron policy (diagonal covariance matrix) Provides functions for executing and updating policy parameters A container for storing the current pre and post update policies

Parameters

- **obs_dim** (*int*) – dimensionality of the observation space -> specifies the input size of the policy
- **action_dim** (*int*) – dimensionality of the action space -> specifies the output size of the policy
- **name** (*str*) – name of the policy used as tf variable scope
- **hidden_sizes** (*tuple*) – tuple of integers specifying the hidden layer sizes of the MLP
- **hidden_nonlinearity** (*tf.op*) – nonlinearity function of the hidden layers
- **output_nonlinearity** (*tf.op or None*) – nonlinearity function of the output layer

- **learn_std** (*boolean*) – whether the standard_dev / variance is a trainable or fixed variable
- **init_std** (*float*) – initial policy standard deviation
- **min_std** (*float*) – minimal policy standard deviation

build_graph()
Builds computational graph for policy

distribution
Returns this policy's distribution

Returns this policy's distribution

Return type (Distribution)

distribution_info_keys(*obs*, *state_infos*)

Parameters

- **obs** (*placeholder*) – symbolic variable for observations
- **state_infos** (*dict*) – a dictionary of placeholders that contains information about the
- **of the policy at the time it received the observation** (*state*) –

Returns a dictionary of tf placeholders for the policy output distribution

Return type (dict)

distribution_info_sym(*obs_var*, *params=None*)
Return the symbolic distribution information about the actions.

Parameters

- **obs_var** (*placeholder*) – symbolic variable for observations
- **params** (*dict*) – a dictionary of placeholders or vars with the parameters of the MLP

Returns a dictionary of tf placeholders for the policy output distribution

Return type (dict)

get_action(*observation*)
Runs a single observation through the specified policy and samples an action

Parameters observation (*ndarray*) – single observation - shape: (obs_dim,)

Returns single action - shape: (action_dim,)

Return type (ndarray)

get_actions(*observations*)
Runs each set of observations through each task specific policy

Parameters observations (*ndarray*) – array of observations - shape: (batch_size, obs_dim)

Returns array of sampled actions - shape: (batch_size, action_dim)

Return type (ndarray)

get_param_values()
Gets a list of all the current weights in the network (in original code it is flattened, why?)

Returns list of values for parameters

Return type (list)

get_params()
Get the tf.Variables representing the trainable weights of the network (symbolic)

Returns a dict of all trainable Variables

Return type (dict)

likelihood_ratio_sym(*obs*, *action*, *dist_info_old*, *policy_params*)
Computes the likelihood p_new(obs|act)/p_old ratio between

Parameters

- **obs** (*tf.Tensor*) – symbolic variable for observations
- **action** (*tf.Tensor*) – symbolic variable for actions
- **dist_info_old** (*dict*) – dictionary of tf.placeholders with old policy information
- **policy_params** (*dict*) – dictionary of the policy parameters (each value is a tf.Tensor)

Returns likelihood ratio

Return type (tf.Tensor)

load_params(*policy_params*)

Parameters **policy_params** (*ndarray*) – array of policy parameters for each task

log_diagnostics(*paths*, *prefix=''*)
Log extra information per iteration based on the collected paths

log_likelihood_sym(*obs*, *action*, *policy_params*)
Computes the log likelihood p(obs|act)

Parameters

- **obs** (*tf.Tensor*) – symbolic variable for observations
- **action** (*tf.Tensor*) – symbolic variable for actions
- **policy_params** (*dict*) – dictionary of the policy parameters (each value is a tf.Tensor)

Returns log likelihood

Return type (tf.Tensor)

set_params(*policy_params*)
Sets the parameters for the graph

Parameters **policy_params** (*dict*) – of variable names and corresponding parameter values

class meta_policy_search.policies.**MetaGaussianMLPPolicy**(*meta_batch_size*, **args*, ***kwargs*)
Bases: meta_policy_search.policies.gaussian_mlp_policy.GaussianMLPPolicy, meta_policy_search.policies.base.MetaPolicy

build_graph()
Builds computational graph for policy

distribution
Returns this policy's distribution

Returns this policy's distribution

Return type (Distribution)

`distribution_info_keys`(*obs*, *state_infos*)

Parameters

- **`obs`** (*`placeholder`*) – symbolic variable for observations
- **`state_infos`** (*`dict`*) – a dictionary of placeholders that contains information about the
- **`of the policy at the time it received the observation`** (*`state`*) –

Returns a dictionary of tf placeholders for the policy output distribution

Return type (dict)

`distribution_info_sym`(*obs_var*, *params=None*)
Return the symbolic distribution information about the actions.

