

---

# **Fruit API - API Reference**

***Release 1.0.0b***

**Duy Nguyen**

**Oct 29, 2019**



**CONTENTS:**

<b>1</b>	<b>Module fruit.envs</b>	<b>1</b>
<b>2</b>	<b>Module fruit.configs</b>	<b>3</b>
<b>3</b>	<b>Module fruit.agents</b>	<b>5</b>
<b>4</b>	<b>Module fruit.learners</b>	<b>7</b>
<b>5</b>	<b>Module fruit.monitor</b>	<b>9</b>
<b>6</b>	<b>Module fruit.networks</b>	<b>11</b>
<b>7</b>	<b>Module fruit.state</b>	<b>13</b>
<b>8</b>	<b>Module fruit.utils</b>	<b>15</b>
<b>9</b>	<b>Indices and tables</b>	<b>17</b>
	<b>Python Module Index</b>	<b>19</b>
	<b>Index</b>	<b>21</b>



## MODULE FRUIT.ENV

**class** fruit.envs.base.BaseEnvironment

BaseEnvironment defines a unique interface used by Fruit API. Therefore, to integrate external environments into the framework, it is necessary to create a subclass of BaseEnvironment and implement all functions declared in this class.

**clone()**

Duplicate itself. The function is useful in RL methods where multiple learners are trained in different environments.

**get\_action\_space()**

Get the action space of the environment

**Returns** the action space

**get\_current\_steps()**

Get the current number of steps.

**Returns** the current number of steps

**get\_number\_of\_agents()**

Get the number of agents in the environment.

**Returns** the number of agents

**get\_number\_of\_objectives()**

Get the number of objectives.

**Returns** the number of objectives

**get\_state()**

Get current state of the environment.

**Returns** the current state

**get\_state\_space()**

Get the state space of the environment

**Returns** the state space

**is\_atari()**

Check if the environment is an Atari game

**Returns** True if Atari game else False

**is\_render()**

Check if the environment shows GUI.

**Returns** True if showing GUI else False

**is\_terminal()**

Check if the episode is terminated.

**Returns** True if the current episode is terminated else False

**reset()**

Reset the environment to the initial state.

**step(actions)**

Execute the next actions.

**Parameters** **actions** – next actions that will be executed.

**Returns** return a set of rewards

**step\_all(action)**

Similar to `step()` but returns verbose information.

**Parameters** **action** – next actions that will be executed

**Returns** next state, rewards, is terminal, debug info

```
class fruit.envs.ale.ALEEnvironment(rom_name,                frame_skip=4,                re-  
                                peat_action_probability=0.0,  
                                max_episode_steps=10000,  
                                loss_of_life_termination=False,  
                                loss_of_life_negative_reward=False,                bit-  
                                wise_max_on_two_consecutive_frames=False,  
                                is_render=False, seed=None, startup_policy=None,  
                                disable_actions=None, num_of_sub_actions=-1,  
                                state_processor=<fruit.state.processor.AtariProcessor  
                                object>)
```

A wrapper of Arcade Learning Environment, which inherits all members of `BaseEnvironment`.

```
class fruit.envs.gym.GymEnvironment(env_name, state_processor=<fruit.state.processor.AtariProcessor  
                                object>)
```

A wrapper of OpenAI Gym, which inherits all members of `BaseEnvironment`.

```
class fruit.envs.juice.FruitEnvironment(game_engine,                max_episode_steps=10000,  
                                state_processor=None, reward_processor=None)
```

A wrapper of built-in games in Fruit API such as Tank Battle, Food Collector, Milk Factory, Mountain Car, and Deep Sea Treasure. `FruitEnvironment` inherits all members of `BaseEnvironment`.

## MODULE FRUIT.CONFIGS

```
class fruit.configs.base.Config(environment, initial_learning_rate=0.004, history_length=4,  
                                debug_mode=False, gamma=0.99, optimizer=None)
```

A network's configuration, which defines network architecture, training step, and optimizer. A user-defined configuration should be a subclass of `Config`.

