**АЛГОРИТМЫ POLICY GRADIENT**

Подготовка разбора методов обучения с подкреплением семейства градиента политики (Policy Gradient), таких как ванильный PG, PPO, TRPO и других (количество алгоритмов выбирает докладчик, но чем обширнее исследование, тем лучше) и реализация их на практике с последующим сравнением их эффективности (в качестве полигона для испытаний предлагается использовать игры Atari или движок OpenAI от Gym)

Обучение с подкреплением – это тип алгоритма машинного обучения, который позволяет агенту учиться у окружающей среды методом проб и ошибок и использовать обратную связь от своих действий, чтобы узнать лучшую стратегию для достижения своих целей. Он работает путем назначения вознаграждений за успешные действия и наказаний за неуспешные, что позволяет агенту соответствующим образом корректировать свою стратегию.

**Градиентный алгоритм политики PG   
и лежащие в его основе принципы:**

Алгоритм Policy Gradient (PG) - это тип алгоритма обучения с подкреплением, используемый для повышения эффективности работы агента в конкретной задаче. Он работает на основе политики, определяющей, как агент должен действовать в той или иной ситуации. Затем агент получает обратную связь о своих действиях и использует ее для корректировки своей политики, чтобы в будущих ситуациях действовать лучше.

В основе алгоритма PG лежит идея градиента политики, который представляет собой скорость, с которой политика корректируется в ответ на наблюдаемое вознаграждение. Градиент политики рассчитывается путем взятия производной от ожидаемой прибыли в зависимости от параметров политики. Этот градиент затем используется для обновления параметров политики таким образом, чтобы агент мог максимизировать свою ожидаемую прибыль.

Затем агент использует эту обновленную политику для выполнения действий и наблюдения за вознаграждением, и процесс повторяется до тех пор, пока агент не выучит оптимальную политику. Этот процесс обучения называется итерацией политики. Ключевая идея PG заключается в том, что агента побуждают больше исследовать и рисковать, чтобы максимизировать ожидаемую прибыль, поскольку он может учиться на получаемых вознаграждениях.

Алгоритмы PG широко используются в робототехнике, видеоиграх и других приложениях ИИ. Они также используются в таких областях исследований, как обработка естественного языка и обучение с подкреплением.

**ПРОГРАММНАЯ РЕАЛИЗАЦИЯ PPO**

**Модуль «PPOnetwork.py»**

Этот модуль определяет класс с названием Policy, который является нейронной сетью для агента обучения с подкреплением.

Ниже представлен разбор того, что делает код:

* Класс Policy является подклассом nn.Module, что означает, что он использует фреймворк PyTorch для нейронных сетей.
* Конструктор (метод \_\_init\_\_) класса Policy принимает два параметра: action\_size и input\_size. action\_size представляет собой количество возможных действий, которые может предпринять агент, а input\_size представляет собой размер входного состояния.
* В конструкторе инициализируются различные слои нейронной сети с помощью nn.Linear и присваиваются инстанс-переменным (self.fc1, self.fc2, self.fc3\_v, self.fc3\_pi). Эти слои полностью связаны (линейные) слои с различными входными и выходными размерами.
* Конструктор также инициализирует функции активации (self.tanh, self.relu, self.softmax) из модуля nn PyTorch. self.tanh представляет собой гиперболический тангенс, self.relu представляет собой функцию прямоугольной линейной единицы, а self.softmax представляет собой функцию softmax.
* Метод pi определяет прямой проход для сети политики. Он принимает входной сигнал x и передает его через полностью связанные слои с функциями активации ReLU (self.relu). Выходной сигнал затем передается через последний полностью связанный слой (self.fc3\_pi) и затем через функцию softmax (self.softmax) для получения распределения вероятностей по действиям.
* Метод v определяет прямой проход для сети значений. Он принимает входной сигнал x и передает его через те же полностью связанные слои, что и метод pi. Выходной сигнал передается через последний полностью связанный слой (self.fc3\_v), чтобы получить одно значение, представляющее приблизительное значение входного состояния.

В целом, данный код определяет модель нейронной сети с тремя полностью связанными слоями для сети политики и сети значений. Сеть политики выводит распределение вероятностей по действиям, а сеть значений выводит одно значение, представляющее значение состояния. Такая модель часто используется в алгоритмах обучения подкреплением, таких как методы градиента политики.

**Модуль «PPOconfig.py»**

Этот код описывает класс PPOconfig, в котором определены различные параметры и настройки для алгоритма PPO (Proximal Policy Optimization).

Параметры и их значения:

* gamma = 0.99: коэффициент дисконтирования для вычисления суммарной награды.
* plot\_every = 10: частота обновления графика, на котором отображается процесс обучения.
* update\_freq = 1: частота обновления модели.
* k\_epoch = 3: количество эпох обучения на каждом обновлении модели.
* learning\_rate = 0.02: скорость обучения (шаг градиентного спуска).
* lmbda = 0.95: коэффициент, используемый для смешивания оценок текущей политики и предыдущей политики.
* eps\_clip = 0.2: значение для отсечения относительной разницы между новой и старой вероятностями действий.
* v\_coef = 1: коэффициент для вычисления потери значения (value loss) в общей функции потерь.
* entropy\_coef = 0.01: коэффициент для вычисления потери энтропии (entropy loss) в общей функции потерь.

Дополнительные настройки:

* memory\_size = 400: размер памяти, используемой для хранения опыта.
* train\_cartpole = True: флаг, указывающий, следует ли обучать модель на задаче CartPole.

Таким образом, этот класс предоставляет значения параметров и настроек для использования в алгоритме PPO, а также определяет некоторые дополнительные настройки для обучения модели на задаче CartPole.