Parameters

- **`obs_var`** (*`placeholder`*) – symbolic variable for observations
- **`params`** (*`dict`*) – a dictionary of placeholders or vars with the parameters of the MLP

Returns a dictionary of tf placeholders for the policy output distribution

Return type (dict)

`get_action`(*observation*, *task=0*)
Runs a single observation through the specified policy and samples an action

Parameters **`observation`** (*`ndarray`*) – single observation - shape: (obs_dim,)

Returns single action - shape: (action_dim,)

Return type (ndarray)

`get_actions`(*observations*)

Parameters **`observations`** (*`list`*) – List of numpy arrays of shape (meta_batch_size, batch_size, obs_dim)

Returns A tuple containing a list of numpy arrays of action, and a list of list of dicts of agent infos

Return type (tuple)

`get_param_values`()
Gets a list of all the current weights in the network (in original code it is flattened, why?)

Returns list of values for parameters

Return type (list)

`get_params`()
Get the tf.Variables representing the trainable weights of the network (symbolic)

Returns a dict of all trainable Variables

Return type (dict)

`likelihood_ratio_sym`(*obs*, *action*, *dist_info_old*, *policy_params*)
Computes the likelihood p_new(obs|act)/p_old ratio between

Parameters

- **obs** (*tf.Tensor*) – symbolic variable for observations
- **action** (*tf.Tensor*) – symbolic variable for actions
- **dist_info_old** (*dict*) – dictionary of tf.placeholders with old policy information
- **policy_params** (*dict*) – dictionary of the policy parameters (each value is a tf.Tensor)

Returns likelihood ratio

Return type (tf.Tensor)

load_params (*policy_params*)

Parameters policy_params (*ndarray*) – array of policy parameters for each task

log_diagnostics (*paths*, *prefix=''*)
Log extra information per iteration based on the collected paths

log_likelihood_sym (*obs*, *action*, *policy_params*)
Computes the log likelihood p(obs|act)

Parameters

- **obs** (*tf.Tensor*) – symbolic variable for observations
- **action** (*tf.Tensor*) – symbolic variable for actions
- **policy_params** (*dict*) – dictionary of the policy parameters (each value is a tf.Tensor)

Returns log likelihood

Return type (tf.Tensor)

policies_params_feed_dict
returns fully prepared feed dict for feeding the currently saved policy parameter values into the lightweight policy graph

set_params (*policy_params*)
Sets the parameters for the graph

Parameters policy_params (*dict*) – of variable names and corresponding parameter values

switch_to_pre_update ()
Switches get_action to pre-update policy

update_task_parameters (*updated_policies_parameters*)

Parameters

- **updated_policies_parameters** (*list*) – List of size meta-batch size. Each contains a dict with the policies
- **as numpy arrays** (*parameters*) –

1.6 Samplers

1.6.1 Sampler

class meta_policy_search.samplers.**Sampler**(*env*, *policy*, *batch_size*, *max_path_length*)
Bases: object

Sampler interface

Parameters

- **env** (*gym.Env*) – environment object
- **policy** (*meta_policy_search.policies.policy*) – policy object
- **batch_size** (*int*) – number of trajectories per task
- **max_path_length** (*int*) – max number of steps per trajectory

obtain_samples()
Collect batch_size trajectories

Returns A list of paths.

Return type (list)

class meta_policy_search.samplers.**MetaSampler**(*env*, *policy*, *rollouts_per_meta_task*, *meta_batch_size*, *max_path_length*, *envs_per_task=None*, *parallel=False*)
Bases: meta_policy_search.samplers.base.Sampler

Sampler for Meta-RL

Parameters

- **env** (meta_policy_search.envs.base.MetaEnv) – environment object
- **policy** (*meta_policy_search.policies.base.Policy*) – policy object
- **batch_size** (*int*) – number of trajectories per task
- **meta_batch_size** (*int*) – number of meta tasks
- **max_path_length** (*int*) – max number of steps per trajectory
- **envs_per_task** (*int*) – number of envs to run vectorized for each task (influences the memory usage)

obtain_samples(*log=False*, *log_prefix=''*)
Collect batch_size trajectories from each task

Parameters

- **log** (*boolean*) – whether to log sampling times
- **log_prefix** (*str*) – prefix for logger

Returns A dict of paths of size [meta_batch_size] x (batch_size) x [5] x (max_path_length)