**Parameters**

- **environment** – the environment
- **initial\_learning\_rate** – the learning rate
- **history\_length** – the number of historical states as a single state
- **debug\_mode** – enable this flag to print verbose information
- **gamma** – the discounted factor
- **optimizer** – an optimizer (can be retrieved from `OptimizerFactory`)

```
get_debug_mode()
```

Get debug mode flag

**Returns** True if in debug mode else False

```
get_history_length()
```

Get history length of a single state

**Returns** history length

```
get_initial_learning_rate()
```

Get initial learning rate

**Returns** initial learning rate

```
get_input_shape()
```

Get shape of the input

**Returns** input shape

```
get_optimizer()
```

Get the current optimizer used by the configuration

**Returns** the current optimizer

```
get_output_size()
```

Get size of the network's output

**Returns** output size

```
get_params(data_dict)
```

Parse a user-defined data dictionary.

**Parameters** `data_dict` – a user-define data dictionary

**Returns** verbose information of `data_dict`

**`init_config()`**

Create the network

**Returns** parameters of the network

**`predict(session, state)`**

Evaluate the network by using a specified state.

**Parameters**

- **`session`** – the current session id (from Tensorflow)
- **`state`** – a state

**`reset_config()`**

Reset the configuration

**`set_optimizer(optimizer)`**

Set new optimizer for the current configuration

**Parameters** **`optimizer`** – new optimizer

**`train(session, data_dict)`**

Train the network

**Parameters**

- **`session`** – the current session id (from Tensorflow)
- **`data_dict`** – a user-defined data dictionary sent by the learner



## MODULE FRUIT.AGENTS

```
class fruit.agents.base.BaseAgent (network, environment, num_of_threads=1,  
                                   num_of_epochs=100, steps_per_epoch=1000000.0,  
                                   log_dir='./log/', report_frequency=1,  
                                   save_frequency=50000.0)
```

`BaseAgent` contains two entities: an `AgentMonitor` and a set of user-defined learners. It provides a unique interface, which is called by the user's program.

### Parameters

- **network** – a reference to the `PolicyNetwork`
- **environment** – a reference to the environment
- **num\_of\_threads** – the number of learners used in this agent
- **num\_of\_epochs** – the number of training epochs
- **steps\_per\_epoch** – the number of training steps per epoch
- **log\_dir** – checkpoints will be saved in this directory
- **report\_frequency** – each learner will report a debug message with `report_frequency`
- **save\_frequency** – checkpoints will be saved for every `save_frequency`

### `evaluate()`

Evaluate the agent by loading a trained model, which is defined in the `PolicyNetwork`.

**Returns** reward distribution during the testing

### `get_log_dir()`

Get log directory.

**Returns** log directory

### `set_learners(learners)`

Assign a set of user-defined learners into the agent.

**Parameters** **learners** – user-defined learners

### `train()`

Train the agent to learn the environment

**Returns** reward distribution during the training

```
class fruit.agents.factory.AgentFactory
```

As its name, the class is used to instantiate the `BaseAgent` and a set of user-defined learners.

```
static create (agent_type, network, environment, num_of_learners=None, check-  
               point_frequency=50000.0, learner_report_frequency=10, num_of_epochs=50,  
               steps_per_epoch=1000000.0, log_dir='./train/a3c', **args)
```

Instantiate a set of user-defined learners.

