# 白杨智能体训练云 V2

## 智能体训练（最佳实践）应用指南 2.2



**2023.07.01**

**白杨时代**

# 目 录

# 1. 基本介绍

“最佳实践”作为智能体训练云平台算法训练层的核心组件，沉淀了经过星际争霸 2 指挥官模式所积累的综合算法技术。该组件的设计初衷是将经过验证的、能够稳定获得效果的深度强化学习算法模版化，作为默认算法基线提供给用户直接调用。另一方面，该设计将业务代码与算法代码解耦，大幅度降低用户对该技术的使用成本和算法门槛。

最佳实践组件相关的代码包含智能体配置（config）、仿真对接（env）以及 Pipeline 开发（pipeline）等三部分。Pipeline 是连接 Env 和 Agent 的桥梁，它会处理 Env 和 Agent 输出的数据，使得二者之间的沟通更简洁与统一。

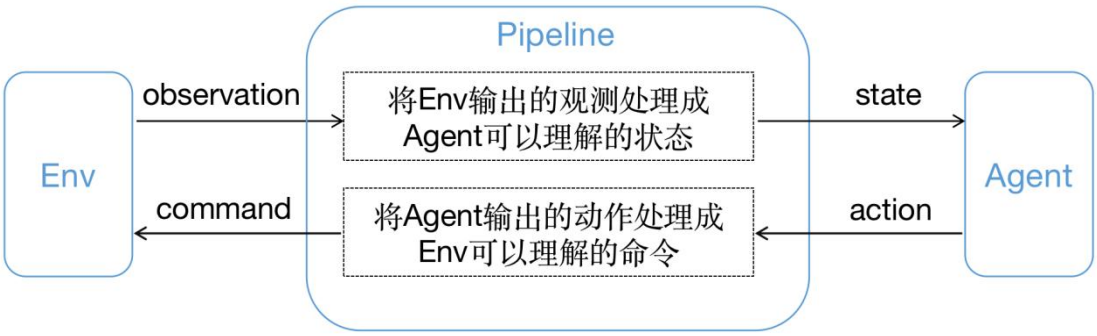


图 1.1 最佳实践架构简图

本手册将介绍上述三部分的功能，并在最后给出若干框架使用建议和其他参考资料。

# 2. 智能体配置

config.py 文件用于智能体配置，该文件用于配置智能体推演和智能体训练的相关参数，在创建训练任务时需要指定。当 config.py 文件中的内容过于庞大时，可以对其进行拆分（如默认样例中的 training_config.py、network_config.py、training_config.py）。

该部分主要分为核心函数/类说明、训练参数以及样例模版等三个部分。其中，核心类/函数说明中主要围绕特征处理和网络构建进行展开，涉及函数/类 PlainFeature、VectorFeature、RangedFeature、EntityEncoder、CommonEncoder 等；训练参数主要围绕训练任务核心参数进行展开，包括 env_num、fragment_size、replay_size、sample_batch_size 等。

## 2.1 核心类/函数说明

表 2.1 CommonFeatureSet 功能说明

| **类名称** | CommonFeatureSet |
|---|---|
| **引用路径** | drill.feature |
| **功能说明** | 通用特征模版。作为相应数据类型的载体，用于提取通用特征。通用特征的特点是数据定长，数据输入的 shape 固定。 |
| **初始化参数** | name: str, feature_dict: dict |

表 2.2 CommonFeatureSet 参数说明

| **参数名称** | **参数类型** | **参数描述** |
|---|---|---|
| name | str | 字符串数据，用于标识模版名称。 |

| | | |
|---|---|---|
| feature_dict | dict | 字典数据，其中字典的 key 为字符串，value 为 FeatureType 类 |

表 2.3 EntityFeatureSet 功能说明

| | |
|---|---|
| **类名称** | EntityFeatureSet |
| **引用路径** | drill.feature |
| **功能说明** | 实体特征模版。用于提取实体特征信息，实体特征不同于通用特征，其长度可变。 |
| **初始化参数** | name: str, max_length: int, feature_dict: dict |

表 2.4 EntityFeatureSet 参数说明

| **参数名称** | **参数类型** | **参数描述** |
|---|---|---|
| name | str | 字符串数据，用于标识模版名称。 |
| max_length | int | 实体的最大数量限制。 |
| feature_dict | dict | 字典数据，其中字典的 key 为字符串，value 为 FeatureType 类 |

表 2.5 SpatialFeatureSet 功能说明

| | |
|---|---|
| **类名称** | SpatialFeatureSet |
| **引用路径** | drill.feature |

| | |
|---|---|
| **功能说明** | 空间特征模版。用于提取高维特征信息，如图像等。 |
| **初始化参数** | name: str, feature_dict: dict, shape: tuple, channel: str |

表 2.6 SpatialFeatureSet 参数说明

| **参数名称** | **参数类型** | **参数描述** |
|---|---|---|
| name | str | 字符串数据，用于标识模版名称。 |
| feature_dict | dict | 字典数据，其中字典的 key 为字符串，value 为 FeatureType 类 |
| shape | tuple | 元组数据，除了 channel 的其他 shape 信息 |
| channel | str | 字符串数据，指定 channel 的位置，仅可为’channel_first’或’channel_last’，默认为’channel_last’。 |

表 2.7 FeatureSet 功能说明

| | |
|---|---|
| **类名称** | FeatureSet |
| **引用路径** | drill.feature |
| **功能说明** | 特征模版的基类。用户可以继承该类，实现自己的特征模板。但是不可以实例化该类！ |
| **初始化参数** | name: str, feature_dict: dict, shape: tuple |

表 2.8 FeatureSet 参数说明

| 参数名称 | 参数类型 | 参数描述 |
| --- | --- | --- |
| name | str | 字符串数据，用于标识模版名称。 |
| feature_dict | dict | 字典数据，其中字典的 key 为字符串，value 为 FeatureType 类 |
| shape | tuple | 元组数据，除了 channel 的其他 shape 信息 |