Return type (dict)

update_tasks()
Samples a new goal for each meta task

1.6.2 Sample Processor

class meta_policy_search.samplers.**SampleProcessor**(*baseline*, *discount=0.99*, *gae_lambda=1*, *normalize_adv=False*, *positive_adv=False*)

Bases: `object`

Sample processor interface

- fits a reward baseline (use zero baseline to skip this step)
- performs Generalized Advantage Estimation to provide advantages (see Schulman et al. 2015 - https://arxiv.org/abs/1506.02438)

Parameters

- **baseline** (`Baseline`) – a reward baseline object
- **discount** (*float*) – reward discount factor
- **gae_lambda** (*float*) – Generalized Advantage Estimation lambda
- **normalize_adv** (*bool*) – indicates whether to normalize the estimated advantages (zero mean and unit std)
- **positive_adv** (*bool*) – indicates whether to shift the (normalized) advantages so that they are all positive

process_samples(*paths*, *log=False*, *log_prefix="*)

Processes sampled paths. This involves:

- computing discounted rewards (returns)
- fitting baseline estimator using the path returns and predicting the return baselines
- estimating the advantages using GAE (+ advantage normalization id desired)
- stacking the path data
- logging statistics of the paths

Parameters

- **paths** (*list*) – A list of paths of size (batch_size) x [5] x (max_path_length)
- **log** (*boolean*) – indicates whether to log
- **log_prefix** (*str*) – prefix for the logging keys

Returns Processed sample data of size [7] x (batch_size x max_path_length)

Return type (dict)

class meta_policy_search.samplers.**DiceSampleProcessor**(*baseline*, *max_path_length*, *discount=0.99*, *gae_lambda=1*, *normalize_adv=True*, *positive_adv=False*, *return_baseline=None*)

Bases: `meta_policy_search.samplers.base.SampleProcessor`

Sample processor for DICE implementations

- fits a reward baseline (use zero baseline to skip this step)
- computes adjusted rewards (reward - baseline)
- normalize adjusted rewards if desired
- zero-pads paths to max_path_length
- stacks the padded path data

Parameters

- **baseline** (Baseline) – a time dependent reward baseline object
- **max_path_length** (*int*) – maximum path length
- **discount** (*float*) – reward discount factor
- **normalize_adv** (*bool*) – indicates whether to normalize the estimated advantages (zero mean and unit std)
- **positive_adv** (*bool*) – indicates whether to shift the (normalized) advantages so that they are all positive
- **return_baseline** (Baseline) – (optional) a state(-time) dependent baseline - if provided it is also fitted and used to calculate GAE advantage estimates

process_samples (*paths*, *log=False*, *log_prefix="*)

Processes sampled paths, This involves:

- computing discounted rewards
- fitting a reward baseline
- computing adjusted rewards (reward - baseline)
- normalizing adjusted rewards if desired
- stacking the padded path data
- creating a mask which indicates padded values by zero and original values by one
- logging statistics of the paths

Parameters

- **paths** (*list*) – A list of paths of size (batch_size) x [5] x (max_path_length)
- **log** (*boolean*) – indicates whether to log
- **log_prefix** (*str*) – prefix for the logging keys

Returns

Processed sample data. A dict containing the following items with respective shapes:

- mask: (batch_size, max_path_length)
- observations: (batch_size, max_path_length, ndim_act)
- actions: (batch_size, max_path_length, ndim_obs)
- rewards: (batch_size, max_path_length)
- adjusted_rewards: (batch_size, max_path_length)
- env_infos: dict of ndarrays of shape (batch_size, max_path_length, ?)

- agent_infos: dict of ndarrays of shape (batch_size, max_path_length, ?)

Return type (dict)

class `meta_policy_search.samplers.`**`MetaSampleProcessor`**(*baseline*, *discount=0.99*, *gae_lambda=1*, *normalize_adv=False*, *positive_adv=False*)

Bases: `meta_policy_search.samplers.base.SampleProcessor`

`process_samples`(*paths_meta_batch*, *log=False*, *log_prefix="*)

Processes sampled paths. This involves:

- computing discounted rewards (returns)
- fitting baseline estimator using the path returns and predicting the return baselines
- estimating the advantages using GAE (+ advantage normalization id desired)
- stacking the path data
- logging statistics of the paths

Parameters

- **`paths_meta_batch`** (*`dict`*) – A list of dict of lists, size: [meta_batch_size] x (batch_size) x [5] x (max_path_length)
- **`log`** (*`boolean`*) – indicates whether to log
- **`log_prefix`** (*`str`*) – prefix for the logging keys

Returns Processed sample data among the meta-batch; size: [meta_batch_size] x [7] x (batch_size x max_path_length)

Return type (list of dicts)

1.6.3 Vectorized Environment Executor

class `meta_policy_search.samplers.vectorized_env_executor.`**`MetaIterativeEnvExecutor`**(*env*, *meta_batc* *envs_per_t* *max_path_*

Bases: `object`

Wraps multiple environments of the same kind and provides functionality to reset / step the environments in a vectorized manner. Internally, the environments are executed iteratively.