#### Parameters

- **agent\_type** – a learner defined by users
- **network** – a reference to the `PolicyNetwork`
- **environment** – is a subclass of `BaseEnvironment`
- **num\_of\_learners** – the number of learners used in the algorithm
- **checkpoint\_frequency** – checkpoints will be saved with `checkpoint_frequency`
- **learner\_report\_frequency** – each learner generates a debug message with `learner_report_frequency`
- **num\_of\_epochs** – the number of training epochs
- **steps\_per\_epoch** – the number of training steps per epoch
- **log\_dir** – the directory that contains checkpoints
- **args** – other args for the specified learner

**Returns** the current agent

```
static get_base_agent (network, environment, num_of_threads=0, check-  
                       point_frequency=50000.0, learner_report_frequency=10,  
                       num_of_epochs=50, steps_per_epoch=1000000.0,  
                       log_dir='./train/a3c')
```

#### Parameters

- **network** – a reference to `PolicyNetwork`
- **environment** – a reference to the environment
- **num\_of\_threads** – the number of learners
- **checkpoint\_frequency** – checkpoints will be saved with `checkpoint_frequency`
- **learner\_report\_frequency** – each learner will print a debug message with `learner_report_frequency`
- **num\_of\_epochs** – the number of training epochs.
- **steps\_per\_epoch** – the number of training steps per epoch
- **log\_dir** – the directory that contains debug information

**Returns** a `BaseAgent`

## MODULE FRUIT.LEARNERS

**class** fruit.learners.base.**Learner**(*agent, name, environment, network, global\_dict, report\_frequency=1*)

Learner represents an RL/deep RL algorithm.

**episode\_end**()

This is a callback function, which is called when an episode ends.

**get\_action**(*state*)

Get the current action from the current state.

**Parameters** *state* – the current state

**Returns** next actions

**get\_probs**(*state*)

Get probability distribution of next actions.

**Parameters** *state* – a state

**Returns** probability distribution over actions

**initialize**()

Initialize the current learner.

**report**(*reward*)

Print verbose information.

**Parameters** *reward* – the current reward

**reset**()

This is a callback function, which is called before or after an episode.

**run**()

Start the learner's thread.

**run\_episode**()

Run an episode

**Returns** a total reward of the episode

**update**(*state, action, reward, next\_state, terminal*)

This is a callback function, which is called for every step.

**Parameters**

- **state** – the current state
- **action** – action
- **reward** – reward retrieved after using *action*

- **next\_state** – the next state
- **terminal** – is it a terminal state or not

**Returns**

## MODULE FRUIT.MONITOR

```
class fruit.monitor.monitor.AgentMonitor(agent, network, log_dir, save_interval=10000.0,  
                                           max_training_epochs=100,  
                                           steps_per_epoch=1000000.0,          num-  
                                           ber_of_objectives=1,          recent_rewards=100,  
                                           idle_time=1)
```

The class is used to monitor the learners and print verbose information during the course of training.

**Parameters**

- **agent** – the BaseAgent
- **network** – the PolicyNetwork
- **log\_dir** – log directory
- **save\_interval** – checkpoints will be saved with `save_interval`
- **max\_training\_epochs** – the maximum number of training epochs
- **steps\_per\_epoch** – the maximum number of training steps
- **number\_of\_objectives** – the number of objectives
- **recent\_rewards** – the number of recent rewards will report
- **idle\_time** – in second (to avoid taking over CPU)

**run\_epochs** (*learners*)

Run all epochs

**Parameters** **learners** – a set of learners

**Returns** reward distribution during the training



## MODULE FRUIT.NETWORKS

```
class fruit.networks.base.BaseNetwork (network_config, using_gpu=True,  
                                         load_model_path=None, num_of_checkpoints=50)
```

This is a holder of network configuration. This class is used to initialize the configuration.

**Parameters**

- **network\_config** – the network configuration
- **using\_gpu** – set True to use GPU if available
- **load\_model\_path** – set a trained model or None
- **num\_of\_checkpoints** – the maximum number of checkpoints during the training

```
create_network ()
```

Create the network :return: network's parameters

```
get_config ()
```

Get the current configuration :return: current configuration

```
get_graph ()
```

Get the current Tensorflow graph :return: the current graph

```
get_session ()
```

Get the current Tensorflow session :return: the current session

```
load_model (path=None)
```

Load network's parameters from file.