表 2.9 PlainFeature 功能说明

| 类名称 | PlainFeature |
| --- | --- |
| 引用路径 | drill.feature |
| 功能说明 | 单数据特征类型。用于表示单个分离的数据，推演运行时，本类定义的特征类型支持单个的浮点型数据输入。 |
| 初始化参数 | 无 |

表 2.10 VectorFeature 功能说明

| 类名称 | VectorFeature |
| --- | --- |
| 引用路径 | drill.feature |
| 功能说明 | 向量特征类型。用于处理特征信息。在推演运行时，本类定义的特征类型支持向量输入，输入向量长度需要和初始化参数 length 相同。 |
| 初始化参数 | length：int |

表 2.11 VectorFeature 参数说明

| 参数名称 | 参数类型 | 参数描述 |
| --- | --- | --- |
| length | int | 向量特征维度。 |

表 2.12 OnehotFeature 功能说明

| 类名称 | OnehotFeature |
| --- | --- |
| 引用路径 | drill.feature |
| 功能说明 | 用于处理类别型数据，如地面单位-0、空中单位-1、海上单位-2等。 |
| 初始化参数 | depth：int |

表 2.13 OnehotFeature 参数说明

| 参数名称 | 参数类型 | 参数描述 |
| --- | --- | --- |
| depth | int | 类别数量上限。 |

表 2.14 RangedFeature 功能说明

| 类名称 | RangedFeature |
| --- | --- |
| 引用路径 | drill.feature |

| | |
|---|---|
| **功能说明** | 相比 PlainFeature 处理，额外引入归一化操作。 |
| **初始化参数** | low: float, high: float |

表 2.15 RangedFeature 参数说明

| **参数名称** | **参数类型** | **参数描述** |
|---|---|---|
| low | float | 输入数值下限。 |
| high | float | 输入数值上限。 |

表 2.16 BinaryhotFeature 功能说明

| | |
|---|---|
| **类名称** | BinaryhotFeature |
| **引用路径** | drill.feature |
| **功能说明** | 将一个整数转换为一个二进制的表示向量。 |
| **初始化参数** | depth：int |

表 2.17 BinaryhotFeature 参数说明

| **参数名称** | **参数类型** | **参数描述** |
|---|---|---|
| depth | int | 表示当前 Feature 能表示的最大值，<br>最大为 2^depth - 1 |

表 2.18 RepeatFeature 功能说明

| **类名称** | RepeatFeature |
| --- | --- |
| **引用路径** | drill.feature |
| **功能说明** | 重复在内部创建多个特征，然后把这些特征的结果合并。 |
| **初始化参数** | max_length: int, sub_features: Dict[str, CellFeature] |

表 2.19 RepeatFeature 参数说明

| **参数名称** | **参数类型** | **参数描述** |
| --- | --- | --- |
| max_length | int | 特征的最大长度。处理时未达到最大值，会自动填充 0 |
| sub_features | Dict[str, CellFeature] | 被用来重复创建和处理数据的特征 |

表 2.20 CellFeature 功能说明

| **类名称** | CellFeature |
| --- | --- |
| **引用路径** | drill.feature |
| **功能说明** | 特征的基类。用户可继承该类实现自定义的特征。 |
| **初始化参数** | length：int |

表 2.21 CellFeature 参数说明

| 参数名称 | 参数类型 | 参数描述 |
| --- | --- | --- |
| length | int | 当前 Feature 在超平面上的堆叠维度上面的长度 |

表 2.22 CommonEncoder 功能说明

| 类名称 | CommonEncoder |
| --- | --- |
| 引用路径 | drill.model.tf.network.encoder |
| 功能说明 | commonfeature 对应的编码器，对定长型输入数据进行编码操作。 |
| 初始化参数 | hidden_layer_sizes: list |

表 2.23 CommonEncoder 参数说明

| 参数名称 | 参数类型 | 参数描述 |
| --- | --- | --- |
| hidden_layer_sizes | list | 编码器隐藏层节点数，例[256, 256]。 |

表 2.24 EntityEncoder 功能说明

| 类名称 | EntityEncoder |
| --- | --- |
| 引用路径 | drill.model.tf.network.encoder |
| 功能说明 | entityfeature 对应的编码器，对实体型输入数据进行编码操作。 |

| | |
|---|---|
| **初始化参数** | hidden_layer_sizes: list, transformer: class, pooling: class |

表 2.25 EntityEncoder 参数说明

| **参数名称** | **参数类型** | **参数描述** |
|---|---|---|
| hidden_layer_sizes | list | 隐藏层节点数，例如[256, 256]。 |
| transformer | class | transformer 模块，默认为 None |
| pooling | class | encoder 的下采样方法，目前支持’ encoder_pooling. Max()’或’ encoder_pooling. Attention()’， 默认为 Max()。 |

表 2.26 SpatialEncoder 功能说明

| | |
|---|---|
| **类名称** | SpatialEncoder |
| **引用路径** | drill.model.tf.network.encoder |
| **功能说明** | SpatialEncoder 对应的编码器，对空间型输入数据进行编码操作。 |
| **初始化参数** | projected_channel_num: int, output_size: int, down_samples: list, res_block_num: int |

表 2.27 SpatialEncoder 参数说明

| **参数名称** | **参数类型** | **参数描述** |
|---|---|---|
| projected_channel_num | int | 输入层维度 |

| | | |
|---|---|---|
| output_size | int | 输出层维度 |
| down_samples | list | 卷积层的参数，包括[filter, kernel_size, stride,padding]，可以多层列表嵌套，默认为 None |
| res_block_num | int | resnet 块的个数，默认为 4 |

表 2.28 DenseAggregator 功能说明

| | |
|---|---|
| **类名称** | DenseAggregator |
| **引用路径** | drill.model.tf.network.aggregator |
| **功能说明** | 用于聚合多个 encoder 的输出，如果有 rnn 的话，通常放在这个组件，DenseAggregator 不具有时序信息捕捉能力。 |
| **初始化参数** | hidden_layer_sizes: list, output_size: int |

表 2.29 DenseAggregator 参数说明

| **参数名称** | **参数类型** | **参数描述** |
|---|---|---|
| hidden_layer_sizes | list | 隐藏层节点数，例[256, 256]。 |
| output_size | int | 输出层维度。 |

