MindSpore和MindSpore RL介绍

赵冬斌 朱圆恒 李浩然

MindSpore（昇思）简介

□ 什么是MindSpore?

昇思MindSpore，全场景AI框架

分布式训练原生

内置大模型训练所需的多种并行能力，提供简单易用的大模型分布式策略配置接口，帮助开发者快速实现高性能的大模型分布式训练

AI4S融合计算框架

硬件潜能极致发挥

全场景快速部署

MindSpore（昇思）简介

01 分布式并行

数据集越来越大，模型参数越来越多，如何解决性能问题?

- 算子级并行
- 流水线并行
- Host&Device异构
- 优化器并行
- 自动并行

02 图算深度融合

打破原有的算子边界，重组成为更合理的复合粒度算子。

- 算子跨边界优化
- 多维度自动融合
- 统一图算前端表达

03 动静统一

动态图灵活方便 VS 静态图运算高效，如何平衡?

- 动静态图一键切换
- 双路线探索
  从静态图出发，扩展语法支持
  从动态图出发，极力提升性能

04 Beyond AI

AI+科学计算将把我们的认知拓展到什么样的边界?

MindSpore Science 科学智能套件

- MindSpore SPONGE 分子模拟
- MindSpore Elec 电磁仿真
- MindSpore Flow 流体仿真

05 企业级安全可信

打造一个靠谱的、负责任的、全流程可信赖的AI框架

模型“偷”不走　隐私不泄露　模型抗攻击　OS级安全保护

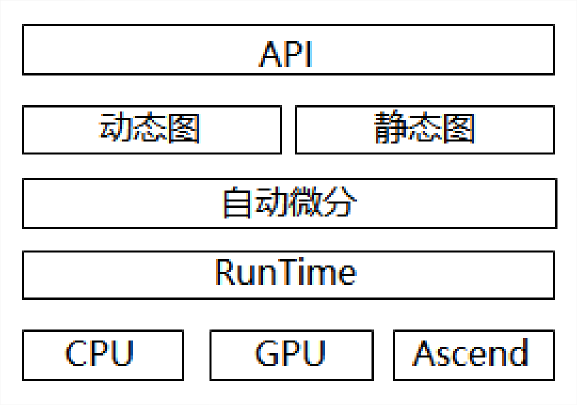
06 融合编程

编程范式全新探索：函数式 + 面向对象编程范式

- 兼顾了定义网络结构和训练过程的易用性
- 实现了从底层兼容AI建模和科学计算建模

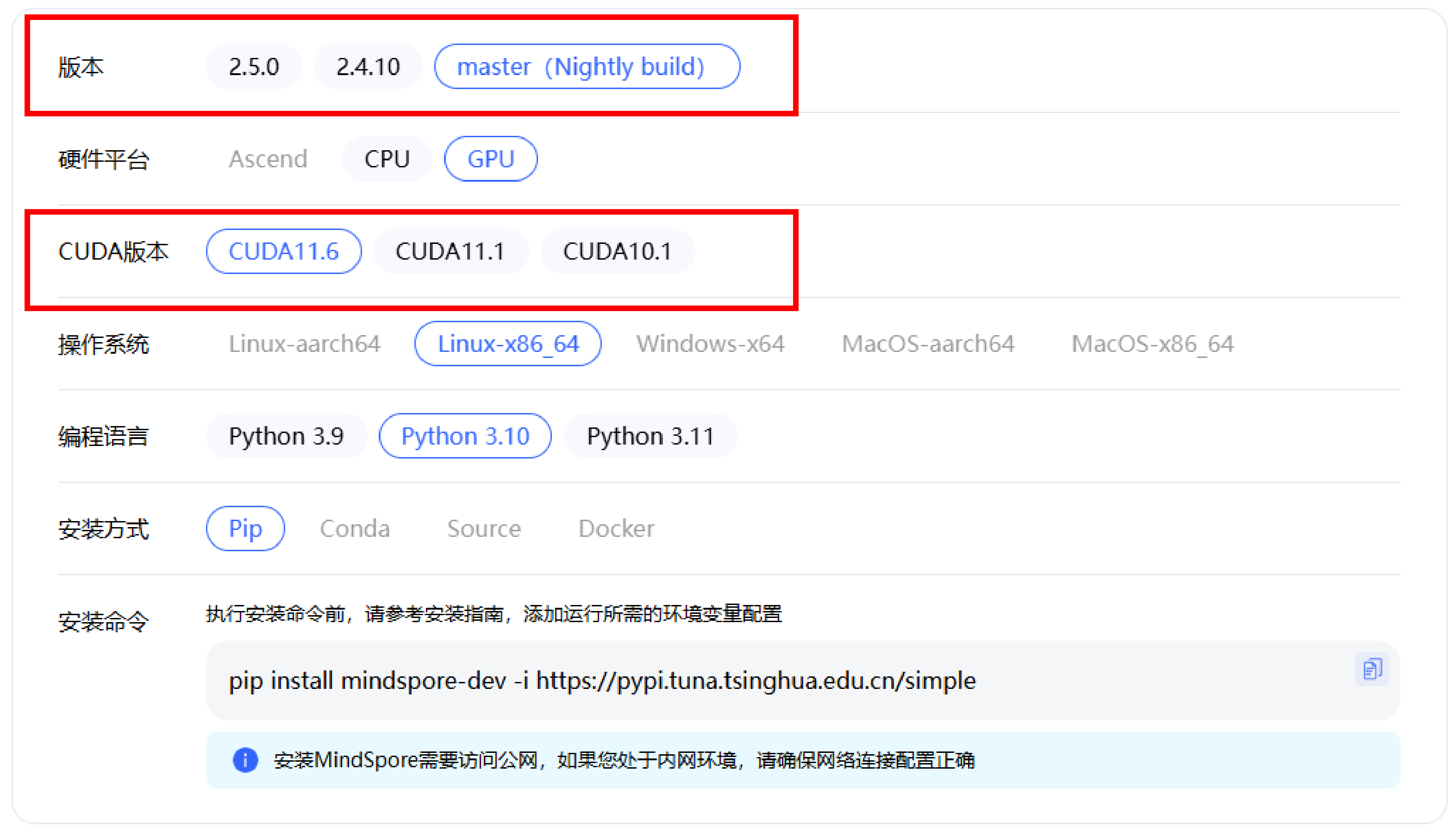
MindSpore（昇思）简介

◻ MindSpore核心架构



特性/框架	MindSpore	PyTorch	TensorFlow
开发语言	Python（原生支持 Ascend）	Python（也支持 C++/Java）	Python + TF Lite/JS/C++ 等
计算图模式	静态 + 动态融合（PyNative + Graph）	默认动态图，支持 TorchScript 编译	静态图为主，Eager 模式支持
设备支持	Ascend ✅、GPU ✅、CPU ✅	GPU ✅、CPU ✅（原生不支持 Ascend）	GPU ✅、CPU ✅、TPU ✅
分布式训练	✅ 强化支持（自动并行等）	✅ 支持 Torch DDP、多机训练	✅ tf.distribute
推理部署	✅ MindIR + Lite + Ascend芯片生态	✅ TorchScript, ONNX	✅ TensorFlow Serving / Lite
模型调试体验	✅ 静/动态图切换易调试	✅ 动态图更自然调试	❌ 静态图调试复杂
官方强化学习库	✅ mindspore_rl	⛔ 无官方库（需用第三方如 RLlib）	✅ TF-Agents / TensorFlow RL
生态支持	新兴生态（华为主推）	✅ 学术界主流，社区活跃	✅ 工业界使用广泛，成熟稳定

MindSpore安装



注意事项

- **关于版本选择：**需要考虑MindRL版本需求，推荐使用MindSpore 2.1.0 + MindRL 0.7.0
- **CUDA版本：**如果使用GPU对CUDA版本以及对应的CUDNN版本有要求，目前应该只支这三个版本号

详细的版本依赖：https://www.mindspore.cn/versions

MindSpore基础使用

核心数据类型：Tensor

✓ 数据获取

```
import numpy as np
import mindspore as ms
from mindspore import Tensor
# 直接赋值
a = Tensor([[1,2],[3,4]], dtype=ms.float32)

# 从numpy中获取
b = Tensor(np.zeros([1,2,3]), dtype=ms.float32)
```

✓ 数据计算

```
a = Tensor(np.random.rand(3, 2), dtype=ms.float32)
b = Tensor(np.random.rand(3, 2), dtype=ms.float32)

# + / -
print(a+b)
print(ops.add(a, b))

print(a-b)
print(ops.sub(a, b))

# *
print(a*b)
print(ops.mul(a,b))
```

✓ 转置乘法

```
# 转置
ops.transpose(b, (1, 0))
b.transpose() # 只适用于2D数组


# 矩阵乘法
print(a @ ops.transpose(b, (1,0)))
print(ops.matmul(a, ops.transpose(b, (1,0))))
```

MindSpore基础使用

mindspore.numpy vs 原生numpy

📌 核心区别总览

特性	`mindspore.numpy`	原生 `numpy`
后端	基于 MindSpore 图执行（可 GPU/Ascend）	仅 CPU，纯 Python 实现
张量类型	`Tensor`（MindSpore 自定义类型）	`ndarray`
支持设备	CPU / GPU / Ascend	仅 CPU（少数情况支持 MKL）
动态图/静态图	支持 `GRAPH_MODE` 编译执行	无图机制，纯命令式
用途	构建神经网络计算图	科学计算、数据处理、教学等
API 覆盖	覆盖 NumPy 常用函数（不完全）	功能最全

✅ 什么时候用哪个?