Parameters

- **`env`** (`meta_policy_search.envs.base.MetaEnv`) – meta environment object
- **`meta_batch_size`** (*`int`*) – number of meta tasks
- **`envs_per_task`** (*`int`*) – number of environments per meta task
- **`max_path_length`** (*`int`*) – maximum length of sampled environment paths - if the max_path_length is reached, the respective environment is reset

`num_envs`

Number of environments

Returns number of environments

Return type (int)

reset()

Resets the environments

Returns list of (np.ndarray) with the new initial observations.

Return type (list)

set_tasks(*tasks*)

Sets a list of tasks to each environment

Parameters **tasks** (*list*) – list of the tasks for each environment

step(*actions*)

Steps the wrapped environments with the provided actions

Parameters **actions** (*list*) – lists of actions, of length meta_batch_size x envs_per_task

Returns

(tuple): a length 4 tuple of lists, containing obs (np.array), rewards (float), dones (bool), env_infos (dict). Each list is of length meta_batch_size x envs_per_task (assumes that every task has same number of envs)

class meta_policy_search.samplers.vectorized_env_executor.**MetaParallelEnvExecutor**(*env*, *meta_batch_* *envs_per_ta* *max_path_le*

Bases: object

Wraps multiple environments of the same kind and provides functionality to reset / step the environments in a vectorized manner. Thereby the environments are distributed among meta_batch_size processes and executed in parallel.

Parameters

- **env** (meta_policy_search.envs.base.MetaEnv) – meta environment object
- **meta_batch_size** (*int*) – number of meta tasks
- **envs_per_task** (*int*) – number of environments per meta task
- **max_path_length** (*int*) – maximum length of sampled environment paths - if the max_path_length is reached, the respective environment is reset

num_envs

Number of environments

Returns number of environments

Return type (int)

reset()

Resets the environments of each worker

Returns list of (np.ndarray) with the new initial observations.

Return type (list)

set_tasks(*tasks=None*)

Sets a list of tasks to each worker

Parameters **tasks** (*list*) – list of the tasks for each worker

step(*actions*)
Executes actions on each env

Parameters actions (*list*) – lists of actions, of length meta_batch_size x envs_per_task

Returns

(tuple): a length 4 tuple of lists, containing obs (np.array), rewards (float), dones (bool), env_infos (dict)
each list is of length meta_batch_size x envs_per_task (assumes that every task has same number of envs)

1.7 Meta-Trainer

class meta_policy_search.meta_trainer.**Trainer**(*algo*, *env*, *sampler*, *sample_processor*, *policy*, *n_itr*, *start_itr=0*, *num_inner_grad_steps=1*, *sess=None*)

Bases: object

Performs steps of meta-policy search.

Pseudocode:

```
for iter in n_iter:
    sample tasks
    for task in tasks:
        for adapt_step in num_inner_grad_steps
            sample trajectories with policy
            perform update/adaptation step
        sample trajectories with post-update policy
    perform meta-policy gradient step(s)
```

Parameters

- **algo** (*Algo*) –
- **env** (*Env*) –
- **sampler** (Sampler) –
- **sample_processor** (SampleProcessor) –
- **baseline** (Baseline) –
- **policy** (Policy) –
- **n_itr** (*int*) – Number of iterations to train for
- **start_itr** (*int*) – Number of iterations policy has already trained for, if reloading
- **num_inner_grad_steps** (*int*) – Number of inner steps per maml iteration
- **sess** (*tf.Session*) – current tf session (if we loaded policy, for example)

get_itr_snapshot(*itr*)
Gets the current policy and env for storage

train()
Trains policy on env using algo

Pseudocode:

```
for itr in n_itr:
    for step in num_inner_grad_steps:
        sampler.sample()
        algo.compute_updated_dists()
    algo.optimize_policy()
    sampler.update_goals()
```

CHAPTER 2

Indices and tables

- genindex
- modindex

Python Module Index

m

Index

L

M

N

O

P

R

S

T

U

V