表 2.30 GRUAggregator 功能说明

| | |
|---|---|
| **类名称** | GRUAggregator |

| | |
|---|---|
| **引用路径** | drill.model.tf.network.aggregator |
| **功能说明** | 速度较快,如果数据集不大,建议使用 GRU 能使收敛速度更快。 |
| **初始化参数** | hidden_layer_sizes: list, state_size: int, output_size: int |

表 2.31 GRUAggregator 参数说明

| 参数名称 | 参数类型 | 参数描述 |
|---|---|---|
| hidden_layer_sizes | list | 隐藏层节点数，例[256, 256]。 |
| state_size | int | GRU 对应层的状态空间大小。 |
| output_size | int | 输出层维度。 |

表 2.32 LSTMAggregator 功能说明

| | |
|---|---|
| **类名称** | LSTMAggregator |
| **引用路径** | drill.model.tf.network.aggregator |
| **功能说明** | 相较于 GRU 多了两个门，参数较多，训练起来比较困难，需要用户自行斟酌。 |
| **初始化参数** | hidden_layer_sizes: list, state_size: int, output_size: int |

表 2.33 LSTMAggregator 参数说明

| 参数名称 | 参数类型 | 参数描述 |
|---|---|---|

| | | |
|---|---|---|
| hidden_layer_sizes | list | 隐藏层节点数，例[256, 256]。 |
| state_size | int | LSTM 对应层的状态空间大小。 |
| output_size | int | 输出层维度。 |

表 2.34 CategoricalDecoder 功能说明

| | |
|---|---|
| **类名称** | CategoricalDecoder |
| **引用路径** | drill.model.tf.network.decoder |
| **功能说明** | 通常作为整个网络的出口，可以存在多个 decoder。Decoder 的主要作用是作出决策，输出 action。CategoricalDecoder 为用于处理离散动作的解码器。 |
| **初始化参数** | n: int, hidden_layer_sizes: list, activation: str, temperature: float |

表 2.35 CategoricalDecoder 参数说明

| **参数名称** | **参数类型** | **参数描述** |
|---|---|---|
| n | int | 动作数量。 |
| hidden_layer_sizes | list | 隐藏层输出神经元的数量。 |
| activation | str | 激活函数，默认为’relu’ |
| temperature | float | 采样的温度系数，默认为 1 |

表 2.36 GaussianDecoder 功能说明

| 类名称 | GaussianDecoder |
| --- | --- |
| 引用路径 | drill.model.tf.network.decoder |
| 功能说明 | 通常作为整个网络的出口，可以存在多个 decoder。Decoder 的主要作用是作出决策，输出 action。GaussianDecoder 为用于处理连续动作的解码器。 |
| 初始化参数 | n: int, hidden_layer_sizes: list, activation: str |

表 2.37 GaussianDecoder 参数说明

| 参数名称 | 参数类型 | 参数描述 |
| --- | --- | --- |
| n | int | 动作数量。 |
| hidden_layer_sizes | list | 隐藏层输出神经元的数量。 |
| activation | str | 激活函数，默认为’relu’ |

表 2.38 SingleSelectiveDecoder 功能说明

| 类名称 | SingleSelectiveDecoder |
| --- | --- |
| 引用路径 | drill.model.tf.network.decoder |
| 功能说明 | SingleSelectiveDecoder 为用于处理单个单位选择的解码器。底层基于注意力网络实现。 |
| 初始化参数 | attention_size: int, temperature: float |

表 2.39 SingleSelectiveDecoder 参数说明

| 参数名称 | 参数类型 | 参数描述 |
| --- | --- | --- |
| attention_size | int | 注意力机制中 query 和 key 的 size 大小。 |
| temperature | float | 采样温度系数，默认为 1.0。 |

表 2.40 OrderedMultipleSelectiveDecoder 功能说明

| | |
| --- | --- |
| **类名称** | OrderedMultipleSelectiveDecoder |
| **引用路径** | drill.model.tf.network.decoder |
| **功能说明** | OrderedMultipleSelectiveDecoder 为用于处理有序选择多个单位的解码器。 |
| **初始化参数** | max_count: int, pooling: class |

表 2.41 OrderedMultipleSelectiveDecoder 参数说明

| 参数名称 | 参数类型 | 参数描述 |
| --- | --- | --- |
| max_count | int | 最大选择的单位数量。 |
| pooling | class | decoder 的下采样方法，目前支持’decoder_pooling.MeanMax()’或’decoder_pooling.Attention()’，默认为 MeanMax()。 |

表 2.42 UnorderedMultiSelectiveDecoder 功能说明

| **类名称** | UnorderedMultiSelectiveDecoder |
|---|---|
| **引用路径** | drill.model.tf.network.decoder |
| **功能说明** | UnorderedMultiSelectiveDecoder 为用于处理无序选择多个单位的解码器。 |
| **初始化参数** | attention_size: int, pooling: class, temperature: float |

表 2.43 UnorderedMultiSelectiveDecoder 参数说明

| **参数名称** | **参数类型** | **参数描述** |
|---|---|---|
| attention_size | int | 注意力机制中 query 和 key 的 size 大小。 |
| pooling | class | 下采样方法，目前支持’ decoder_pooling.MeanMax()’或’ decoder_pooling.Attention()’，默认为 MeanMax()。 |
| temperature | float | 采样温度系数，默认为 1.0。 |

表 2.44 ValueApproximator 功能说明

| **类名称** | ValueApproximator |
|---|---|
| **引用路径** | drill.model.tf.network.layer |

| | |
|---|---|
| **功能说明** | 可以输出 V(s) 的 value network。 |
| **初始化参数** | hidden_layer_sizes: list |

表 2.45 ValueApproximator 参数说明

| 参数名称 | 参数类型 | 参数描述 |
|---|---|---|
| hidden_layer_sizes | list | Value network 的隐藏层大小。 |

表 2.46 CommanderNetwork 功能说明

| | |
|---|---|
| **类名称** | CommanderNetwork |
| **引用路径** | drill.model.tf.network.commander |
| **功能说明** | CommanderAgent 与 Env 交互时底层使用的网络结构，支持谓语、主语、宾语场景的想定。 |
| **初始化参数** | encoder_config: dict, aggregator_config: dict, decoder_config: dict, value_approximator_config: dict, share_critic: bool |