**Parameters** **path** – model file

```
predict (state)
```

Evaluate the network

**Parameters** **state** – a state

**Returns** network output

```
reset_network ()
```

Reset the network

```
save_model (*args, **kwargs)
```

Save network's parameters

```
set_save_model (save_model)
```

Enable saving model

**Parameters** **save\_model** – set True to enable saving model

```
train_network (data_dict)
```

Train the network

**Parameters** `data_dict` – data dictionary sent by the learner



## MODULE FRUIT.STATE

```
class fruit.state.processor.Processor
    A state processor, which is used to apply pre-processing into the current state

    clone ()
        Duplicate itself.

    get_number_of_agents ()
        Get the number of agents

        Returns the number of agents

    get_number_of_objectives ()
        Get the number of objectives

        Returns the number of objectives

    get_rewards (reward)
        Get shaping rewards.

        Parameters reward – the original reward

        Returns shaping rewards

    process (obj)
        Process the current state

        Parameters obj – current state

        Returns processed state

    reset ()
        Reset the processor.
```



## MODULE FRUIT.UTILS

**class** `fruit.utils.annealer.Annearer` (*start, end, steps*)

Anneal a value from `start` to `end` in `steps`.

**Parameters**

- **start** – initial value
- **end** – end value
- **steps** – the number of steps is used to anneal a value from `start` to `end`

**anneal** (*steps=1*)

Anneal the current value by the number of steps

**Parameters** **steps** – steps to anneal

**Returns** the current value

**get\_current\_value** ()

Get the current value

**Returns** the current value

**class** `fruit.utils.hypervolume.HVCalculator`

Calculates hypervolume, which is used in multi-objective RL.



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### f

- `fruit.agents.base`, [5](#)
- `fruit.agents.factory`, [5](#)
- `fruit.configs.base`, [3](#)
- `fruit.envs.ale`, [2](#)
- `fruit.envs.base`, [1](#)
- `fruit.envs.gym`, [2](#)
- `fruit.envs.juice`, [2](#)
- `fruit.learners.base`, [7](#)
- `fruit.monitor.monitor`, [9](#)
- `fruit.networks.base`, [11](#)
- `fruit.state.processor`, [13](#)
- `fruit.utils.annealer`, [15](#)
- `fruit.utils.hypervolume`, [15](#)





## A

AgentFactory (class in fruit.agents.factory), 5  
 AgentMonitor (class in fruit.monitor.monitor), 9  
 ALEEnvironment (class in fruit.envs.ale), 2  
 anneal() (fruit.utils.annealer.Annealer method), 15  
 Annealer (class in fruit.utils.annealer), 15

## B

BaseAgent (class in fruit.agents.base), 5  
 BaseEnvironment (class in fruit.envs.base), 1  
 BaseNetwork (class in fruit.networks.base), 11

## C

clone() (fruit.envs.base.BaseEnvironment method), 1  
 clone() (fruit.state.processor.Processor method), 13  
 Config (class in fruit.configs.base), 3  
 create() (fruit.agents.factory.AgentFactory static method), 5  
 create\_network() (fruit.networks.base.BaseNetwork method), 11

## E

episode\_end() (fruit.learners.base.Learner method), 7  
 evaluate() (fruit.agents.base.BaseAgent method), 5

## F

fruit.agents.base (module), 5  
 fruit.agents.factory (module), 5  
 fruit.configs.base (module), 3  
 fruit.envs.ale (module), 2  
 fruit.envs.base (module), 1  
 fruit.envs.gym (module), 2  
 fruit.envs.juice (module), 2  
 fruit.learners.base (module), 7  
 fruit.monitor.monitor (module), 9  
 fruit.networks.base (module), 11  
 fruit.state.processor (module), 13  
 fruit.utils.annealer (module), 15  
 fruit.utils.hypervolume (module), 15  
 FruitEnvironment (class in fruit.envs.juice), 2