场景	推荐库
数据加载、预处理	原生 `numpy`
模型计算图、训练逻辑	`mindspore.numpy`
调试、探索性数据分析	原生 `numpy`
部署到 Ascend/GPU	`mindspore.numpy` + `ops` / `Tensor`

🧠 注意事项

- `mindspore.numpy.array()` 返回的是 `Tensor` **类型**，不是 `ndarray`。
- `mindspore.numpy` 是为了让你在 **写神经网络时用 NumPy 风格代码**，但享受硬件加速。
- 它并 **不支持所有 NumPy 函数**（例如某些高级统计、傅里叶变换、字符串处理等）。
- 用了 `mindspore.numpy` 就要遵循 MindSpore 的执行模型，比如不能用 `for` 循环更新 Tensor 内容等。

MindSpore基础使用

如何构建一个简单的神经网络?

```
import mindspore
from mindspore import nn, Tensor, ops
import numpy as np
class SimpleNN(nn.Cell):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Dense(input_dim, hidden_dim)  # 输入层到隐藏层
        self.fc2 = nn.Dense(hidden_dim, hidden_dim)  # 隐藏层到隐藏层
        self.fc3 = nn.Dense(hidden_dim, output_dim)  # 隐藏层到输出层
        self.relu = nn.ReLU()  # 激活函数

    def construct(self, x):
        x = self.fc1(x)  # 第一层
        x = self.relu(x)  # 激活
        x = self.fc2(x)  # 第二层
        x = self.relu(x)  # 激活
        x = self.fc3(x)  # 输出层
        return x
```

- ✓ 在MindSpore中，Cell类是构建所有网络的基类，也是网络的基本单元。
- ✓ 一个神经网络模型表示为一个Cell，它由不同的子Cell构成
- ✓ 构建神经网络（计算图），类似于Pytorch中的forward函数

特性	MindSpore 的 construct	PyTorch 的 forward
作用	定义前向传播	定义前向传播
必须实现	✅ 是必须的	✅ 是必须的
名字能改吗	❌ 不行，必须叫 construct	✅ 可以随意改（比如叫 run），但不推荐
自动调用方式	model(x) 自动调用 construct()	model(x) 自动调用 forward()
兼容静态图	✅ 是的（Graph模式构图）	❌ 默认动态图（可手动 trace 静态）
背后机制	Graph模式构图依赖 AST 分析	Python 本地执行

MindSpore基础使用

中国科学院自动化研究所
INSTITUTE OF AUTOMATION
CHINESE ACADEMY OF SCIENCES

□ 如何训练一个神经网络?

✓ 准备数据，模型和优化

```python
'''---------------准备数据---------------------'''
# 生成随机数据
X_train = np.random.randn(100, 3).astype(np.float32)  # 100个样本, 3维特征
y_train = np.random.randint(0, 2, size=(100, 1)).astype(np.float32)  # 100个标

# 转换为MindSpore Tensor
X_train = Tensor(X_train)
y_train = Tensor(y_train)

'''---------------构建网络---------------------'''
model = SimpleNN(3, 5, 1)  # 输入3个特征, 5个隐藏层单元, 输出1个预测值

'''---------------准备优化器-------------------'''
# 损失函数: 交叉熵损失
loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')

# 优化器: SGD
optimizer = nn.SGD(params=model.trainable_params(), learning_rate=0.01)
```

✓ 计算梯度并反传

```python
def forward_fn(data, label):
    logits = model(data)
    loss = loss_fn(logits, label)
    return loss, logits

grad_fn = mindspore.value_and_grad(forward_fn, None,
                                   optimizer.parameters, has_aux=True)

def train_step(data, label):
    (loss, _), grads = grad_fn(data, label)
    optimizer(grads)
    return loss

model.set_train()

for epoch in range(10):
    loss = train_step(X_train, y_train)
    print(f"Epoch {epoch+1}, Loss: {loss.asnumpy():.4f}")
```

✓ 一种更简单的方式

```python
# 另一种方式
# 包装为标准训练步骤
loss_net = nn.WithLossCell(model, loss_fn)
train_step = nn.TrainOneStepCell(loss_net, optimizer)
train_step.set_train()
for epoch in range(10):
    loss = train_step(X_train, y_train)
    print(f"Epoch {epoch+1}, Loss: {loss.asnumpy():.4f}")
```

MindSpore基础使用

□ 如何使用MindSpore写一个简单的强化学习算法（例如Policy Gradient）

在θ参数化下的策略下产生的轨迹为$\tau \sim p_\theta(\tau) = p(\tau|\theta)$，则该轨迹的回报为$r(\tau) = \sum_{t=0}^{T} \gamma_t r_t$。那么强化学习的目标函数为

$$J(\theta) = E[r(\tau)] = \int_I p_\theta(\tau) r(\tau) d\tau$$

$$\nabla_\theta J(\theta) = E[(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t))(\sum_{t=0}^{T} r(s_t, a_t))]$$

REINFORCE算法流程

1. 从$\pi_\theta(a_t|s_t)$中采用策略并执行得到轨迹$\{\tau^i\}$
2. $\nabla_\theta J(\theta) \approx \frac{1}{N}(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t))(\sum_{t=0}^{T} r(s_t, a_t))$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
4. 返回第一步，直到策略收敛。

参考https://coderhaoranlee.github.io/2020/07/13/rl-pg/

MindSpore基础使用

```python
# ------------------------
# 策略网络
# ------------------------
class PolicyNet(nn.Cell):
    def __init__(self, state_dim, action_dim):
        super(PolicyNet, self).__init__()
        self.fc1 = nn.Dense(state_dim, 128, weight_init=Normal(0.1))
        self.fc2 = nn.Dense(128, action_dim, weight_init=Normal(0.1))
        self.softmax = nn.Softmax(axis=-1)

    def construct(self, x):
        x = ops.relu(self.fc1(x))
        return self.softmax(self.fc2(x))
```

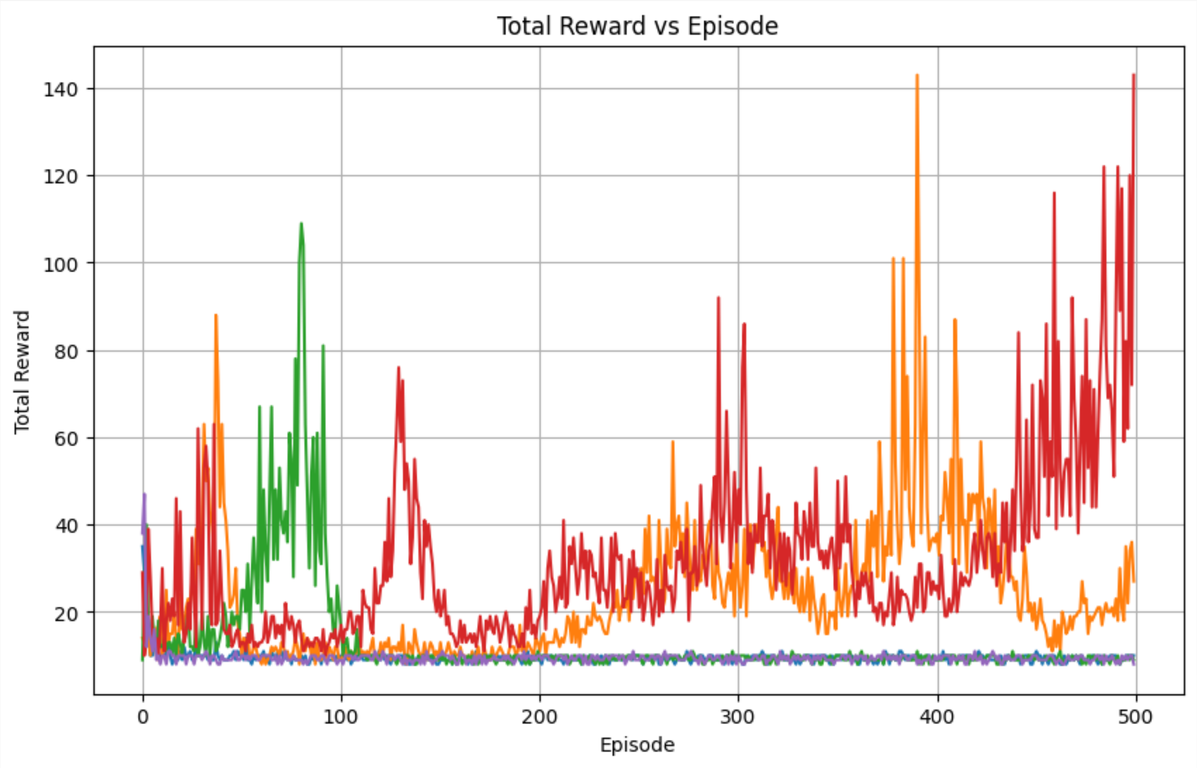
```python
# ------------------------
# 回报计算函数
# ------------------------
def compute_returns(rewards, gamma=0.99):
    returns = []
    G = 0
    for r in reversed(rewards):
        G = r + gamma * G
        returns.insert(0, G)
    return np.array(returns, dtype=np.float32)
```

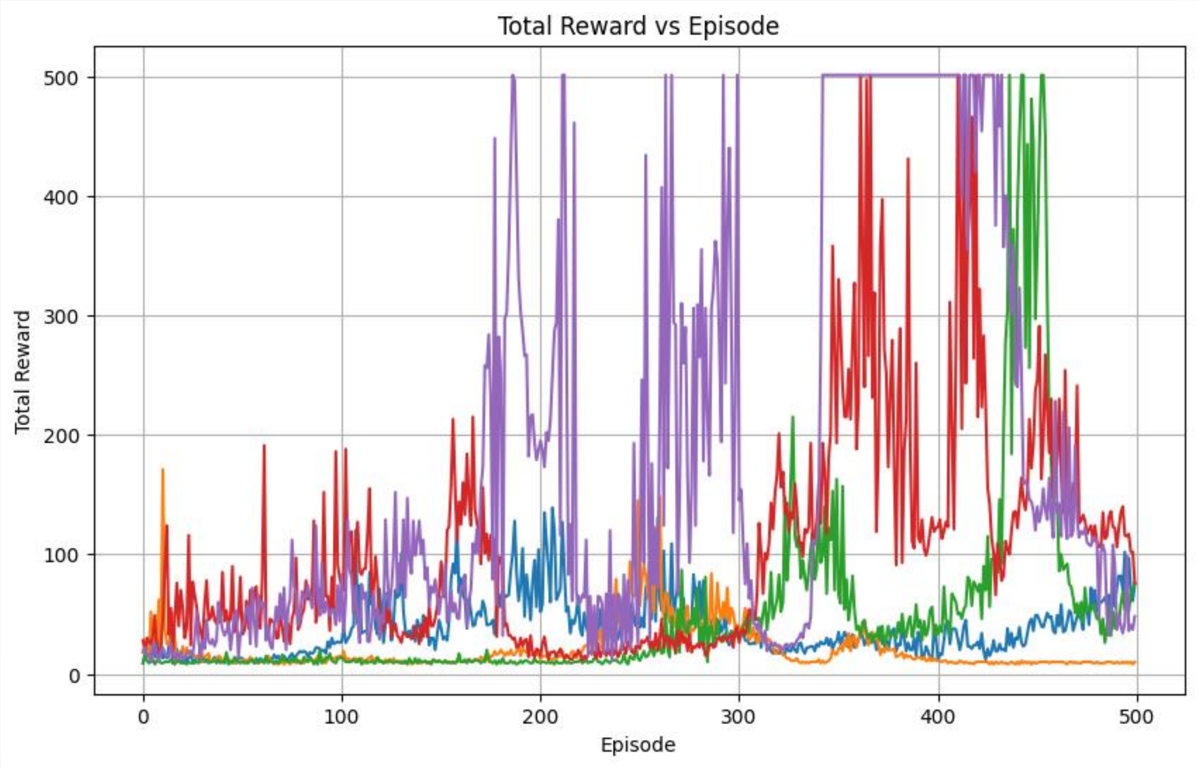
1. 从$\pi_\theta(a_t|s_t)$中采用策略并执行得到轨迹$\{\tau^i\}$
2. $\nabla_\theta J(\theta) \approx \frac{1}{N}(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t))(\sum_{t=0}^{T} r(s_t, a_t))$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
4. 返回第一步，直到策略收敛。

```python
# ------------------------
# 自定义损失（负的 log 概率 * 回报）
# ------------------------
class PolicyLoss(nn.Cell):
    def __init__(self, policy_net):
        super(PolicyLoss, self).__init__()
        self.policy_net = policy_net
        self.log = ops.Log()

    def construct(self, states, actions, returns):
        probs = self.policy_net(states)
        log_probs = self.log(probs)

        # BatchGather: 每个样本选对应动作的 log_prob
        batch_indices = Tensor(np.arange(actions.shape[0]), ms.int32)
        selected_log_probs = log_probs[batch_indices, actions]

        loss = -ops.ReduceMean()(selected_log_probs * returns)
        return loss
```

MindSpore基础使用

□ 实验结果



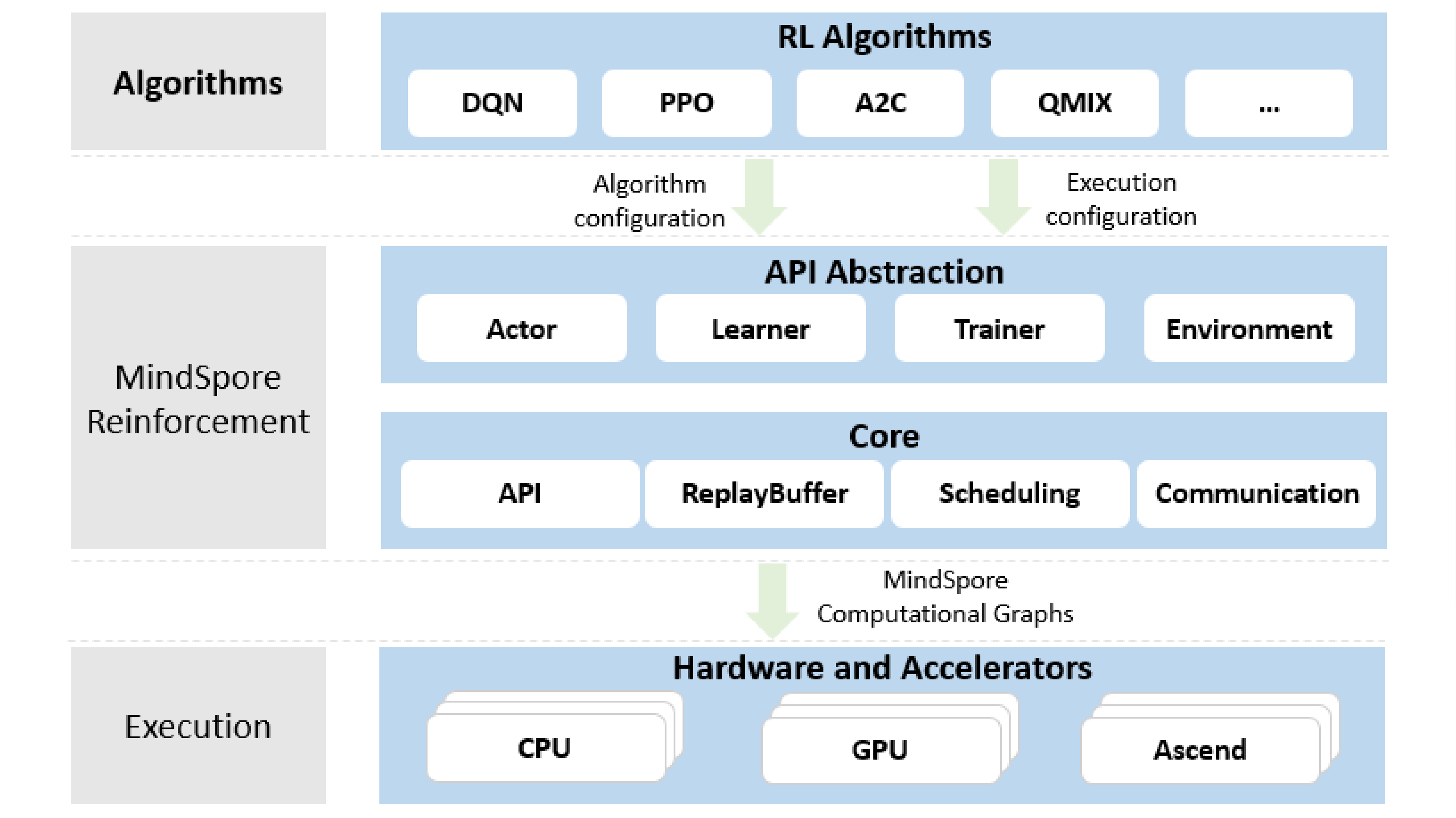


MindSpore RL（MindRL）介绍

✓ 一个开源的强化学习框架，支持使用强化学习算法对agent进行**分布式训练**。

✓ 提供了**干净整洁的API抽象**，将算法与部署和执行注意事项解耦，包括加速器的使用、并行度和跨worker集群计算的分布。

✓ 将强化学习算法转换为一系列编译后的**计算图**，然后由MindSpore框架在CPU、GPU或Ascend AI处理器上高效运行。



参考链接： https://gitee.com/mindspore-lab/mindrl/tree/r0.7/

MindSpore RL介绍

中国科学院自动化研究所
INSTITUTE OF AUTOMATION CHINESE ACADEMY OF SCIENCES

pip安装

使用pip命令安装，请从MindSpore Reinforcement下载页面下载并安装whl包。

```
pip install https://ms-release.obs.cn-north-4.myhuaweicloud.com/{MindSpore_v
```

- 在联网状态下，安装whl包时会自动下载MindSpore Reinforcement安装包的依赖项（依赖项详情参见requirement.txt），其余情况需自行安装。
- `{MindSpore_version}` 表示MindSpore版本号，MindSpore和Reinforcement版本配套关系参见页面。
- `{Reinforcement_version}` 表示Reinforcement版本号。例如下载0.1.0版本Reinforcement时，`{MindSpore_version}应写为1.5.0`，`{Reinforcement_version}` 应写为0.1.0。

源码编译安装

下载源码，下载后进入 `reinforcement` 目录。

```
git clone https://gitee.com/mindspore/reinforcement.git
cd reinforcement/
bash build.sh
pip install output/mindspore_rl-{Reinforcement_version}-py3-none-any.whl
```

其中，`build.sh` 为 `reinforcement` 目录下的编译脚本文件。`{Reinforcement_version}` 表示MindSpore Reinforcement版本号。

安装依赖项

```
cd reinforcement && pip install requirements.txt
```

验证是否成功安装

执行以下命令，验证安装结果。导入Python模块不报错即安装成功：

```
import mindspore_rl
```

需要注意MindSpore和MindRL的版本对应！

MindSpore Reinforcement	分支	MindSpore
0.7.0	r0.7	2.1.0
0.6.0	r0.6	2.0.0
0.5.0	r0.5	1.8.0
0.3.0	r0.3	1.7.0
0.2.0	r0.2	1.6.0
0.1.0	r0.1	1.5.0

MindSpore RL介绍

中国科学院自动化研究所
INSTITUTE OF AUTOMATION CHINESE ACADEMY OF SCIENCES

快速入门

MindSpore Reinforcement的算法示例位于`reinforcement/example/`下，以一个简单的算法Deep Q-Learning (DQN) 示例，演示MindSpore Reinforcement如何使用。

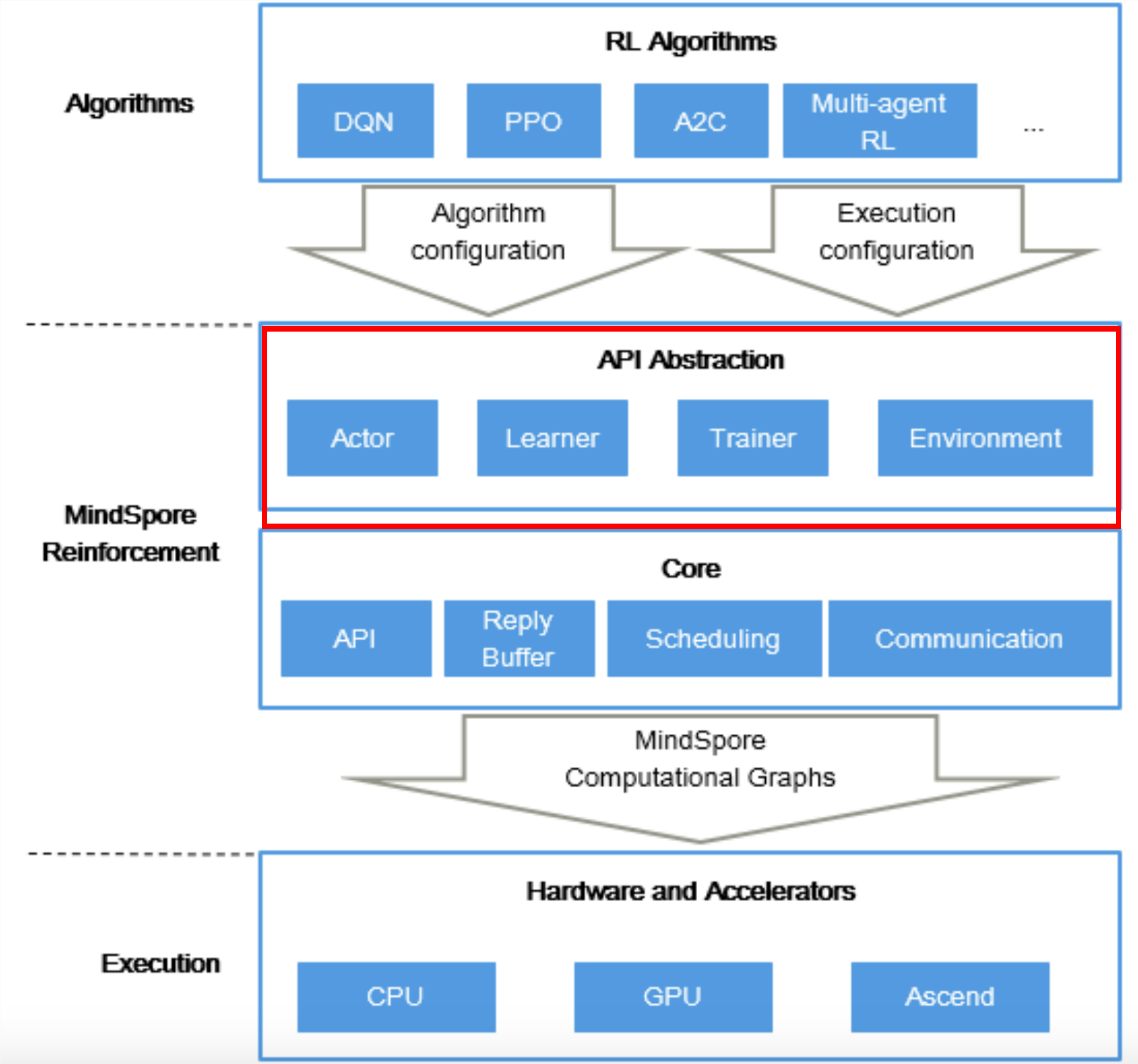
第一种开箱即用方式，使用脚本文件直接运行:

```
cd reinforcement/example/dqn/scripts
bash run_standalone_train.sh
```

第二种方式，直接使用`config.py`和`train.py`，可以更灵活地修改配置：

```
cd reinforcement/example/dqn
python train.py --episode 1000 --device_target GPU
```

MindSpore RL介绍



☐ Actor

- ✓ 与环境交互，产生经验（obs, action, reward, next_obs）
- ✓ 通常使用策略网络输出动作（可能带随机性）

☐ Learner

- ✓ 负责使用经验（如 ReplayBuffer）进行训练。
- ✓ 通常封装 loss 构建、梯度计算、优化器更新等。

☐ Trainer

- ✓ 协调整个训练流程：调度Actor收集数据，调用Learner更新模型
- ✓ 控制训练 epoch、保存模型、评估等。

☐ Environment

- ✓ 封装 Gym、MindSpore Env、模拟器等，提供标准化的reset()和step()接口。

```
[Trainer]
  ├── calls → [Actor] → 与 [Environment] 交互，生成经验
  └── calls → [Learning] ← 用经验更新 [Policy]
```

MindSpore RL介绍

☐ Actor

r0.7

- .github/workflows
- docs
- example
- mindspore_rl
  - agent
    - __init__.py
    - actor.py
    - agent.py
    - learner.py
    - trainer.py

```python
class Actor(nn.Cell):
    r"""
    Base class for all actors. Actor is a class used to interact with the envir
    """

    def __init__(self):
        super(Actor, self).__init__(auto_prefix=False)

    def get_action(self, phase, params):
        """
        get_action is the method used to obtain the action.
        User will need to overload this function according to
        the algorithm. But argument of this function should be
        phase and params. This interface will not interact with
        environment

        Args:
            phase (enum): A enumerate value states for init, collect, eval or c
            params (tuple(Tensor)): A tuple of tensor as input, which is used t

        Returns:
            tuple(Tensor), a tuple of tensor as output, containing actions and
        """

        raise NotImplementedError("Method should be overridden by subclass.")

    def act(self, phase, params):
        """
        The act function will take an enumerate value and observation or other
        calculating the action. It will return a set of output which contains r
        experience. In this function, agent will interact with environment.

        Args:
            phase (enum): A enumerate value states for init, collect, eval or c
            params (tuple(Tensor)): A tuple of tensor as input, which is used t

        Returns:
            tuple(Tensor), a tuple of tensor as output, which states for experi
        """

        raise NotImplementedError("Method should be overridden by subclass.")
```

方法	用途	返回内容	适用场景
`get_action()`	根据当前观测选择动作	返回动作本身（通常是 `Tensor`）	在线决策、环境交互、部署
`act()`	与环境交互并记录所有信息	返回完整的 `Transition`（状态、动作、log_prob 等）	训练数据采集、用于更新策略

1. `get_action(obs)` 示例：

python

```python
action = actor.get_action(obs)
env.step(action)
```

- 只返回动作
- 一般用于 **部署阶段**、测试、评估、或收集轨迹不需要训练信息时

2. `act(obs)` 示例：

python

```python
transition = actor.act(obs)
# transition 包含：
# - state
# - action
# - log_prob
# - value（可能有）
# - done
```

- 返回的是一个完整的数据结构（通常是 `Transition`）
- 用于 **训练阶段**：你需要 `log_prob` 计算 loss，需要 `value` 计算 advantage 等

MindSpore RL介绍

Learner

r0.7

- .github/workflows
- docs
- example
- mindspore_rl
  - agent
    - __init__.py
    - actor.py
    - agent.py
    - learner.py
    - trainer.py

```python
class Learner(nn.Cell):
    r"""
    The base class of the learner. Calculate and update the self generated network through
    """

    def __init__(self):
        super(Learner, self).__init__(auto_prefix=False)

    def learn(self, experience):
        """
        The interface for the learn function. The behavior of the `learn` function
        depend on the user's implementation. Usually, it takes the `samples` form
        replay buffer or other Tensors, and calculates the loss for updating the networks.

        Args:
            experience(tuple(Tensor)): Sampling from the buffer.

        Returns:
            tuple(Tensor), result which outputs after updating weights
        """

        raise NotImplementedError("Method should be overridden by subclass.")
```

learn() 通常要做的事情：

1. **解构 batch**：从 ReplayBuffer 或交互中采样数据。
2. **前向传播**：用策略网络输出 logits/probs/值函数等。
3. **计算损失**：构造 loss，比如 policy gradient、value loss、entropy 等。
4. **反向传播并优化**：用 MindSpore 的 `value_and_grad()` 或 `GradOperation`。
5. **返回 loss 或调试信息**：方便打印、记录、监控。

```python
for epoch in range(num_epochs):
    batch = buffer.sample()
    loss = learner.learn(batch)
    print("Loss:", loss)
```

MindSpore RL介绍

□ Agent

r0.7

- .github/workflows
- docs
- example
- mindspore_rl
  - agent
    - __init__.py
    - actor.py
    - agent.py
    - learner.py
    - trainer.py

```
class Agent(nn.Cell):
    r"""
    The base class for the Agent. As the definition of agent, it is composed of
    It has basic act and learn functions for interaction with environment and u

    Args:
        actors(Actor): The actor instance.
        learner(Learner): The learner instance.
    """

    def __init__(self, actors, learner):
        super(Agent, self).__init__(auto_prefix=False)
        self._actors = actors
        self._learner = learner

    def get_action(self, phase, params):
        """
        The get_action function will take an enumerate value and observation or
        calculating the action. It will return a set of outputs containing acti
        function, agent will not interact with environment.
        """

        raise NotImplementedError("Method should be overridden by subclass.")

    def act(self, phase, params):
        """
        The act function will take an enumerate value and observation or other
        calculating the action. It will return a set of output which contains r
        experience. In this function, agent will interact with environment.
        """

        raise NotImplementedError("Method should be overridden by subclass.")

    def learn(self, experience):
        """
        The learn function will take a set of experience as input to calculate
        the weights.
        """

        raise NotImplementedError("Method should be overridden by subclass.")
```

✓ 传入参数是实例化的Actor和Learner

名称	角色	举例
`Actor.act()`	生成动作和训练数据	与环境交互
`Learner.learn()`	用数据更新策略参数	反向传播优化

MindSpore RL介绍

□ Trainer

r0.7

- .github/workflows
- docs
- example
- mindspore_rl
  - agent
    - __init__.py
    - actor.py
    - agent.py
    - learner.py
    - trainer.py

```python
class Trainer(nn.Cell):
    def train(self, episodes, callbacks=None, ckpt_path=None):

        Args:
            episodes(int): the number of training episodes.
            callbacks(Optional[list[Callback]]): List of callbac
            ckpt_path(Optional[str]): The checkpoint file path t
        """

        cb_params = CallbackParam()
        cb_params.episodes_num = episodes

        # Move TimeCallback to the first to exclude the time of
        for item in callbacks: ···
        # 1 Using `CallbackManager` to traverse each callback.
        with CallbackManager(callbacks) as callback_list: ···

    def train_one_episode(self):
        """
        The interface of train one episode function in train.
        And the output of this function must be constricted as `
        """
        raise NotImplementedError("Method train_one_episode shou
            and the output must be constricted as `loss, rewards

    def evaluate(self):
        """
        The interface of the evaluate function for evaluate in t
        """
        raise NotImplementedError("Method evaluate should be ove

    def _load_ckpt(self, ckpt_path=None, name=None, net=None): ···

    def _init_or_restore(self, ckpt_path=None): ···

    def trainable_variables(self): ···

    def load_and_eval(self, ckpt_path=None): ···
```

```python
for i in range(episodes):
    callback_list.episode_begin(cb_params)

    # 4 Get the result of `train_one_episode` func, and deal with three situation:
    #   a) Default using: Three objects in tuple, each stand for `loss`, `rewards` and `steps`.
    #   b) User defined: Four objects in tuple, the first three is same as default using, the last
    #      one `others` can be tuple or single one as user defined.
    #   c) Other situation: Runtime error.
    ans = self.train_one_episode()
    loss, rewards, steps, others = [], [], [], []
    if len(ans) == 3:
        loss, rewards, steps = ans
    elif len(ans) == 4:
        loss, rewards, steps, others = ans
    else:
        raise RuntimeError("The output number of function `train_one_episode` must be 3 or 4, \
            and represent for `loss, rewards, steps, [optional]others.` in order")

    cb_params.loss = loss
    cb_params.total_rewards = rewards
    cb_params.steps = steps
    cb_params.others = others
    callback_list.episode_end(cb_params)
    cb_params.cur_episode = i + 1
callback_list.end(cb_params)
```

Trainer 主要负责什么?

功能	说明
控制训练循环	epoch、episode、step 的管理
采集数据	调用 `Actor.act()` 与 `Environment` 交互
触发更新	调用 `Learner.learn()` 用经验更新策略
保存模型	周期性保存 checkpoint
日志记录与评估	打印 loss / reward，调用 `Evaluator`

MindSpore RL介绍

r0.7

☐ Environment

- environment
  - __init__.py
  - _remote_env_wrapper.py
  - action_norm_wrapper.py
  - action_repeat_wrapper.py
  - async_parallel_wrapper.py
  - batch_wrapper.py
  - dmc_environment.py
  - env_process.py
  - environment.py
  - gym_environment.py
  - ms_environment.py
  - multi_environment_wrapper.py
  - petting_zoo_mpe_environment.py
  - process_environment.py
  - pyfunc_wrapper.py
  - python_environment.py
  - random_environment.py
  - registration.py
  - sc2_environment.py
  - space.py
  - space_adapter.py
  - sync_parallel_wrapper.py

```python
class Environment(nn.Cell):
        super(Environment, self).__init__()

    @property
    def action_space(self) -> Space:…

    @property
    def observation_space(self) -> Space:…

    @property
    def reward_space(self) -> Space:…

    @property
    def done_space(self) -> Space:…

    @property
    def config(self) -> dict:…

    @property
    def batched(self) -> bool:…

    @property
    def num_environment(self) -> int:…

    @property
    def num_agent(self) -> int:…

    @property
    def _num_reset_out(self) -> int:…

    @property
    def _num_step_out(self) -> int:…

    def reset(self):…

    def step(self, action: Union[Tensor, np.ndarray]):…

    def send(self, action: Union[Tensor, np.ndarray], env_id: Union[Tensor, np.ndarray])

    def recv(self):…

    def set_seed(self, seed_value: Union[int, Sequence[int]]) -> bool:…

    def render(self) -> Union[Tensor, np.ndarray]:…

    def close(self) -> bool:…
```

📦 Gym 与 MindRL 对比：

环境接口项	Gym 定义（返回值）	MindRL 定义（空间描述）
`obs`	`obs_space`	`obs_space`
`action`	`action_space`	`action_space`
`reward`	标量 float	✅ `reward_space`（标量/分布）
`done`	bool	✅ `done_space`（通常是 {True, False}）

MindSpore RL介绍

◻ 以Policy Gradient为例

✓ 策略网络定义

```python
class PGPolicyAndNetwork:
    """PGPolicyAndNetwork"""

    class ActorNet(nn.Cell):
        """ActorNet"""

        def __init__(
            self, input_size, hidden_size, output_size, compute_typ
        ):
            super().__init__()
            self.dense1 = nn.Dense(
                input_size, hidden_size, weight_init="XavierUniform
            ).to_float(compute_type)
            self.dense2 = nn.Dense(
                hidden_size, output_size, weight_init="XavierUnifor
            ).to_float(compute_type)
            self.active = P.Tanh()
            self.softmax = P.Softmax()
            self.cast = P.Cast()

        def construct(self, x):
            x = self.dense1(x)
            x = self.active(x)
            x = self.dense2(x)
            return self.cast(self.softmax(x), mindspore.float32)

    class CollectPolicy(nn.Cell):···

    class EvalPolicy(nn.Cell):···

    def __init__(self, params):
        self.actor_net = self.ActorNet(···
        self.collect_policy = self.CollectPolicy(self.actor_net)
        self.eval_policy = self.EvalPolicy(self.actor_net)
```

✓ Actor实现

```python
class PGActor(Actor):
    """PG Actor"""

    def __init__(self, params=None):
        # pylint: disable=R1725
        super(PGActor, self).__init__()
        self._params_config = params
        self._environment = params.get("collect_environment")
        self._eval_env = params.get("eval_environment")
        self.collect_policy = params.get("collect_policy")
        self.eval_policy = params.get("eval_policy")
        self.expand_dims = P.ExpandDims()
        self.cast = P.Cast()
        self.print = P.Print()

    def act(self, phase, params):
        if phase == 2:
            # Sample action to act in env
            ts0 = self.expand_dims(params, 0)
            action = self.collect_policy(ts0)
            action = self.cast(action, mindspore.int32)
            new_state, reward, done = self._environment.step(action)
            reward = self.expand_dims(reward, 0)
            done = self.expand_dims(done, 0)
            return done, reward, new_state, action
        if phase == 3:
            # Evaluate the trained policy
            ts0 = self.expand_dims(params, 0)
            action = self.eval_policy(ts0)
            new_state, reward, done = self._eval_env.step(
                self.cast(action, mindspore.int32)
            )
            reward = self.expand_dims(reward, 0)
            done = self.expand_dims(done, 0)
            return done, reward, new_state

        self.print("Phase is incorrect")
        return 0
```

✓ Learner实现

```python
class PGLearner(Learner):
    class ActorNNLoss(nn.Cell):
        """Actor loss"""
        def __init__(self, actor_n
            super().__init__(auto_prefix=False)
            self.actor_net = actor_net
            self.reduce_mean = P.ReduceMean()
            self.reduce_sum = P.ReduceSum()
            self.onehot = P.OneHot()
            self.depth = depth
            self.on_value = Tensor(1.0, mindspore.float32)
            self.off_value = Tensor(0.0, mindspore.float32)
            self.log = P.Log()
            self.cast = P.Cast()

        def construct(self, state, action, reward):
            onehot_action = self.onehot(
                action, self.depth, self.on_value, self.off_value
            ).reshape((-1, self.depth))
            act_prob = self.actor_net(state)
            log_prob = self.reduce_sum(-1.0 * self.log(act_prob) * onehot_action, 1)
            loss = self.reduce_mean(log_prob * reward.reshape((-1,)))
            return loss

    def __init__(self, params):
        # pylint: disable=R1725
        super(PGLearner, self).__init__()
        self._params_config = params
        self.actor_net = params["actor_net"]
        self.action_dim = params["action_space_dim"]
        self.zero_float = Tensor([0.0], mindspore.float32)
        optimizer = nn.Adam(
            self.actor_net.trainable_params(), learning_rate=params["lr"]
        )
        actor_loss_net = self.ActorNNLoss(self.actor_net, self.action_dim)
        self.actor_net_train = nn.TrainOneStepCell(actor_loss_net, optimizer)
        self.actor_net_train.set_train(mode=True)
        self.discount_return = DiscountedReturn(gamma=params["gamma"])

    def learn(self, experience):
        """Calculate the td_error"""
        state = experience[0]
        reward = experience[1]
        action = experience[2]
        mask = experience[3]
        returns = self.discount_return(reward, mask, self.zero_float)
        loss = self.actor_net_train(state, action, returns)
        return loss
```

MindSpore RL介绍

□ 以Policy Gradient为例

✓ Trainer实现

```python
class PGTrainer(Trainer):
    """PGTrainer"""

    def __init__(self, msrl, params):...

    def trainable_variables(self):
        """Trainable variables for saving."""
        trainable_variables = {"actor_net": self.msrl.learner.actor_net}
        return trainable_variables

    @jit
    def train_one_episode(self):
        """Train one episode"""
        obs_list, action_list, reward_list, masks, steps = self.run_one_episode()
        loss = self.msrl.agent_learn([obs_list, reward_list, action_list, masks])
        return loss, steps, self.loop_size

    @jit
    def run_one_episode(self):
        """run_one_episode(dynamic list is not supported in graph mode, so use static loop.)"""
        steps = self.zero_value
        done_status = self.zero_value
        done_num = self.zero_value
        masks = self.masks
        obs = self.msrl.collect_environment.reset()
        while steps < self.loop_size:
            self.obs_list.write(steps, obs)
            done, r, obs, a = self.msrl.agent_act(trainer.COLLECT, obs)
            self.action_list.write(steps, a)
            self.reward_list.write(steps, r)...
                self.msrl.collect_environment.reset()
            steps += 1
        states = self.obs_list.stack()
        rewards = self.reward_list.stack()
        actions = self.action_list.stack()
        self.obs_list.clear()
        self.reward_list.clear()
        self.action_list.clear()
        return states, actions, rewards, masks, done_num

    @jit
    def evaluate(self):...
```

✓ 训练策略

✓ 收集数据

MindSpore RL介绍

以DQN为例

✓ 策略网络定义

```python
class DQNPolicy:
    """DQN Policy"""

    def __init__(self, params):
        self.policy_network = FullyConnectedNet(
            params["state_space_dim"],
            params["hidden_size"],
            params["action_space_dim"],
            params["compute_type"],
        )
        self.target_network = FullyConnectedNet(
            params["state_space_dim"],
            params["hidden_size"],
            params["action_space_dim"],
            params["compute_type"],
        )

        self.init_policy = RandomPolicy(params["action_space_dim
        self.collect_policy = EpsilonGreedyPolicy(
            self.policy_network,
            (1, 1),
            params["epsi_high"],
            params["epsi_low"],
            params["decay"],
            params["action_space_dim"],
        )
        self.evaluate_policy = GreedyPolicy(self.policy_network)
```

✓ Actor实现

```python
class DQNActor(Actor):
    def __init__(self, params):
        self._environment = params["collect_environment"]
        self._eval_env = params["eval_environment"]
        self.replay_buffer = params["replay_buffer"]
        self.step = Parameter(Tensor(0, ms.int32), name="step", requires_grad=Fa
        self.expand_dims = P.ExpandDims()
        self.reshape = P.Reshape()
        self.ones = P.Ones()
        self.abs = P.Abs()
        self.assign = P.Assign()
        self.select = P.Select()
        self.reward = Tensor(…
        self.penalty = Tensor(…
        self.print = P.Print()
    def act(self, phase, params):
        """act func"""
        if phase == 1:
            # Fill the replay buffer
            action = self.init_policy()
            new_state, reward, done = self._environment.step(action)
            done = self.expand_dims(done, 0)
            action = self.reshape(action, (1,))
            my_reward = self.select(done, self.penalty, self.reward)
            return done, reward, new_state, action, my_reward
        if phase == 2:
            # Experience collection
            self.step += 1
            ts0 = self.expand_dims(params, 0)
            step_tensor = self.ones((1, 1), ms.float32) * self.step

            action = self.collect_policy(ts0, step_tensor)
            new_state, reward, done = self._environment.step(action)
            done = self.expand_dims(done, 0)
            action = self.reshape(action, (1,))
            my_reward = self.select(done, self.penalty, self.reward)
            return done, reward, new_state, action, my_reward
        if phase == 3:
            # Evaluate the trained policy
            ts0 = self.expand_dims(params, 0)
            action = self.evaluate_policy(ts0)
            new_state, reward, done = self._eval_env.step(action)
            done = self.expand_dims(done, 0)
            return done, reward, new_state
        self.print("Phase is incorrect")
        return 0

    def get_action(self, phase, params):
        """Default get_action function"""
        return
```

✓ Learner实现

```python
class DQNLearner(Learner):
    class PolicyNetWithLossCell(nn.Cell):
        def __init__(self, backbone, loss_fn): …

        def construct(self, x, a0, label):
            """constructor for Loss Cell"""
            out = self._backbone(x)
            out = self.gather(out, 1, a0)
            loss = self._loss_fn(out, label)
            return loss
    def __init__(self, params=None):
        super().__init__()
        self.policy_network = params["policy_network"]
        self.target_network = params["target_network"]
        self.policy_param = ParameterTuple(self.policy_network.get_parameters())
        self.target_ (import) nn: Any uple(self.target_network.get_parameters())
        optimizer = nn.Adam(
            self.policy_network.trainable_params(), learning_rate=params["lr"]
        )
        loss_fn = nn.MSELoss()
        loss_q_net = self.PolicyNetWithLossCell(self.policy_network, loss_fn)
        self.policy_network_train = nn.TrainOneStepCell(loss_q_net, optimizer)
        self.policy_network_train.set_train(mode=True)
        self.gamma = Tensor(params["gamma"], ms.float32)
        self.expand_dims = P.ExpandDims()
        self.reshape = P.Reshape()
        self.hyper_map = C.HyperMap()
        self.ones_like = P.OnesLike()
        self.select = P.Select()
    def learn(self, experience):
        """Model update"""
        s0, a0, r1, s1 = experience
        next_state_values = self.target_network(s1)
        next_state_values = next_state_values.max(axis=1)
        r1 = self.reshape(r1, (-1,))
        y_true = r1 + self.gamma * next_state_values
        # Modify last step reward
        one = self.ones_like(r1)
        y_true = self.select(r1 == -one, one, y_true)
        y_true = self.expand_dims(y_true, 1)

        success = self.policy_network_train(s0, a0, y_true)
        return success
    def update(self):
        """Update the network parameters"""
        assign_result = self.hyper_map(
            _update_opt, self.policy_param, self.target_param
        )
        return assign_result
```

MindSpore RL介绍

中国科学院自动化研究所
INSTITUTE OF AUTOMATION
CHINESE ACADEMY OF SCIENCES

□ 以DQN为例

```
class DQNTrainer(Trainer):
    """DQN Trainer"""
    def __init__(self, msrl, params):
        super(DQNTrainer, self).__init__(msrl)
        self.zero = Tensor(0, ms.float32)
        self.squeeze = P.Squeeze()
        self.less = P.Less()
        self.zero_value = Tensor(0, ms.float32)
        self.fill_value = Tensor(1000, ms.float32)
        self.inited = Parameter(Tensor((False,), ms.bool_), name="init_flag")
        self.mod = P.Mod()
        self.false = Tensor((False,), ms.bool_)
        self.true = Tensor((True,), ms.bool_)
        self.num_evaluate_episode = params["num_evaluate_episode"]
        self.update_period = Tensor(5, ms.float32)
    def trainable_variables(self):...
    @ms.jit
    def init_training(self):
        """Initialize training"""
        state = self.msrl.collect_environment.reset()
        done = self.false
        i = self.zero_value
        while self.less(i, self.fill_value):...
        return done
    @ms.jit
    def train_one_episode(self):
        """Train one episode"""
        if not self.inited:
            self.init_training()
            self.inited = self.true
        state = self.msrl.collect_environment.reset()
        done = self.false
        total_reward = self.zero
        steps = self.zero
        loss = self.zero
        while not done:
            done, r, new_state, action, my_reward = self.msrl.agent_act(
                trainer.COLLECT, state
            )
            self.msrl.replay_buffer_insert([state, action, my_reward, new_state])
            state = new_state
            r = self.squeeze(r)
            loss = self.msrl.agent_learn(self.msrl.replay_buffer_sample())...
            if not self.mod(steps, self.update_period):...
        return loss, total_reward, steps
    @ms.jit
    def evaluate(self):...
```

- ✓ 收集数据，初始化Replay Buffer
- ✓ 环境交互，采集数据
- ✓ 训练策略

MindSpore RL介绍

□ MSRL类

✓ Policy Gradient

```python
class PGTrainer(Trainer):
    """PGTrainer"""

    def __init__(self, msrl, params):···

    def trainable_variables(self):
        """Trainable variables for saving."""
        trainable_variables = {"actor_net": self.msrl.learner.actor_net}
        return trainable_variables

    @jit
    def train_one_episode(self):
        """Train one episode"""
        obs_list, action_list, reward_list, masks, steps = self.run_one_episode()
        loss = self.msrl.agent_learn([obs_list, reward_list, action_list, masks])
        return loss, steps, self.loop_size

    @jit
    def run_one_episode(self):
        """run_one_episode(dynamic list is not supported in graph mode, so use static loop.)"""
        steps = self.zero_value
        done_status = self.zero_value
        done_num = self.zero_value
        masks = self.masks
        obs = self.msrl.collect_environment.reset()
        while steps < self.loop_size:
            self.obs_list.write(steps, obs)
            done, r, obs, a = self.msrl.agent_act(trainer.COLLECT, obs)
            self.action_list.write(steps, a)
            self.reward_list.write(steps, r)···
                self.msrl.collect_environment.reset()
            steps += 1
        states = self.obs_list.stack()
        rewards = self.reward_list.stack()
        actions = self.action_list.stack()
        self.obs_list.clear()
        self.reward_list.clear()
        self.action_list.clear()
        return states, actions, rewards, masks, done_num

    @jit
    def evaluate(self):···
```

✓ DQN

```python
class DQNTrainer(Trainer):
    """DQN Trainer"""
    def __init__(self, msrl, params):
        super(DQNTrainer, self).__init__(msrl)
        self.zero = Tensor(0, ms.float32)
        self.squeeze = P.Squeeze()
        self.less = P.Less()
        self.zero_value = Tensor(0, ms.float32)
        self.fill_value = Tensor(1000, ms.float32)
        self.inited = Parameter(Tensor((False,), ms.bool_), name="init_flag")
        self.mod = P.Mod()
        self.false = Tensor((False,), ms.bool_)
        self.true = Tensor((True,), ms.bool_)
        self.num_evaluate_episode = params["num_evaluate_episode"]
        self.update_period = Tensor(5, ms.float32)
    def trainable_variables(self):···
    @ms.jit
    def init_training(self):
        """Initialize training"""
        state = self.msrl.collect_environment.reset()
        done = self.false
        i = self.zero_value
        while self.less(i, self.fill_value):···
        return done
    @ms.jit
    def train_one_episode(self):
        """Train one episode"""
        if not self.inited:
            self.init_training()
            self.inited = self.true
        state = self.msrl.collect_environment.reset()
        done = self.false
        total_reward = self.zero
        steps = self.zero
        loss = self.zero
        while not done:
            done, r, new_state, action, my_reward = self.msrl.agent_act(
                trainer.COLLECT, state
            )
            self.msrl.replay_buffer_insert([state, action, my_reward, new_state])
            state = new_state
            r = self.squeeze(r)
            loss = self.msrl.agent_learn(self.msrl.replay_buffer_sample())···
            if not self.mod(steps, self.update_period):···
        return loss, total_reward, steps
    @ms.jit
    def evaluate(self):···
```

MindSpore RL介绍

中国科学院自动化研究所
INSTITUTE OF AUTOMATION
CHINESE ACADEMY OF SCIENCES

◻ MSRL类：集成RL算法开发所需要的函数句柄或API

```python
class MSRL(nn.Cell):

    def init(self, config):
        """
        Initialization of MSRL object.
        The function creates all the data/objects that the algorithm requires.
        It also initializes all the function handler.

        Args:
            config (dict): algorithm configuration file.
        """
        # ---------------------- ReplayBuffer ----------------------
        replay_buffer = config.get("replay_buffer")
        if replay_buffer:
            if replay_buffer.get("multi_type_replaybuffer"):
                self.buffers = {}
                for key, item in replay_buffer.items():
                    if key != "multi_type_replaybuffer":
                        self.buffers[key] = self.__create_replay_buffer(item)
            else:
                self.buffers = self.__create_replay_buffer(replay_buffer)
                if replay_buffer.get("number") <= 1:
                    self.replay_buffer_sample = self.buffers.sample
                    self.replay_buffer_insert = self.buffers.insert
                    self.replay_buffer_full = self.buffers.full
                    self.replay_buffer_reset = self.buffers.reset
```

```python
# --------------------- Agent ---------------------
agent_config = config.get("agent")
if not agent_config:
    self._compulsory_items_check(config["actor"], ["number"], "actor")
    num_actors = config["actor"]["number"]
    # We consider eval_env is alwarys shared, so only create one instance whether
    share_env = True
    if "share_env" in config["actor"]:
        share_env = config["actor"]["share_env"]
    # --------------------- Environment ---------------------
    self.collect_environment, self.num_collect_env = MSRL.create_environments(
        config, "collect_environment", deploy_config=self.deploy_config
    )
    need_batched = True if (self.num_collect_env > 1) else False
    self.eval_environment, _ = MSRL.create_environments(
        config,
        "eval_environment",
        need_batched=need_batched,
    )
    # ---------------------------------------------------------
    if self.distributed:...
    else:
        if num_actors == 1:
            self.policy_and_network = self.__create_policy_and_network(config)
            self.actors = self.__create_actor(config, self.policy_and_network)
            self.learner = self.__create_learner(
                config, self.policy_and_network
            )
            self.agent_act = self.actors.act
            self.agent_learn = self.learner.learn
            self.agent_get_action = self.actors.get_action
        elif num_actors > 1:
        else:
            raise ValueError(
                "The number of actors should >= 1, but get ", num_actors
            )
```

MindSpore RL介绍

□ 经验回放池

经验回放

在强化学习中，ReplayBuffer是一个常用的基本数据存储方式，它的功能在于存放智能体与环境交互得到的数据。 使用ReplayBuffer可以解决以下几个问题：

1. 存储的历史经验数据，可以通过采样或一定优先级的方式抽取，以打破训练数据的相关性，使抽样的数据具有独立同分布的特性。
2. 可以提供数据的临时存储，提高数据的利用率。

一般情况下，算法人员使用原生的Python数据结构或Numpy的数据结构来构造ReplayBuffer, 或者一般的强化学习框架也提供了标准的API封装。不同的是，MindSpore实现了设备端的ReplayBuffer结构，一方面能在使用GPU/Ascend硬件时减少数据在Host和Device之间的频繁拷贝，另一方面，以MindSpore算子的形式表达ReplayBuffer，可以构建完整的IR图，使能MindSpore GRAPH_MODE的各种图优化，提升整体的性能。

类别	特性	设备		
		CPU	GPU	Ascend
UniformReplayBuffer	1 FIFO先进先出 2 支持batch 输入	✓	✓	/
PriorityReplayBuffer	1 proportional-based优先级策略 2 Sum Tree提升采样效率	✓	✓	✓
ReservoirReplayBuffer	采用无偏采样	✓	✓	✓

MindSpore RL介绍

□ 全部整合！ →session

```python
class Session:
    """
    The Session is a class for running MindSpore RL algorithms.

    Args:
        alg_config (dict): the algorithm configuration or the deployment configuration of the algorithm.
        deploy_config (dict): the deployment configuration for distribution. Default: ``None``.
            For more details of configuration of algorithm, please have a look at
            `detail <https://www.mindspore.cn/reinforcement/docs/zh-CN/r0.7/custom_config_info.html>`_.
        params (dict): The algorithm specific training parameters. Default: ``None``.
        callbacks (list[Callback]): The callback list. Default: ``None``.
    """

    def __init__(self, alg_config, deploy_config=None, params=None, callbacks=None):
        if alg_config is not None:
            self.msrl = MSRL(alg_config, deploy_config)
        self.params = params
        self.callbacks = callbacks
        self.dist = False
        self.alg_config = alg_config
        if deploy_config:
            self.dist = True
            self.worker_num = deploy_config.get("worker_num")
            self.config = deploy_config.get("config")
            self.dist_policy = deploy_config.get("distribution_policy")
            self.is_auto = deploy_config.get("auto_distribution")
            self.algo_name = deploy_config.get("algo_name")
            self.frag_file = deploy_config.get("fragment_file")
```

```python
def run(self, class_type=None, is_train=True, episode=0, duration=0):
    """
    Execute the reinforcement learning algorithm.

    Args:
        class_type (Trainer): The class type of the algorithm"s trainer class. Default: ``None``.
        is_train (bool): Run the algorithm in train mode or eval mode. Default: ``True``.
        episode (int): The number of episode of the training. Default: ``0``.
        duration (int): The number of duration of the training. Default: ``0``.
    """

    if self.dist:···
    else:
        if self.params is None:
            trainer = class_type(self.msrl)
        else:
            trainer = class_type(self.msrl, self.params)
        ckpt_path = None
        if self.params and "ckpt_path" in self.params:
            ckpt_path = self.params["ckpt_path"]
        if is_train:
            trainer.train(episode, self.callbacks, ckpt_path)
            print("training end")
        else:
            if ckpt_path:
                trainer.load_and_eval(ckpt_path)
                print("eval end")
            else:
                print("Please provide a ckpt_path for eval.")

    # Close the environment to release the resource
    if self.msrl.collect_environment is not None:
        if isinstance(self.msrl.collect_environment, nn.CellList):
            for env in self.msrl.collect_environment:
                env.close()
        else:
            self.msrl.collect_environment.close()
    if self.msrl.eval_environment is not None:
        if isinstance(self.msrl.eval_environment, nn.CellList):
            for env in self.msrl.eval_environment:
                env.close()
        else:
            self.msrl.eval_environment.close()
```

✓ 初始化

✓ 训练（Trainer类中）

MindSpore RL介绍

□ 全部整合！→session

✓ Policy Gradient

```python
class PGSession(Session):
    """PG session"""

    def __init__(self, env_yaml=None, algo_yaml=None):
        update_config(config, env_yaml, algo_yaml)
        params = config.trainer_params
        eval_cb = EvaluateCallback(10)
        time_cb = TimeCallback(10)
        cbs = [time_cb, eval_cb]
        super().__init__(config.algorithm_config, None, params=params, callbacks=cbs)
```

✓ DQN

```python
class DQNSession(Session):
    """DQN session"""

    def __init__(self, env_yaml=None, algo_yaml=None):
        update_config(config, env_yaml, algo_yaml)
        compute_type = (
            mstype.float16
            if context.get_context("device_target") in ["Ascend"]
            else mstype.float32
        )
        config.algorithm_config["policy_and_network"]["params"][
            "compute_type"
        ] = compute_type
        env = config.algorithm_config.get("collect_environment").get("type")(
            config.collect_env_params[
                config.algorithm_config.get("collect_environment").get("type").__name__
            ]
        )
        config.algorithm_config["replay_buffer"]["data_shape"] = [
            env.observation_space.shape,
            (1,),
            (1,),
            env.observation_space.shape,
        ]
        config.algorithm_config["replay_buffer"]["data_type"] = [
            env.observation_space.ms_dtype,
            env.action_space.ms_dtype,
            env.reward_space.ms_dtype,
            env.observation_space.ms_dtype,
        ]
        params = config.trainer_params
        loss_cb = LossCallback()
        ckpt_cb = CheckpointCallback(
            config.trainer_params.get("save_per_episode"),
            config.trainer_params.get("ckpt_path"),
        )
        eval_cb = EvaluateCallback(config.trainer_params.get("eval_per_episode"))
        time_cb = TimeCallback()
        cbs = [loss_cb, ckpt_cb, eval_cb, time_cb]
        super().__init__(config.algorithm_config, None, params=params, callbacks=cbs)
```

MindSpore RL介绍

□ 参数如何传进去？ config.py！

Config文件对应一个算法组件：*actor*、*learner*、*policy*、*replay buffer*和两个*environment*

```
algorithm_config = {
    'actor': {
        'number': 1,                                                # Act
        'type': DQNActor,                                           # 
        'policies': ['init_policy', 'collect_policy', 'evaluate_policy'],
    },
    'learner': {
        'number': 1,                                                # Lea
        'type': DQNLearner,                                         #
        'params': learner_params,
        'networks': ['policy_network', 'target_network']
    },
    'policy_and_network': {
        'type': DQNPolicy,                                          # 
        'params': policy_params
    },
    'collect_environment': {
        'number': 1,                                   # Collect Environment实例的数量
        'type': GymEnvironment,                        # 需要创建的Collect Environment类
        'params': collect_env_params                   # Collect Environment中需要用到的参数
    },
    'eval_environment': {
        'number': 1,                                   # 同Collect Environment
        'type': GymEnvironment,
        'params': eval_env_params
    },
    'replay_buffer': {'number': 1,                     # ReplayBuffer实例的数量
                      'type': ReplayBuffer,            # 需要创建的ReplayBuffer类
                      'capacity': 100000,              # ReplayBuffer大小
                      'data_shape': [(4,), (1,), (1,), (4,)],          # ReplayBuffer中的数据Shape
                      'data_type': [ms.float32, ms.int32, ms.float32, ms.float32],  # ReplayBuffer中的数据Type
                      'sample_size': 64},              # ReplayBuffer单次采样的数据量
}
```

MindSpore RL介绍

□ 完成入口文件

```python
# pylint: disable=C0413
import argparse

from mindspore import context
from mindspore import dtype as mstype

from mindspore_rl.algorithm.pg import config
from mindspore_rl.algorithm.pg.pg_session import PGSession
from mindspore_rl.algorithm.pg.pg_trainer import PGTrainer

parser = argparse.ArgumentParser(description="MindSpore Reinforcement PG")
parser.add_argument("--episode", type=int, default=600, help="total episode numbers.")
parser.add_argument(…
parser.add_argument(…
parser.add_argument(…
parser.add_argument(…
options, _ = parser.parse_known_args()


def train(episode=options.episode):
    """PG train entry."""
    if options.device_target != "Auto":
        context.set_context(device_target=options.device_target)
    compute_type = (
        mstype.float32 if options.precision_mode == "fp32" else mstype.float16
    )
    config.algorithm_config["policy_and_network"]["params"][
        "compute_type"
    ] = compute_type
    if compute_type == mstype.float16 and options.device_target != "Ascend":
        raise ValueError("Fp16 mode is supgrted by Ascend backend.")
    duration = config.trainer_params.get("duration")
    context.set_context(mode=context.GRAPH_MODE, max_call_depth=100000)

    pg_session = PGSession(options.env_yaml, options.algo_yaml)
    pg_session.run(class_type=PGTrainer, episode=episode, duration=duration)


if __name__ == "__main__":
    train()
```

```python
import argparse
from mindspore_rl.algorithm.dqn import config
from mindspore_rl.algorithm.dqn.dqn_session import DQNSession
from mindspore_rl.algorithm.dqn.dqn_trainer import DQNTrainer
from mindspore import context
from mindspore import dtype as mstype

parser = argparse.ArgumentParser(description='MindSpore Reinforcement DQN')
parser.add_argument('--episode', type=int, default=650, help='total episode numbers.')
parser.add_argument('--device_target', type=str, default='Auto', choices=['Ascend', 'CPU', 'GPU', 'Auto'],
                    help='Choose a device to run the dqn example(Default: Auto).')
parser.add_argument('--precision_mode', type=str, default='fp32', choices=['fp32', 'fp16'],
                    help='Precision mode')
parser.add_argument('--env_yaml', type=str, default='../env_yaml/CartPole-v0.yaml',
                    help='Choose an environment yaml to update the dqn example(Default: CartPole-v0.yaml).')
parser.add_argument('--algo_yaml', type=str, default=None,
                    help='Choose an algo yaml to update the dqn example(Default: None).')
options, _ = parser.parse_known_args()


def train(episode=options.episode):
    """start to train dqn algorithm"""
    if options.device_target != 'Auto':
        context.set_context(device_target=options.device_target)
    if context.get_context('device_target') in ['CPU']:
        context.set_context(enable_graph_kernel=True)
    context.set_context(mode=context.GRAPH_MODE)
    compute_type = mstype.float32 if options.precision_mode == 'fp32' else mstype.float16
    config.algorithm_config['policy_and_network']['params']['compute_type'] = compute_type
    if compute_type == mstype.float16 and options.device_target != 'Ascend':
        raise ValueError("Fp16 mode is supported by Ascend backend.")
    dqn_session = DQNSession(options.env_yaml, options.algo_yaml)
    dqn_session.run(class_type=DQNTrainer, episode=episode)


if __name__ == "__main__":
    train()
```

MindSpore RL介绍

□ 如何实现自己的算法?

➢ **修改网络结构：**

可以在XXXPolicy中（如DQN就是在dqn.py下的DQNPolicy）修改对应Network的结构或者定义自己的Network。

➢ **实现新的RL算法：**

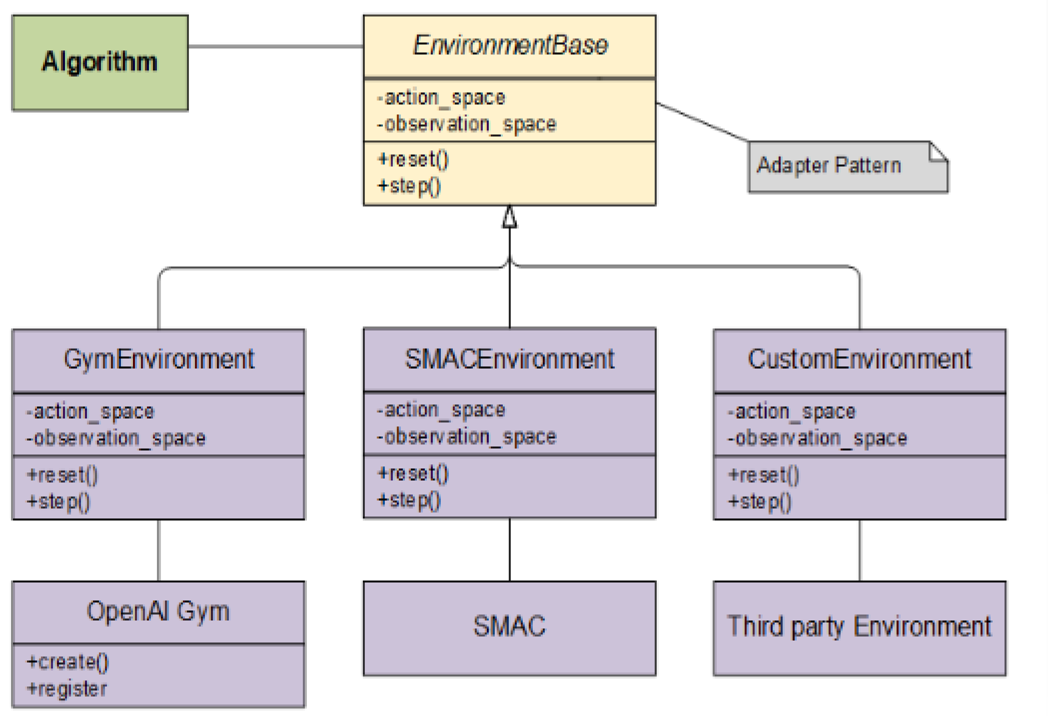
1. 重写Actor和Learner类：以DQN举例就是DQNActor和DQNLearner，以及他们所需要的网络和收集经验的策略CollectPolicy/EvalPolicy（在dqn.py下）。
2. 重写Trainer类：例如DQNTrainer中的train_one_episode方法（在dqn_trainer.py下）。
3. 配置运行时依赖config.py：指定需要用到哪个Actor类，哪个Learner类，ReplayBuffer是否需要，需要用到哪个Environment等。
4. 全部整合定义新的Session：实现DQNSession来配置一些Callback和通过预创建环境来获取ReplayBuffer所需要的shape和dtype
5. 实现执行入口文件（train.py）来配置一些相关的参数

中国科学院自动化研究所
INSTITUTE OF AUTOMATION CHINESE ACADEMY OF SCIENCES

MindSpore RL介绍

算法	RL版本	动作空间		设备			示例环境
		离散	连续	CPU	GPU	Ascend	
DQN	>= 0.1	✔	/	✔	✔	✔	CartPole-v0
PPO	>= 0.1	/	✔	✔	✔	✔	HalfCheetah-v2
AC	>= 0.1	✔	/	✔	✔	✔	CartPole-v0
A2C	>= 0.2	✔	/	✔	✔	✔	CartPole-v0
DDPG	>= 0.3	/	✔	✔	✔	✔	HalfCheetah-v2
QMIX	>= 0.5	✔	/	✔	✔	✔	SMAC, Simple Spread
SAC	>= 0.5	/	✔	✔	✔	✔	HalfCheetah-v2
TD3	>= 0.6	/	✔	✔	✔	✔	HalfCheetah-v2
C51	>= 0.6	✔	/	✔	✔	✔	CartPole-v0
A3C	>= 0.6	✔	/	/	✔	✔	CartPole-v0
CQL	>= 0.6	/	✔	✔	✔	✔	Hopper-v0
MAPPO	>= 0.6	✔	/	✔	✔	✔	Simple Spread
GAIL	>= 0.6	/	✔	✔	✔	✔	HalfCheetah-v2
MCTS	>= 0.6	✔	/	✔	✔	/	Tic-Tac-Toe
AWAC	>= 0.6	/	✔	✔	✔	✔	Ant-v2
Dreamer	>= 0.6	/	✔	/	✔	✔	Walker-walk
IQL	>= 0.6	/	✔	✔	✔	✔	Walker2d-v2
MADDPG	>= 0.6	✔	/	✔	✔	✔	simple_spread
Double DQN	>= 0.6	✔	/	✔	✔	✔	CartPole-v0
Policy Gradient	>= 0.6	✔	/	✔	✔	✔	CartPole-v0
Dueling DQN	>= 0.6	✔	/	✔	✔	✔	CartPole-v0

环境	版本
Gym	>= v0.1
MuJoCo	>= v0.1
MPE	>= v0.6
SMAC	>= v0.5
DMC	>= v0.6
>= v0.6	
D4RL	>= v0.6



中国科学院
自动化研究所
INSTITUTE OF AUTOMATION
CHINESE ACADEMY OF SCIENCES

谢谢！