表 2.47 CommanderNetwork 参数说明

| 参数名称 | 参数类型 | 参数描述 |
|---|---|---|
| encoder_config | dict | encoder 部分的网络配置，以 dict 类型存储。 |
| aggregator_config | dict | aggregator 部分的网络配置，以 dict 类型存 |

| | | |
|---|---|---|
| | | 储。 |
| decoder_config | dict | decoder 部分的网络配置，以 dict 类型存储。 |
| value_approximator_config | dict | value_approximator 部分的网络配置，以 dict 类型存储。 |
| share_critic | bool | 是否共享 Critic 网络，默认为 True。 |

表 2.48 CommanderModelPPO 功能说明

| **类名称** | CommanderModelPPO |
|---|---|
| **引用路径** | drill.model.tf.commander_model_ppo |
| **功能说明** | PPO 算法的实现 |
| **初始化参数** | net: CommanderNetwork, learning_rate: float, clip_param: float, vf_clip_param: float, vf_loss_coef: float, entropy_coef: float, sync: bool, sync_interval: int |

表 2.49 CommanderModelPPO 参数说明

| **参数名称** | **参数类型** | **参数描述** |
|---|---|---|
| net | CommanderNetwork | PPO 算法的网络模型 |
| learning_rate | float | PPO 算法的学习率 |
| clip_param | float | actor 网络的裁剪值，用于限制新旧策略网 |

| | | |
|---|---|---|
| | | 络输出的差距 |
| vf_clip_param | float | critic 网络的裁剪值，用于限制新旧价值网络输出的差距 |
| vf_loss_coef | float | critic 网络损失值的放缩系数，为 1 则不放缩 |
| entropy_coef | float | actor 网络的熵损失值的放缩系数，为 1 则不放缩 |
| sync | bool | 同步模式的开关，默认为 False，即默认为异步模式 |
| sync_interval | int | 同步模式下，模型同步间隔，默认为 10 次权重更新-同步模型 1 次；异步模式下不生效。 |

## 2.2 参数分析

表 2.50 flow_config 参数说明

| 参数名称 | 参数类型 | 功能注释 |
|---|---|---|
| algorithm | dict | 选择算法，目前最佳实践标准化流程推荐 PPO。 |
| builder | class | 传入当前训练任务需要使用的 builder（训练任务入口）。 |
| actor_config | dict | 见下表 |

表 2.51 algorithm 参数说明

| 参数名称 | 参数类型 | 功能注释 |
| --- | --- | --- |
| flow_env | class | drill.flow 的 env 接口 |
| flow_model | class | drill.flow 的 model 接口 |

表 2.52 builder 参数说明（builder 是 BPBuilder 的一个实例）

| 参数名称 | 参数类型 | 功能注释 |
| --- | --- | --- |
| agents | dict | 用于将 agent 名称与 model 和 pipeline 名称一一对应。 |
| models | dict | 用于配置强化学习算法和神经网络模型的参数。 |
| env | dict | 用于对接用户继承实现的 env 接口类。 |
| pipeline | dict | 用于配置 agent_pipeline 和 global_pipeline，并对相关特征模版和处理函数进行对应。 |

表 2.53 actor_config 参数说明

| 参数名称 | 参数类型 | 功能注释 |
| --- | --- | --- |
| [actor_name] | dict | actor 引擎名称 |
| training_models | list[dict] | 配置需要训练的模型，多个模型通 |

| | | |
|---|---|---|
| | | 过列表方式导入 |
| inference_models | list | 配置只跑前向的模型，inference_models 列表中的模型仅进行模型推理，不参与训练 |
| env_num | int | 仿真环境数量 |
| extra_info | dict | extra_info 中增加 actor 描述，extra_info 支持字典格式对描述信息进行定义，可通过区分 extra_info 来实现"训练-验证"相独立 |

表 2.54 training_models 参数说明

| **参数名称** | **参数类型** | **功能注释** |
|---|---|---|
| model_names | list | 参与训练的模型名称。 |
| fragment_size | int | fragment 长度，到达该长度后会使用 enhance_fragment。 |
| replay_size | int | 将 fragment 切片成 replay_size 大小的 replay，并打乱。 |
| sample_batch_size | int | replay buffer 中 sample 单个 batch 中的 replays 数量。 |
| max_data_reuse | int | learner 在收集到下一个 batch 的数据前，最多对当前 batch 额外重复训练多少次。 |

| | | |
|---|---|---|
| putback_replays | bool | Replay buffer 采样后是否放回。 |
| sample_mode | str | Replay buffer 采样方式，支持’LIFO’和’RANDOM’两种模式。 |
| replay_buffer_size | int | buffer 中的 batch_size 数量上限，LIFO 模式默认为 16，RANDOM 模式下默认为 64。 |

表 2.55 env 参数说明

| 参数名称 | 参数类型 | 功能注释 |
|---|---|---|
| class | class | 继承 drill.env.Env 实现的环境接口类，具体说明见下文的环境对接 |
| params | dict | 参数字典，默认为空 |

表 2.56 pipeline 参数说明

| 参数名称 | 参数类型 | 功能注释 |
|---|---|---|
| [pipeline_name] | dict | agent_pipeline 的配置。 |
| global | dict | 全局 pipeline，用于控制 agent_pipeline 的数据流，为可选配置。 |

表 2.57 agent_pipeline 参数说明

| 参数名称 | 参数类型 | 功能注释 |
|---|---|---|

| | | |
|---|---|---|
| class | AgentPipeline | drill 内置的 agent_pipeline 类 |
| params | dict | 全局 pipeline，用于控制 agent_pipeline 的数据流，为可选配置。 |
| handler_dict | dict | 为 params 字典内的参数，用于指定 handler 函数，具体用法参考样例。 |
| batch_algo | str 或 Callable | 为 params 字典内的参数，默认为'gae'，用于计算 value。 |
| batch_config | dict | 为 params 字典内的参数，用于指定 batch_algo 的参数。默认'gae'的参数包括 gamma 和 lamb，具体用法参考样例。 |