## G

get\_action() (fruit.learners.base.Learner method), 7  
 get\_action\_space() (fruit.envs.base.BaseEnvironment method), 1  
 get\_base\_agent() (fruit.agents.factory.AgentFactory static method), 6  
 get\_config() (fruit.networks.base.BaseNetwork method), 11  
 get\_current\_steps() (fruit.envs.base.BaseEnvironment method), 1  
 get\_current\_value() (fruit.utils.annealer.Annealer method), 15  
 get\_debug\_mode() (fruit.configs.base.Config method), 3  
 get\_graph() (fruit.networks.base.BaseNetwork method), 11  
 get\_history\_length() (fruit.configs.base.Config method), 3  
 get\_initial\_learning\_rate() (fruit.configs.base.Config method), 3  
 get\_input\_shape() (fruit.configs.base.Config method), 3  
 get\_log\_dir() (fruit.agents.base.BaseAgent method), 5  
 get\_number\_of\_agents() (fruit.envs.base.BaseEnvironment method), 1  
 get\_number\_of\_agents() (fruit.state.processor.Processor method), 13  
 get\_number\_of\_objectives() (fruit.envs.base.BaseEnvironment method), 1  
 get\_number\_of\_objectives() (fruit.state.processor.Processor method), 13  
 get\_optimizer() (fruit.configs.base.Config method), 3  
 get\_output\_size() (fruit.configs.base.Config method), 3

*method*), 3  
get\_params() (*fruit.configs.base.Config method*), 3  
get\_probs() (*fruit.learners.base.Learner method*), 7  
get\_rewards() (*fruit.state.processor.Processor method*), 13  
get\_session() (*fruit.networks.base.BaseNetwork method*), 11  
get\_state() (*fruit.envs.base.BaseEnvironment method*), 1  
get\_state\_space() (*fruit.envs.base.BaseEnvironment method*), 1  
GymEnvironment (*class in fruit.envs.gym*), 2

## H

HVCalculator (*class in fruit.utils.hypervolume*), 15

## I

init\_config() (*fruit.configs.base.Config method*), 4  
initialize() (*fruit.learners.base.Learner method*), 7  
is\_atari() (*fruit.envs.base.BaseEnvironment method*), 1  
is\_render() (*fruit.envs.base.BaseEnvironment method*), 1  
is\_terminal() (*fruit.envs.base.BaseEnvironment method*), 1

## L

Learner (*class in fruit.learners.base*), 7  
load\_model() (*fruit.networks.base.BaseNetwork method*), 11

## P

predict() (*fruit.configs.base.Config method*), 4  
predict() (*fruit.networks.base.BaseNetwork method*), 11  
process() (*fruit.state.processor.Processor method*), 13  
Processor (*class in fruit.state.processor*), 13

## R

report() (*fruit.learners.base.Learner method*), 7  
reset() (*fruit.envs.base.BaseEnvironment method*), 2  
reset() (*fruit.learners.base.Learner method*), 7  
reset() (*fruit.state.processor.Processor method*), 13  
reset\_config() (*fruit.configs.base.Config method*), 4  
reset\_network() (*fruit.networks.base.BaseNetwork method*), 11  
run() (*fruit.learners.base.Learner method*), 7  
run\_episode() (*fruit.learners.base.Learner method*), 7

run\_epochs() (*fruit.monitor.monitor.AgentMonitor method*), 9

## S

save\_model() (*fruit.networks.base.BaseNetwork method*), 11  
set\_learners() (*fruit.agents.base.BaseAgent method*), 5  
set\_optimizer() (*fruit.configs.base.Config method*), 4  
set\_save\_model() (*fruit.networks.base.BaseNetwork method*), 11  
step() (*fruit.envs.base.BaseEnvironment method*), 2  
step\_all() (*fruit.envs.base.BaseEnvironment method*), 2

## T

train() (*fruit.agents.base.BaseAgent method*), 5  
train() (*fruit.configs.base.Config method*), 4  
train\_network() (*fruit.networks.base.BaseNetwork method*), 11

## U

update() (*fruit.learners.base.Learner method*), 7