## 2.3 样例模版

```
from drill.feature import VectorFeature, PlainFeature, OnehotFeature, RangedFeature

from drill.feature import CommonFeatureSet, EntityFeatureSet

from drill.model.tf.network.encoder import EntityEncoder, CommonEncoder, encoder_pooling

from drill.model.tf.network.aggregator import GRUAggregator, DenseAggregator, LSTMAggregator

from drill.model.tf.network.decoder import CategoricalDecoder, SingleSelectiveDecoder, UnorderedMultiSelectiveDecoder, \
    OrderedMultipleSelectiveDecoder, GaussianDecoder

from drill.model.tf.network.layer import ValueApproximator

from drill.model.tf.network.commander import CommanderNetwork

from drill.model.tf import CommanderModelPPO
```

```
"""
常用特征模板
1.common_feature_set = CommonFeatureSet(name: str, feature_dict: dict)
    通用特征模版。作为相应数据类型的载体，用于提取通用特征。
    通用特征的特点是数据定长，数据输入的 shape 固定。

2.EntityFeatureSet(name: str, max_length: int, feature_dict: dict)
    实体特征模版。用于提取实体特征信息。
    实体特征不同于通用特征，其长度可变。在推演中由于毁伤或不完全观测等原因，导致实体信息形状不固定；而本特征模版则用于处理这类的变长信息。

常用特征
1.PlainFeature:
    单数据特征类型。用于表示单个连续的数据。
    例如：一些经度、纬度、高度、血量、距离、都可以用此种特征表示。
2.VectorFeature(length: int):
    向量特征类型。用于处理特征信息。在推演运行时，本类定义的特征类型支持向量输入，输入向量长度需要和初始化参数 length 相同
    例如：高纬的坐标信息。
3.RangedFeature(low: float, high: float, length: int)
    用于处理需要归一化的特征信息。在推演运行时，本类定义的特征类型支持单数据和向量类型输入。输入后会统一对数据做归一化处理。
    归一化处理方式是将每一个数据 x 都进行（x-最小值）/（最大值-最小值）的处理
4.OnehotFeature(depth: int)
    用于处理类别信息，可以将一个整数转换为一个 one-hot 的表示向量。
    例如：男女性别、存活状态。
"""

# feature 配置
entity_feature_set = EntityFeatureSet(
    name='my_units',
```

```
    max_length=4,
    feature_dict={
        "pos": VectorFeature(3),        # 坐标信息
        'theta': PlainFeature(),        # 转角信息
        'v': PlainFeature(),           # 二维平面速度
        'alpha_max': PlainFeature(),    # alpha 转角限制
        'theta_max': PlainFeature(),    # theta 转角限制
        'dv': PlainFeature(),          # 垂直（深度）速度
        'radar': VectorFeature(3),      # 雷达射线正前方最远点坐标信息
    },
)
target_feature_set = CommonFeatureSet(
    name='b_info',
    feature_dict={
        'b_pos': VectorFeature(3),      # 坐标信息
        'b_visible': PlainFeature(),    # 是否可见，该类信息可根据场景调整为 onehot 型特征
    }
)

# feature 输入
encoders = {
    "entity_encoder": {
        "class": EntityEncoder,
        "params": {
            "hidden_layer_sizes": [256, 128],   # 隐藏层参数
            "transformer": None,              # transformer 参数
            "pooling": encoder_pooling.Max(),   # 池化方法，可选 Max() 或 Attention(num_query, num_head, head_size)
        },
        "inputs": entity_feature_set           # 输入特征配置
    },
    "common_encoder": {
```

```
        "class": CommonEncoder,
        "params": {
            "hidden_layer_sizes": [256, 128],
        },
        "inputs": target_feature_set
    }
}

# feature 聚合器参数配置，通常可选用 GRUAggregator 或 DenseAggregator
aggregator = {
    "class": GRUAggregator,
    "params": {
        "hidden_layer_sizes": [512, 256],
        "state_size": 64,
        "output_size": 512,
        "seq_len": 1,
    }
}
# aggregator = {
#     "class": DenseAggregator,
#     "params": {
#         "hidden_layer_sizes": [512, 256],
#         "output_size": 512,
#     }
# }

# action 输出
decoders = {
    "action_x": {
        "class": CategoricalDecoder,           # 离散动作选择
        "params": {
            "n": 10,                           # 动作数量
```

```
            "hidden_layer_sizes": [512, 256],    # 解码器隐藏层参数
            # "activation": 'relu',
            # "temperature": 1.0,
        },
        # "mask": None,
        # "dependency": None,
    },
    "action_y": {
        "class": CategoricalDecoder,
        "params": {
            "n": 10,
            "hidden_layer_sizes": [512, 256]
        }
    },
    "action_dv": {
        "class": CategoricalDecoder,
        "params": {
            "n": 10,
            "hidden_layer_sizes": [512, 256]
        }
    }
}

# critic 参数配置
value_layer = {
    "class": ValueApproximator,
    "params": {
        "hidden_layer_sizes": [64, 32]
    }
}

# network 配置，将定义好的编码器、聚合器、解码器、价值评估网络进行组合
```

```
network = {
    "class": CommanderNetwork,
    "params": {
        "encoder_config": encoders,
        "aggregator_config": aggregator,
        "decoder_config": decoders,
        "value_approximator_config": value_layer,
    }
}


# model 配置
model_config = {
    "f4v1_model": {
        "class": CommanderModelPPO,     # 选用最佳实践推荐的模型，基于 ppo 的 CommanderModel
        "params": {
            "network": network,         # 神经网络结构
            # "sync": False,            # 2152：是否开启同步模式，默认为 False，即异步训练；True 则为同步训练
            # "sync_interval": 10,      # 2152：若选择同步训练模式，则需设定模型同步间隔，默认为 10 次权重更新-同步模型 1 次（清空 buffer）
            "learning_rate": 2e-4,      # 学习率
            "clip_param": 0.3,          # 为了训练稳定，限制新旧 policy network 的差距
            "vf_clip_param": 10.,       # 为了训练稳定，限制 value network 的差距
            "vf_loss_coef": 1.,         # value loss 的 scale factor（影响因子），为 1 则不 scale
            "entropy_coef": 0.1,        # entropy loss 的 scale factor（影响因子），为 1 则不 scale
        },
        # 模型存储参数
        "save": {
            "interval": 100,            # 模型存储间隔，即网络更新多少次存储
```

```
一次模型
        },
        # 模型加载参数，不设置则默认不进行预训练模型加载
        # "load": {
        #     "model_file": "models/f4v1x_model/f4v1x_model_200.npz",
# 预训练模型存储路径
        # }
    },
}



from drill.flow.flow_env_ppo import FlowEnvPPO
from drill.flow.flow_model_ppo import FlowModelPPO
from configs.builder_config import builder

algo = {'flow_env': FlowEnvPPO, 'flow_model': FlowModelPPO}

flow_config = {
    'algorithm': algo,      # 选择算法，目前最佳实践标准化流程推荐 PPO
    'builder': builder,      # 选择 builder，配置 builder_config 实现的
builder
    'actor_config': {
        # 根据需求可配置不同 actor 进行采样，不同 actor 可以用于单独训练某些模
型或区分"训练-验证"环境
        'actor_0': {
            # 配置需要训练的模型，多个模型通过列表方式导入
            'training_models': [
                {
                    'model_name': 'f4v1_model',      # 模型名称，注意跟
network_config 中对应
                    'fragment_size': 1024,           # GAE 的切片大小，
n-steps 模式
                    'replay_size': 1,              # 若存在 lstm、gru 等时
序网络，该数值可适度调大，默认 1 即可
```

```
                'sample_batch_size': 128,           # 训练模型的batch_size，通常推荐为128/256/512
                # 'max_data_reuse': 1,          # 在收集到下一个batch的数据前，最多对当前batch额外重复训练多少次，设置为0即禁止重复使用
                # 'putback_replays': False,         # 是否将使用的replay原路放回ReplayBuffer
                # 'sample_mode': "LIFO",          # LIFO：后进先出，RANDOM：优先选择最新的未曾使用过的样本
                # 'replay_buffer_size': 16,         # buffer中的batch_size数量上限，LIFO模式默认为16，RANDOM模式下默认为64。
            },
        ],
        'inference_models': None,               # 配置只跑前向的模型，inference_models列表中的模型仅进行模型推理，不参与训练
        'env_num': 20,                          # 仿真环境数量
        # extra_info中增加actor描述，extra_info支持字典格式对描述信息进行定义，可通过区分extra_info来实现"训练-验证"相独立
        'extra_info': {'index': 'training', 'description': 'used for training'},
    },
    'actor_1': {
        'training_models': None,    # 'training_models'为空，则该actor只跑前向，不训练智能体
        'inference_models': ['f4v1_model'],
        'env_num': 5,
        'extra_info': {'index': 'evaluating'},
    },
},
}



from drill.builder import BPBuilder
from env_interface import F4v1Env
from drill.pipeline.agent_pipeline import AgentPipeline,
```

```
HandlerSpecies

from drill.pipeline import GlobalPipeline

from pipeline import feature_handler, reward_handler, action_handler,
player_done_process

from     configs.network_config     import     entity_feature_set,
target_feature_set

from configs.network_config import model_config



# env 配置，如果需要对环境进行配置，可以通过 params 传递相应参数，并在 env 创建
时使用

env = {"class": F4v1Env, "params": {}}


# 智能体名称

AGENT_NAMES = ['player0', 'player1', 'player2', 'player3']


# pipeline 配置

pipeline = {

   # 根据 pipeline 中实现的 feature_handler， action_handler，
reward_handler 等函数配置智能体训练 pipeline

   "f4v1_pipeline": {

      "class": AgentPipeline,

      "params": {

         "handler_dict": {

            HandlerSpecies.FEATURE:             (feature_handler,
[entity_feature_set, target_feature_set]),

            HandlerSpecies.REWARD: reward_handler,

            HandlerSpecies.ACTION: action_handler,

         },

         "batch_algo": 'gae',

         "batch_config": {

            "gamma": 0.99,

            "lamb": 0.95,

         },
```

```
        },
    },
    "global": {
        "class": GlobalPipeline,
        "params": {
            "pre_process": player_done_process
        }
    }
}

# 智能体与神经网络模型映射关系，支持多智能体
agents = {
    'player0': {
        "model": "f4v1_model",          # 选择智能体对应网络模型
        "pipeline": "f4v1_pipeline"     # 选择对应 pipeline 配置
    },
    'player1': {
        "model": "f4v1_model",
        "pipeline": "f4v1_pipeline"
    },
    'player2': {
        "model": "f4v1_model",
        "pipeline": "f4v1_pipeline"
    },
    'player3': {
        "model": "f4v1_model",
        "pipeline": "f4v1_pipeline"
    },
}

builder = BPBuilder(agents, model_config, env, pipeline)
```

# 3. 环境对接

Env 目录负责定义环境相关信息，根据功能的不同，目录所包含的文件可以大致分为 xxx_env.py 和其他文件（非必须）。其中，xxx_env.py 属于最佳实践框架标准接口的一部分，用于对接仿真环境的输入输出信息；其他文件则独立于训练框架，主要负责诸如定义环境客户端、环境指令等等环境/业务相关信息，供最佳实践标准模块进行调用。

## 3.1 核心类/函数说明

表 3.1 Env 功能说明

| | |
|---|---|
| **类名称** | Env |
| **引用路径** | drill.env |
| **功能说明** | 环境接口，基于 bp 标准化的 env 类使训练框架与仿真环境进行交互。 |
| **初始化参数** | env_id: int, extra_info: dict |

表 3.2 reset 功能说明

| | |
|---|---|
| **函数名称** | reset |
| **所属类** | Env |
| **功能说明** | 通过发送控制命令重启仿真环境，开始下一局对抗推演。 |
| **输入参数** | 无 |
| **返回值** | 环境初始化态势 |

表 3.3 step 功能说明

| **函数名称** | step |
| --- | --- |
| **所属类** | Env |
| **功能说明** | 通过发送行为命令控制仿真环境中的智能体行为，并返回实时态势。 |
| **输入参数** | command_dict: Dict[str, Any] |
| **返回值** | 态势信息，是否终止，环境奖励（如有） |

## 3.2 样例模版

```
from drill.env import Env
from drill import summary
from raw_env.env_def import *
from raw_env.f4v1_game import make_env_f4v1
from typing import Any, DefaultDict, Dict, List, Tuple
from drill.pipeline.interface import ObsData



class F4v1Env(Env):


    def __init__(self, env_id: int, extra_info: dict):
        self.env_id = env_id
        self._env = make_env_f4v1()
        self.extra_info = extra_info
        from configs.builder_config import AGENT_NAMES
        self.agent_names = AGENT_NAMES
```

```
        # self.reset()
        print("Environment is ready!")

    def reset(self) -> Any:
        """重置 Env

        Returns
        -------
        Any
            环境的初始状态
        """

        obs = self._reset_env()
        obs_dict = {agent_name: ObsData(obs=obs[agent_name]) for
agent_name in self.agent_names}
        return obs_dict

    def step(self, command_dict: Dict[str, Any]) -> Tuple[Dict[str,
ObsData], bool]:
        """所有参与者依次执行 command

        Parameters
        ----------
        command_dict : Dict[str, Any]
            包含所有参与者的动作

        Returns
        -------
        Tuple[Dict[str, Any], bool, Dict[str, Any]]
            obs_dict, done, reward_dict
        """

        # 向仿真端发送命令
```

```
        # try:
        actions = []
        from configs.builder_config import AGENT_NAMES
        for agent_name in AGENT_NAMES:
            actions.append([command_dict.get(agent_name, None)])
        self.raw_obs,  self._reward,  self.done,  self.ob_info  =
self._env.step(actions, VIS_STEPS)
        obs_dict = {agent_name: ObsData(self.raw_obs[agent_name],
                                        {"reward":
self._reward[int(agent_name[-1])],},
                                        agent_name)
                    for agent_name in self.agent_names}


        if self.done:
            for agent_name in self.agent_names:
                # 默认以时间作为横轴记录数据
                summary.average(f"{agent_name}_reward_"             +
self.extra_info['index'], self._reward[int(agent_name[-1])])
            summary.average("total_step_" + self.extra_info['index'],
self.ob_info['eplen'])
            summary.average("final_v_"  +  self.extra_info['index'],
sum(self.ob_info['final_v']))
            summary.average("hit_rate_"  +  self.extra_info['index'],
sum(self.ob_info['pldone']))
            summary.average(name="episode_reward(time)_"            +
self.extra_info['index'], value=self.ob_info['eprew'])
            # 如果需要以 learn_step 作为横轴记录数据，则需要传递-指定模型学习
步数-作为横轴参数
            summary.average(name="episode_reward(learn_step)_"     +
self.extra_info['index'], value=self.ob_info['eprew'],
                            x_axis="f4v1_model_learn_step")
            print(f'Env {self.env_id}: The total number of steps in the
current episode: {self._env.step_cum}')


        return obs_dict, self.done
```

```
   def _reset_env(self):
       # 重置为初始状态
       self.raw_obs = self._env.reset()
       self.done = False
       self.error = False
       self._reward = [0, 0, 0, 0]
       self.ob_info = {'eprew': 0, 'eplen': 0, 'pldone': [False, False,
False, False], 'final_v': [0, 0, 0, 0]}
       return self.raw_obs
```

# 4. Pipeline 开发

根据 RL 的使用场景，Drill 实现了 AgentPipeline 和 GlobalPipeline。AgentPipeline 用于处理每个智能体的数据流，可以有多个不同的 AgentPipeline 实例，但是每个智能体只能对应一个 AgentPipeline 实例。GlobalPipeline 可以访问所有 agent 的数据，用于对送入 AgentPipeline 的数据进行前置处理，只能有一个 GlobalPipeline 实例。

AgentPipeline 有三个处理函数，分别对状态/特征、奖励、动作进行处理，用户需自行实现这三个函数的代码逻辑以匹配业务需要。GlobalPipeline 是可选的，并不一定要使用，是否使用根据业务需要决定。它默认有前置处理和后置处理两个函数，前置处理函数用于对输入 AgentPipeline 的数据进行处理，后置处理函数主要对 AgentPipeline 输出的数据进行处理，两个函数都需要用户自行实现代码逻辑以满足业务需要。同时,我们帮助用户实现了 advantage 计算，避免了重复实现导致的代码重复以及潜在出现 bug 的可能。

## 4.1 核心类/函数说明

表 4.1 reward_handler 功能说明

| | |
|---|---|
| **函数名称** | reward_handler |
| **功能说明** | 用户通过该函数根据原始态势信息和历史信息自定义奖励函数；如果环境有奖励返回值，可以考虑使用环境奖励和人为奖励共同对智能体训练进行引导。 |
| **输入参数** | data: ObsData, history: History |
| **返回值** | reward: dict |

表 4.2 feature_handler 功能说明

| 函数名称 | feature_handler |
| --- | --- |
| 功能说明 | 接收环境返回的原始态势信息，将其转化为可传入神经网络的状态信息，并且需要与特征模板一一对应。 |
| 输入参数 | data: ObsData, history: History |
| 返回值 | state: dict |

表 4.3 action_handler 功能说明

| 函数名称 | action_handler |
| --- | --- |
| 功能说明 | 接收神经网络返回的原始动作信息，将其转化为可传入仿真端推演的数据格式，返回动作指令。ActionData 数据结构中含 action、action_mask 等项，通过 action 中含不含动作名实现动作头的 valid，action_mask 用于实现动作掩码。 |
| 输入参数 | data: ActionData, history: History |
| 返回值 | action: ActionData |

表 4.4 自定义 handler 函数功能说明

| 函数名称 | 自定义 |
| --- | --- |
| 功能说明 | 用于 global_pipeline 中控制 agent_pipeline 的数据流。为非必须实现的函数，仅当 global_pipeline 启用时。 |
| 输入参数 | data: ObsData, history: History |
| 返回值 | agent_dict: dict[str, ObsData], history: History |

## 4.2 样例模版

```
from drill.pipeline.interface import ObsData, ActionData, History



def reward_handler(data: ObsData, history: History):
    """
    实现奖励函数，并根据态势数据计算当前奖励

    :param data: 原始态势信息
    :param history: 历史信息
    :return: 智能体当前获得奖励
    """
    # 在 f4v1 样例中，环境返回值包括了原始奖励，则在奖励计算时，可以直接使用环境奖励
    extra_info_dict = data.extra_info_dict
    if (extra_info_dict is None) or ('reward' not in extra_info_dict) or (not extra_info_dict):
        return {'reward': 0}
    else:
        return {'reward': extra_info_dict['reward']}



def feature_handler(data: ObsData, history: History):
    """
    o2s，实现态势信息到神经网络输入信息的数据转换

    :param data: 原始态势信息
    :param history: 历史信息
    :return: 神经网络输入数据
    """
```

```
    my_units_list, ally_feature = [], []
    for k, v in data.obs.items():
        if k == 'b_info':
            b_info = {
                'b_pos': v['b_pos'],
                'b_visible': v['b_visible']
            }
            continue
        if k[-1] == data.agent_name[-1]:
            my_units_list.append(v)
        else:
            if any([item.sum() for item in v.values()]):
                ally_feature.append(v)
    my_units_list.extend(ally_feature)
    name2feature = {
        'my_units': my_units_list,
        'b_info': b_info
    }
    # history.agents_history[data.agent_name] = name2feature # 如果想将 obs 数据传到 action_handler 中，可通过该方式实现
    return name2feature



def action_handler(data: ActionData, history: History) -> ActionData:
    """
    a2c，实现网络输出到智能体动作的转换
    注：data.action 会用于训练，目前版本不建议 a2c 在该模块实现，后续版本会对该部分进行修正

    :param data: 原始神经网络输出
    :param history: 历史信息
    :return: 解析后的动作
    """
```

```
    # f4v1 样例中，环境可以直接接受神经网络输出值，则不需要做动作映射处理
    # data.action.pop('action_x')    # 如果想令某个动作头不参与训练，即 valid_action 为 false，则可以通过该方式实现
    return data



def player_done_process(data: ObsData, history: History):
    """
    在多智能体环境中，经常会出现部分训练单位在 episode 结束前死亡的情况。
    不同的仿真环境对死亡单位的处理方式不同，有的可能直接传递一个空的状态，有的可能传递一个全 0 的状态，
    有的可能保留其死亡前的状态，有的可能直接剔除死亡单位。
    为了保障数据流的正确性和智能体模型训练效果，在训练单位死亡后应当不再训练其智能体。
    因此我们需要对死亡单位的数据进行屏蔽处理。

    该方法用于实现屏蔽死亡训练单位数据的功能

    :param data: 原始态势数据
    （注：data 为 dict 类型，包含'agent_name'、'obs'、'extra_info_dict'三个字段。
    其中'extra_info_dict'包括'episode_done'字段以及其他从 Env.step 中 obs_dict 传过来的变量，'episode_done'表示本局对抗是否结束。）
    :param history: 历史信息
    :return: 经过预处理后的态势信息，传给 feature_handler
    """
    pre_agent_dict = {}
    for agent_name, info in data.items():
        # 对齐环境中单位的名字
        agent_name_fix = agent_name[:-1] + '_' + agent_name[-1]

        # 本环境中，死亡单位会传递全 0 的状态，我们可以通过判断状态是否全 0 来判断单位是否死亡
        # player_done 为 True 表示单位死亡，为 False 表示单位存活
```

```
        player_done = not any([item.sum() for item in
info.obs[agent_name_fix].values()])


        # 一个 episode 结束时，会立即获取初始状态，无需屏蔽数据，否则会导致
KeyError
        # episode 没结束时，屏蔽死亡训练单位的数据。
        if info.extra_info_dict.get("episode_done", False) or not
player_done:
            pre_agent_dict[agent_name] = info


    return pre_agent_dict, history
```

# 5. 使用建议

## 5.1 智能体开发建议

1. 如何实现态势到状态数据的转换

采用字典表示态势数据。该字典的每个 key 与 config 中数据特征模板的 key 一一对应。通过构造相对应的数据结构，将仿真从获取原始数据填入对应的数据结构中。

2. 如何实现动作到指令数据的转换

采用字典表示决策数据，每个 key 与 config 中动作类型模板的 key 一一对应。通过构造相对应的数据结构，将神经网络输出的 action 相应地转换成为仿真环境能够理解的指令输出。

## 5.2 参数设置建议

1. Learner 参数相关建议

replay_size * batch_size 的值，直接决定了一次训练用多少个 step 的数据，太大了内存无法承受；太小了一次训练数据量不够，训练效果受影响。

# 6. 补充资料

1. Python 基础参考资料

| 资料名称 | 资料链接 |
| --- | --- |
| 菜鸟课程 | https://www.runoob.com/python/python-tutorial.html |
| 官方文档 | https://docs.python.org/zh-cn/3.9/ |

2. 深度学习推荐参考资料

| 资料名称 | 资料链接 |
| --- | --- |
| 《动手学深度学习》 | https://zh.d2l.ai/ |
| 李宏毅机器学习课程 | https://speech.ee.ntu.edu.tw/~hylee/ml/2021-spring.html |

3. 强化学习推荐参考资料

| 资料名称 | 资料链接 |
| --- | --- |
| 李宏毅深度强化学习 | https://www.bilibili.com/video/av24724071 |
| 深度强化学习笔记 | https://blog.csdn.net/cindy_1102/article/details/87904928 |
| EasyRL | https://datawhalechina.github.io/easy-rl/#/ |
| 动手学强化学习 | https://hrl.boyuai.com/ |

4. 强化学习进阶参考资料

| 资料名称 | 资料链接 |
| --- | --- |
| DeepMind - RL | https://www.bilibili.com/video/BV1xp4y1q7kW |
| AlphaStar: Mastering the Real-Time Strategy Game StarCraft II | https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii |