基于强化学习的轨迹简化方法

杨振宇

摘要

轨迹数据被用于各种下游应用中，包括智能导航、兴趣点推荐和未来流量预测等等。轨迹数据通常由传感器持续采集，并在存储在硬盘里或者云端里，从而产生大量的轨迹点数据。针对数据量太大的问题一个常见的做法是进行轨迹简化，即在收集轨迹时放弃一些点。现有的算法通常涉及一些决策任务，基于此仍需要使用一些人为制定的规则。在本文中，我们通过强化学习来学习决策任务的策略，并根据学到的策略来进行轨迹简化的指导。与现有的算法相比，基于强化学习的方法是数据驱动的，可以适应不同问题的动态变化。在进行了广泛的实验后，基于强化学习方法的简化轨迹方法与现有的方法相比具有较小的误差，且能够达到不错的运行速度。

关键词：轨迹数据；轨迹简化；强化学习

1 引言

轨迹数据是一种捕捉移动性物体（如车辆、行人、机器人等）行动轨迹的数据类型。它是许多下游应用的核心驱动数据，如智能导航、兴趣点推荐、未来流量预测城市和流动性分析等。轨迹数据通常是连续产生的，由 GPS 传感器等设备远程采集。一个传统的采集策略是，传感器定期读取经纬度坐标和时间戳信息，这表示为一个有时间标记的位置点（或称为时空点），并将这些时空点存储在一个缓冲区。通常情况下，一个传感器的内存空间有一个限制存储预算，低效的计算能力和有限的网络带宽。随之而来的问题是，缓冲区会经常被占用导致多余的数据不能及时存储而导致丢失，或者导致传输空间点的工作量很大。此外，在某些应用中，可能有成百上千的传感器，它们同时收集轨迹数据。一旦所有这些传感器收集的轨迹数据被上传到服务器上，其存储体积将是无比巨大的。随之而来的另一个问题是，巨大的轨迹数据量会增加存储成本，更重要的是使数据的查询处理变得困难。

处理上述两个问题（存储成本和数据查询）的常见做法是进行轨迹简化，轨迹简化的本质是放弃给定原始轨迹中的一些点，保留剩余的点作为简化轨迹。具体来说，基于在线模式下，轨迹数据是逐点输入的，一旦某个点被放弃，就不能再访问。在批处理模式下，轨迹数据被一次性输入到算法中，并在整个轨迹简化过程中保持可访问。轨迹能够被简化的动机有两个方面。首先，不是所有的轨迹点都携带同等数量的信息，有些点携带很少甚至没有信息。例如，当一个物体以恒定的速度沿直线运动时，除了第一个和最后一个点之外，所有的点都不携带任何信息，可以被放弃。其次，如果放弃一些空间点，剩余的数据使得传输、存储和查询处理的负担就会大大降低。在本文中，我们考虑了一个轨迹简化的问题，即放弃一定数量的时空点（或者最多保留一定数量的时空点），使信息损失（被称为简化轨迹的”误差”）达到最小。我们称这个问题为最小误差问题。

综上所述，本次复现的这篇论文的贡献即是提出了一种基于强化学习的方法，称为 RLTS，用于轨迹简化，这是在轨迹简化问题上的第一个使用到强化学习方法的。其次，RLTS 是数据驱动的方法，有能力适面对不同情况下的不同动态变化。此外，RLTS 具有很好的通用性，在面对多种误差测量方法上都能实现很好的简化效果，而许多现有算法只适用于某些特定的误差测量。最后，原文还提出了

RLTS 的一个变体，即 RLTS-Skip，它比 RLTS 运行得更快，并在有效性和效率之间提供了一个可控的权衡。特别是对于批处理模式，鉴于数据访问量的增加，原论文作者研究了基于 RLTS 方法的三种变体，由于时间和任务量原因，复现只针对在线模式下，RLTS 和 RLTS-Skip 的方法进行了复现研究。在现实生活中的数据集上进行的大量实验表明，与现有的算法相比，原论文的方法具有更好的效率，复现结果基本达到了原论文原有水平。

2 相关工作

本节介绍轨迹简化问题的两个场景以及所使用的主要技术。

2.1 轨迹简化的实时模式

在线模式下，轨迹数据由传感器持续收集并存储在本地缓冲区。轨迹简化问题是决定哪些点要放弃，相应地哪些点要保留在缓冲区并在未来发送到服务器端。在现有的在线模式下的轨迹简化问题研究中，可以转化为最小化简化后轨迹误差问题，比如 Potamias 等人[1]提出了 STTrace 算法，该算法基于人工规定的启发式方法逐一处理传入的时空点，对于每一个点决定是直接丢弃还是插入到缓冲区中。Muckell 等人[2]提出了 SQUISH 算法遵循 STTrace 的框架，本质上也是利用启发式方法简化轨迹点，但对时空点的启发式规则采用不同的定义。其他关于在线模式轨迹简化的建议并不针对最小简化轨迹误差问题，Lin 等人[3]研究了最小误差问题的对偶问题。Trajcevski 等人[4]通过假设预测运动物体的未来位置，如果一个收集点明显偏离预测的位置，则放弃该点，否则保留。Katsikouli 等人[5]提出可以根据一些拓扑学上的持续特征进行轨迹简化，这些特征表明了不同时空点的重要性程度。

2.2 轨迹简化的批处理模式

在批处理模式下，要简化的轨迹中的所有时空点都是作为一个整体输入到模型中，并在整个简化过程中都保持可访问性。在现有的关于批处理模式下的轨迹简化的研究中，同样也是针对最小化简化轨迹误差问题。具体来说，如 Bellman 等人[6]提出了一种叫做 Bellman 的动态编程算法，但 Bellman 至少要在立方时间复杂度中运行，这对海量数据集来说是非常耗时的。Long 等人[7]提出了一种称为 Span-Search 的近似算法，它是专门为测量方向感知距离误差而设计的，所以缺少了通用性。其他批处理模式的轨迹简化方法并不以最小误差问题为目标，现回顾如下，如 M.Chen 等人[8]研究了最小误差的对偶问题。J.Chen 等人[9]提出了一个从轨迹中选择最具代表性的点的子集的解决方案。zhao 等人[10]构建了一个参考轨迹集以支持轨迹简化。

2.3 强化学习

强化学习是一种机器学习方法，旨在让智能体通过尝试不同的动作并获得奖励或惩罚来学习如何完成一个任务。在强化学习中，智能体会不断尝试新的动作并学习如何最大化期望奖励。它与监督学习和无监督学习不同，因为它不是通过给出正确的答案来训练模型，而是通过不断迭代来让智能体自己发现最优解。近年来，强化学习模型已被成功用于解决各种算法问题。例如，Kong 等人[11]探讨了三个经典组合优化问题的强化学习方法。Wang 等人[12]提出了一种有效的基于强化学习的动态双比特匹配算法。Marcus 等人[13]将强化学习用于连接顺序枚举。在原论文方法中，作者将轨迹简化问题建模为马尔科夫决策过程，并使用流行的策略梯度方法来解决该问题。据知悉，这是第一个基于深度强化学习的轨迹简化解决方案。

3 本文方法

3.1 本文方法概述

本次复现仅考虑在实时模式下的情况，轨迹当中的时空点是以在线方式逐一输入的，而只有一个大小为 W 的缓冲区可用，即在整个轨迹简化过程中最多可以保留 W 个点。针对于前 W 个点，原作使用了一种现有的策略使得有 W 个时空点将初步存储在缓冲区中，对于剩余的每个时空点，由于缓冲区已经满了，所以每次缓冲区中放弃一个点以释放一些空间，然后将新的点存储在缓冲区中。与那些现有的策略不同的是，这些策略使用一些人为规定的启发式数值来决定在缓冲区已满时放弃哪一个点，本作目的是为这个决策任务实现一个更智能的方法。具体来说，该方法将轨迹简化问题视为一个连续的决策过程，并将其建模为马尔可夫决策过程，使用策略梯度方法学习马尔可夫决策的最优策略，然后提出一种名为为 RLTS 的算法，该算法使用学到的策略处理最小误差问题。在 RLTS 的基础上，原作还提出了 RLTS 的一个变种，称为 RLTS-Skip，它通过跳过一些被扫描的点来提高 RLTS 的效率。轨迹简化概念效果如图 1所示：

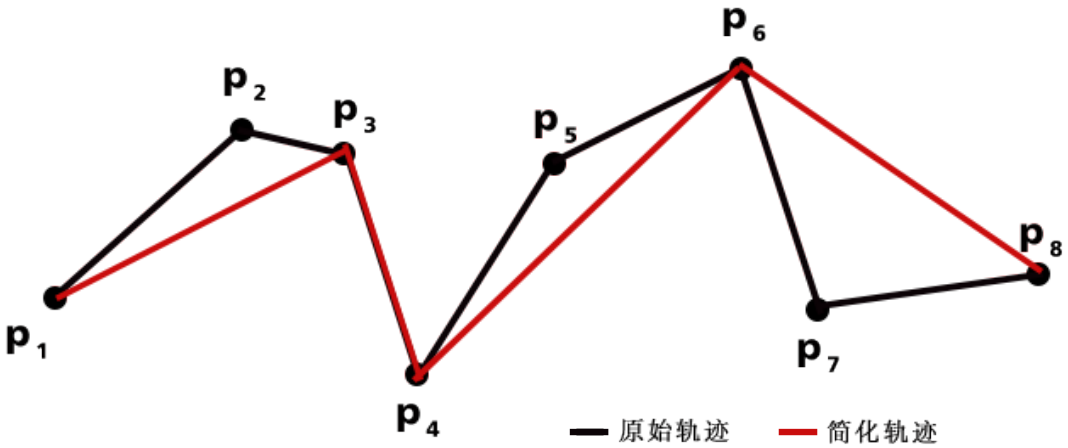

图 1: 轨迹简化效果图

3.2 先验知识

首先了解简化轨迹中常用的四种简化误差计算方式，轨迹简化误差与后面计算总误差以及状态和奖励部分关联密切。以下共介绍四种误差 (1) synchronized Euclidean distance(SED)，(2) perpendicular Euclidean distance(PED)，(3) direction-aware distance(DAD)，(4) speed-aware distance (SAD)

- SED，同步欧几里得距离，以点到锚线段的最短线段为误差距离；
- PED，同步欧几里得距离，以点到锚线段的垂直距离为误差距离；
- DAD，方向感知距离，以点到锚线段偏差方向的角度为误差标准；
- SAD，速度感知距离，计算出相同时间平均速度在锚线段走过的距离的位置点到简化点的距离为误差标准。

3.3 MDP 过程及最小化误差

我们将最小误差问题建模为 MDP，它由四个组成部分组成，即状态，动作，状态转移和奖励，如下所述。

状态 状态是智能体当前的环境状态，它是智能体决策的基础。智能体可以通过观察环境来感知到当前的状态，并根据当前的状态来决定下一步的动作。考虑目前的环境，即在缓存区的 W 个点 $p_{s_1}, p_{s_2}, ..., p_{s_W}$ 以及新插入的点 p_i。RLTS 算法模型的任务目标就是抉择是否从当前的缓存区当中丢弃一个点并把插入点纳入当前缓存区，还是把当前待插入点丢弃。值得注意的是，根据轨迹简化的定义，我们不允许丢弃时空点 p_{s_1}，即 p_1，也就是初始点。一个基于直觉的想法是删除其中一个点，使得由于删除操作而引入更小的的误差。现假设如果删除点 p_{s_j} (2 ≤ j ≤ W)，现有的两个锚线段 $\overline{p_{s_{j-1}}p_{s_j}}$ 和 $\overline{p_{s_j}p_{s_{j+1}}}$ 将被破坏，一个新的锚线段 $\overline{p_{s_j}p_{s_{j+1}}}$ 将被创建，而其他线段不变。由于简化轨迹的误差由其锚线段的误差决定，并且锚线段的误差进一步由其误差原始轨迹中以该段作为其锚段的空间点决定，因此新产生的误差是有关于时空点 p_{s_j} 和 $\overline{p_{s_{j-1}}p_{s_{j+1}}}$

基于以上分析，对于每一个在缓存区的点定义一个分量状态函数，定义为 $v(p_{s_j})$, 具体可见公式(1)：

$$\mathrm{v}(\mathrm{p}_{\mathrm{s}_j}) := \varepsilon(\overline{\mathrm{p}_{\mathrm{s}_{j-1}}\mathrm{p}_{\mathrm{s}_{j+1}}} \,|\, \mathrm{p}_{\mathrm{s}_j}) \tag{1}$$

, 如果一个时空点的价值较低意味着一旦该点被丢弃，引入的误差往往较小，因此应该给予更高的概率丢弃它。话句话说就是要找一个更高的误差，一个拥有更大的价值的时空点，然后使得这个最大误差是所有简化轨迹的误差最小，即回归到最迟的定义，寻找最小误差问题。然后，根据缓冲区中点的值定义情况的状态，用 s 表示。一个直觉的想法是将所有（W - 1）个点 p_{s_j}（2 ≤ j ≤ W）的值合并到缓冲区中以定义状态。但是，这个定义有两个问题。首先，由于 W 是问题的输入，并且对于不同的问题实例，W 通常是不同的。使用此定义，为一个输入 W 定义的模型将不适用于与 W 不同的其他输入。其次，W 通常是一个中等到大的整数，例如，它可以以千为单位。基于此，状态空间将会非常巨大，模型将难以训练。

原作建议将状态 s 定义为包含 k 个最低值的集合，其中 k 是可以调整的超参数，而不是包含所有在缓存区的值的集合。具体来说，对在缓存区的所有价值做一个升序排列，使 $\mathrm{v}(\pi_{\mathrm{p}(1)}), \mathrm{v}(\pi_{\mathrm{p}(2)}), ..., \mathrm{v}(\pi_{\mathrm{p}(\mathrm{W})})$ 是一个以升序排列的数值列表。然后，定义状态 s 如公式(2)。

$$\mathrm{s} := \{\mathrm{v}(\mathrm{p}_{\pi(1)}), \mathrm{v}(\mathrm{p}_{\pi(2)}), ..., \mathrm{v}(\mathrm{p}_{\pi(\mathrm{k})})\} \tag{2}$$

基于此定义，状态具有独立于问题输入的固定大小。此外，状态空间大小可通过超参数 k 进行控制。

动作抉择 动作是强化学习中系统可以采取的行为的集合。通常情况下，强化学习系统会在每一步都根据当前的状态和可用的动作来决策采取哪个动作。这些决策通常是基于系统所学到的策略，该策略告诉系统在某些状态下哪些动作是最优的。系统会不断学习并改进这些策略，以获得尽可能多的奖励。

一个缓存区大小为 W，此时考虑要插入一个新的数据点。此时一个动作决策就是从 W+1 个点中决定哪个点需要被去掉。在强化学习中，动作会根据当前的状态决定下一个动作，而此时的状态便是 $s = \{v(\mathrm{p}_{\pi(1)}), \mathrm{v}(\mathrm{p}_{\pi(2)}), ..., \mathrm{v}(\mathrm{p}_{\pi(\mathrm{k})})\}$ 综上，一个动作就是从 $p_{s_1}, p_{s_2}, ..., p_{s_W}, p_{s_{W+1}}$ 中选择最优的点去掉。一个比较直觉的想法是定义 W 个动作，即针对缓存区的每一个点都有去掉的可能性，因为初始点包含的信息量最大所以不考虑在去掉的可能性。这样的决策有两个问题，第一个问题是算法的灵活性，对于一个轨迹点来说少则包括数千个点，对于记录频率频繁的传感器来说甚至更多，这样的动作抉择会使得问题规模变得异常大；第二个问题是动作决策空间会非常的大，即前一问题的延伸，这么大的

动作空间会使得模型很难训练。因此事实上，将我们的注意力限制在那些值较小的点上是很直观的，因为放弃其中一个点会导致一个小的后续错误。因此，我们专注于那些在状态中保持其值的点，即 $v(\mathrm{p}_{\pi(1)}), \mathrm{v}(\mathrm{p}_{\pi(2)}),..., \mathrm{v}(\mathrm{p}_{\pi(\mathrm{k})})$

综合考虑以上两个问题，我们定义了一个包含 k 个动作的动作空间，每个动作的意思是去掉一个点 $\mathrm{v}(\pi_{\mathrm{p(j)}})$（$1 \leq \mathrm{j} \leq \mathrm{k}$）。形式上，我们定义一个动作，我们用 a 表示，如公式(3)所示：

$$\mathrm{a} := \mathrm{j} \quad (1 \leqslant \mathrm{j} \leqslant \mathrm{k}) \tag{3}$$

其中动作 a = j 表示它丢弃点 $p_{\pi(j)}$。

奖励函数 假设模型在状态 s 处执行操作 a，然后到达新的状态 s'。此时定义从状态 s 到状态 s' 的转换相关的奖励，定义为 r 表示，如余下所述。在状态 s 处，考虑在缓冲区中有 W 点，并插入一个新的点 p_i。此时考虑缓冲区中的 W 个点，它们共同构成了一个轨迹，对应于到目前为止已经简化的轨迹，可用 T[1 : i-1] 表示。可用 T' 表示这种简化的轨迹。类似地，在状态 s' 处，产生另一个新的简化的 T[1 ：i] 轨迹，可用 T'' 表示。然后，公式(4)按如下方式定义奖励 r。

$$\mathrm{r} = \varepsilon(\mathrm{T}') - \varepsilon(\mathrm{T}'') \tag{4}$$

其中 T’ 表示一条原始轨迹的简化轨迹，而 T” 是基于 T’ 更新后的新的简化轨迹。一种直觉的做法是，如果动作导致的简化轨迹误差（即 $\varepsilon(\mathrm{T}')$）较小，则奖励更大。有了这个定义，它将有利于那些导致简化轨迹和较小误差的操作。事实上，通过这个奖励定义，可以验证马尔可夫决策问题的目标与轨迹简化问题的目标非常一致。为了看到这一点，假设我们经历了一系列状态 $s_1, s_2, ..., s_N$，并相应地，模型给出一系列奖励 $r_1, r_2, ..., r_{N-1}$. 如果未来的奖励没有损失，则可以计算总奖励，可用公式(5)表示：

$$\sum_{t=1}^{N-1} r_t = \sum_{t=1}^{N-1} (\varepsilon(T'_t) - \varepsilon(T''_t)) = (\varepsilon(T'_1) - \varepsilon(T''_{N-1})) = -\varepsilon(T''_{N-1}) \tag{5}$$

其中 T’ 是在 s_t 的状态下，且动作 a_t 未执行前的简化轨迹,T” 是在 s_t 的状态下，且动作 a_t 执行后的简化轨迹。需要注意的是，$\varepsilon(T'_1) = 0$，因为在起始状态下，没有点被丢弃，因此误差等于 0，并且 $\varepsilon(T''_{N-1})$ 对应于 T 简化轨迹的总误差。

状态转移 在强化学习中，状态转移指的是从一个状态（s）转移到另一个状态的过程。状态表示与决策有关的信息，包括环境和智能体的状态。状态转移过程中，智能体会根据当前的状态和所采取的动作（a）来决定下一步的状态。状态转移在强化学习中具有重要意义，因为它决定了智能体能够在环境中采取的动作。通过不断地观察状态转移，智能体可以学习到最优策略，从而达到最大化预期回报（r）的目标。

在强化学习的轨迹简化问题中，面对环境状态 s，假设动作 a 采取把原先缓存器中的 p_{s_j} 点删除，重新插入一个点 p_i。当动作采取后，环境状态发生变化，此时需要重新计算一下状态，即 s'。状态 s' 将是状态 s 执行操作 a 时的下一个状态。此时将状态 s 更新为状态 s'，如上节所述，状态 s 主要取决于关于缓冲区中点的值，为了计算状态 s'，此时重新计算缓冲区中的点及其相应值在删除 p_{s_j} 后的变化，及输入新的点 p_i 后如何变化。

当在缓存区中去掉时空点 p_{s_j}，并且时空点 p_i 插入后，各点的价值的计算相关的对象会发生变化，

因此状态转移计算会分为两步。第一步具体来讲，当点 p_{s_j} 去掉后，两个直接相邻的位置点是 $p_{s_{j-1}}$ 和 $p_{s_{j+1}}$，所以原先 $p_{s_{j-1}}$ 和 $p_{s_{j+1}}$ 两个点的价值的计算对象发生变化，具体可用公式(6)表述：

$$\begin{aligned} \mathrm{v}(\mathrm{p}_{\mathrm{s}_{\mathrm{j}-1}}) &= \max\{\varepsilon(\overline{\mathrm{p}_{\mathrm{s}_{\mathrm{j}-2}}\mathrm{p}_{\mathrm{s}_{\mathrm{j}+1}}} \,|\mathrm{p}_{\mathrm{s}_{\mathrm{j}-1}}), \varepsilon(\overline{\mathrm{p}_{\mathrm{s}_{\mathrm{j}-2}}\mathrm{p}_{\mathrm{s}_{\mathrm{j}+1}}} \,|\mathrm{p}_{\mathrm{s}_{\mathrm{j}}})\} \\ \mathrm{v}(\mathrm{p}_{\mathrm{s}_{\mathrm{j}+1}}) &= \max\{\varepsilon(\overline{\mathrm{p}_{\mathrm{s}_{\mathrm{j}-1}}\mathrm{p}_{\mathrm{s}_{\mathrm{j}+2}}} \,|\mathrm{p}_{\mathrm{s}_{\mathrm{j}+1}}), \varepsilon(\overline{\mathrm{p}_{\mathrm{s}_{\mathrm{j}-1}}\mathrm{p}_{\mathrm{s}_{\mathrm{j}+2}}} \,|\mathrm{p}_{\mathrm{s}_{\mathrm{j}}})\} \end{aligned} \tag{6}$$

以上公式更新了当轨迹中某一点删除后邻接节点的价值，而介于时空点 $p_{s_{j-1}}$ 和 $p_{s_{j+1}}$ 之前和之后的点因为未受到影响所以不用重新计算。

第二步，是考虑插入点 p_i 的变化。将点 p_i 插入缓冲区后，将有 W+1 个点，依然可以用 $p_{s_1}, p_{s_2}, p_{s_3}, ..., p_{s_{W+1}}$ 表示简化轨迹。其中 $p_{s_{W+1}}$ 即代表新插入到缓存区的点。而我们要计算的点便是 p_{s_W}，再插入新时空点之前，缓存区的最后一个点没有锚线段而不用计算误差与价值，当新的位置点插入后需要进行计算价值纳入排序。具体可以表示为公式(7)：

$$\mathrm{v}(\mathrm{p}_{\mathrm{s}_{\mathrm{W}}}) = \varepsilon(\overline{\mathrm{p}_{\mathrm{s}_{\mathrm{W}-1}}\mathrm{p}_{\mathrm{s}_{\mathrm{W}+1}}} \,|\mathrm{p}_{\mathrm{s}_{\mathrm{W}}}) \tag{7}$$

所有其他点的值保持不变。根据这些值，可以计算状态 s' 的方式与计算状态 s 的方式相同

3.4 策略学习

在强化学习中，策略（policy）是指智能体决定如何在环境中行动的规则。策略选择指的是选择不同的策略来完成任务的过程。策略选择在强化学习中是非常重要的，它决定了智能体在环境中的行为、学习效率、适应能力和性能，也决定了智能体的学习方式。因此，选择合适的策略对于智能体在强化学习中取得成功是至关重要的。

在本文中，马尔可夫决策的核心问题是为代理找到一个最优策略，该策略对应于一个函数，该函数指定代理在特定状态下应选择的操作，以实现累积奖励的最大化。本文通过广泛使用的策略梯度（PNet）方法学习马尔可夫决策的策略。PNet 将随机策略建模为 $\pi_\theta(\mathrm{a}\,|\mathrm{s})$，这意味着选择操作 a 的概率取决于当前所处的状态。而这个 Pnet 的本质上也是一个线性层，可以表述为：

$$\pi_\theta(\mathrm{a}\,|\mathrm{s}) = \sigma(\mathrm{Ws} + \mathrm{b}) \tag{8}$$

其中 σ 代表 softmax 函数，θ 代表此时神经网络的参数，然后，PNet 通过公式(9)计算参数梯度进行更新，可以表述为如下：

$$\nabla_\theta J(\theta) = \sum_{t=1}^{N} \frac{R_t - \overline{R}}{\sigma_R} \nabla_\theta \ln \pi_\theta(a_t\,|s_t) \tag{9}$$

其中 $s_1, s_2, ..., s_N$ 是一系列状态，同理 $a_1; a_2, ..., a_{N-1}$ 是根据 $\pi_\theta(\mathrm{a}\,|\mathrm{s})$ 而选择的一系列动作序列，R_t 表示自采取行动以来的累积奖励，$\overline{R}$ 是 R_t 的平均值，σ_R 是 R_t 的标准差。PNet 根据公式(9)中计算的梯度通过公式反复更新参数，直到满足用户指定的阈值而停止。值得注意的是，PNet 基于平均值和标准差的归一化机制有助于减少公式(9)中计算梯度的方差。

3.5 算法流程

本作提出的 RLTS 算法基于为最小误差问题建模的马尔可夫决策问题的学习策略，如算法 1 中所示。具体来说，它将前 W 个点直接存储在缓冲区中。它为要遍历的一系列状态和动作初始化索引 t，然后，对于以下每个点，以 p_i 举例来说，它按如下方式进行：即首先计算缓冲区中除第一个点之外每一个时空点的价值，即 $v(p_{s_j})$ $(2 \le \mathrm{j} \le \mathrm{W})$，并将这些值保存在最小优先级队列中，其中把价值按照升序排列。然后，它构建一个状态 s_t，其中包含一组 k 个最低值的点，并根据已学习的随机策略对动作

进行采样。基于在 $a_t = j$ 处的采样动作，它丢弃具有第 j 个最低值的点，即 $p_{\pi(j)}$。然后插入正在处理的点 p_i。最后，它返回包含缓冲区中所有 W 点的简化轨迹 T'。

Procedure 1 The RLTS algorithm

Input: 原始轨迹 T =<p_1,p_2,...,p_n>，缓存区大小 W
Output: 简化轨迹 T'
for *i* ***in*** *1,2,3,...,W* **do**
| 存储 p_i 到缓存区中
end
$t \leftarrow 1$ **for** *i* ***in*** $W+1, W+2, ..., n$ **do**
| 计算缓存区中每个点的价值 把价值升序排序 计算 s_t 根据 π_θ(a |s) 采样出动作 a
| 执行动作 $t \leftarrow t+1$
end
缓存区中点即简化轨迹
Return 简化轨迹 T'

4 复现细节

4.1 数据集及实验环境搭建

数据集 Geolife 轨迹数据集由 182 个用户在五年多的时间里（从 2007 年 4 月到 2012 年 8 月）收集的。该数据集的 GPS 轨迹由一系列带有时间戳的点表示，每个点都包含纬度、经度和高度信息。该数据集包含 17,621 条轨迹，总距离为 1,292,951 公里，总持续时间为 50,176 小时。这些轨迹由不同的 GPS 记录器和 GPS 手机记录，并具有多种采样率。91.5% 的轨迹以密集表示形式记录，例如每 1-5 秒或每 5-10 米每点。该数据集记录了广泛的用户户外运动，不仅包括回家和上班等生活常规，还包括一些娱乐和体育活动，如购物、观光、餐饮、徒步旅行和骑自行车。

实验环境 环境搭建包括以下几个步骤：安装必要的软件：安装 Pycharm，使用 Anaconda3 创建并配置一个虚拟环境 RLTS, 并配置好 TensorFlow（1.8.0）,Python（3.6）。

安装必要的 Python 库：NumPy、math、matplotlib、time 等。

准备数据：使用公共数据集 Geolife。

构建模型。使用强化学习框架和随机策略梯度算法构建深度学习模型。

训练模型。使用训练数据训练模型。

评估模型。使用测试数据评估模型的性能。

调整超参数，并重复训练和评估步骤。

4.2 源码结构分析

data_loader，这个文件主要是用来预处理轨迹数据的，代码中包含了三个不同的函数：包括 latlon2map，points2meter，to_traj，具体来说是对轨迹数据进行坐标转换,latlon2map 函数是用来将经纬度坐标转换为地图投影坐标；接着 points2meter 函数调用了 latlon2map 函数，将轨迹中的所有点的坐标都转换为地图投影坐标；最后 to_traj 函数用于读取文件，将文件中的信息转化为轨迹的三元组（经度、纬度、时间戳）。

utils 文件主要包括三类函数：第一类函数主要包括 sed，ped，dad_op 和 speed_op, 这些函数主要用于计算具体的时空点对用的误差值，用作价值计算和累计最小误差等。第二类函数包括 sed_max_e

等，计算原始轨迹和简化轨迹之间的误差，即论文中谈及的误差，其它三种方式亦同。第三类是可视化, 具体来说，draw_error 函数接受原始轨迹和简化后的轨迹作为参数，并返回最大垂线距离和相关信息。为了计算最大垂线距离，函数遍历原始轨迹中的所有折线段，并调用 draw_sed_op 函数来计算每个折线段的最大垂线距离。最后，函数返回最大垂线距离和相关信息。draw 函数则接受原始轨迹和简化后的轨迹作为参数，并使用这些轨迹信息绘制图像。为了获取最大垂线距离和相关信息，函数调用 draw_error 函数。然后使用最大垂线距离信息在图像中绘制垂线，并使用其他信息绘制轨迹图。

main 文件代码中定义了两个主要函数：run_online 和 run_comp。前者用于验证，这个函数的作用是使用预先训练好的强化学习模型在环境中执行动作，并计算验证错误率。后者用于训练，并将训练好的模型以及验证结果存储。主程序 mian 部分创建了一个环境和一个强化学习模型，进行部分超参数包括学习率、动作。空间大小等的设置并循环调用 run_comp 函数进行训练。在训练过程中，主程序还会定期调用 run_online 函数进行验证。

environment 文件是对 agent 所处环境的定义，reset 函数是重置环境，并返回初始状态，reward_update 实现根据四种不同的情况对误差的计算即环境的奖励更新，delete_heap 代码的作用是在一个堆中删除一个节点，堆是一种特殊的树，并维护堆的性质。step 是环境调整的流程图，利用上述已定义的函数实现对一个旅行轨迹的处理，包括在轨迹上删除一个节点、更新节点的信息、更新奖励值、更新状态信息以及进行边界处理。最后会由 output 函数对处理后的旅行轨迹的输出。它有三种不同的输出模式，针对不同模式打印输出信息并返回最终的误差值。

evaluate 文件对模型进行评估，定义了一个 effectiveness 列表，用来存储每个序列的效果。然后，对于 elist 中的每一个序列（episode），进行以下操作：通过调用 env.reset() 方法来重置环境，并获取步数和观察值。使用一个循环，从缓冲区的大小到步数的范围内的每一个索引（index）进行以下操作：如果当前索引是步数的最后一个元素，那么将 done 设置为 True，否则设置为 False。通过调用 RL.quick_time_action() 方法来获取 action。通过调用 env.step() 方法来更新观察值。最后，通过调用 env.output() 方法来获取序列的效果，并将其存储在 effectiveness 列表中。最后，函数会返回 effectiveness 列表的平均值。

strategy 文件主要用于策略学习，其中 bulid_net 函数定义了一个基于 TensorFlow 的两层神经网络，用于预测行动的概率分布。有 pro_choose_action 函数输入环境状态得到的计算出所有行动的概率, 根据行动的概率随机选择一个行动。另一种策略就是计算出所有行动的概率, 根据行动的概率, 选择最大的一个进行动作选择。其它的一些函数就是激活函数的定义以及奖励值的计算。一个整的学习通过 learn 函数，计算 agent 在本轮训练中的 reward。具体来说，它将每一步的回报进行折扣，并将所有回报进行归一化，以便训练模型。使用 TensorFlow 的 Session.run 函数，通过梯度下降算法调整策略网络的参数，以便让策略网络尽可能地预测出智能体在本轮训练中的最优策略。之后会对参数进行清空，数据清空之后，learn 函数就结束了。该函数会返回本轮训练中 agent 的 reward，然后强化学习算法会进入下一轮训练。在下一轮训练中，agent 会在新的环境中采取新的行动，并获得新的回报，然后算法会再次调用 learn 函数，进行新的训练。

4.3 创新点与改进对比

本算法主要使用代码均来自于原论文开源代码和 github 仓库:tensorflow_practice、introRL、reinforcement-learning、reinforcement-learning-an-introduction 等为参考，并做以下改进和创新尝试：创新一，引入新的策略是提高强化学习算法学习效率的一种有效方法。在更复杂的环境中，新的策略可以使算法更有效地工作，并且在解决更复杂的问题时也会更有优势。在复现过程中，尝试把 Pnet 的策略替换成 Q 学习策略，即对环境建立一个状态-动作值函数（即 Q 函数）。轨迹简化中使用 Q 学习策略的方法与在其他强化学习应用中使用 Q 学习策略的方法类似。首先，建立一个状态-动作值函数（即 Q 函数）来评估在特定状态下采取特定动作的价值。然后，使用随机梯度下降来学习 Q 函数的值。总的来说，复现创新尝试建立一个 Q 函数，然后使用随机梯度下降来学习 Q 函数的值，使得算法能够学习到真实世界的正确行为。

创新二，通过调整奖励值：通过调整奖励值，可以使得某些行为更加有利，而某些行为更加不利。在源代码中，发现奖励机制仅仅与误差的增减量呈线性相关，于是利于增加此幂函数规范，使奖励能够在误差随着误差的减小而逐步给与更大的奖励，反之亦然。再者，通过加入引入惩罚机制：通过引入惩罚机制，可以让智能体在做出不良行为时受到惩罚。

创新三，对于几个重要的超参数，使用贝叶斯优化调参。贝叶斯优化是一种基于贝叶斯定理的参数调整方法，它能够自动根据学习的结果来调整参数的范围，并在参数范围内进行参数搜索。贝叶斯优化可以帮助您更快地找到最优的参数值

5 实验结果分析

本部分对实验所得结果进行分析，详细对实验内容进行说明，实验结果进行描述并分析。

Overall 从数据集中随机抽取 1,000 个轨迹 T，并按照将存储预算 W 从 0.1 x$|T|$ 更改为 0.5x$|T|$。图 3 和图 4 显示了 Geolife 数据集上实时模式的原来算法和复现算法的区别。总的来说，结果清楚地表明复现结果基本达到了原来论文的水平。

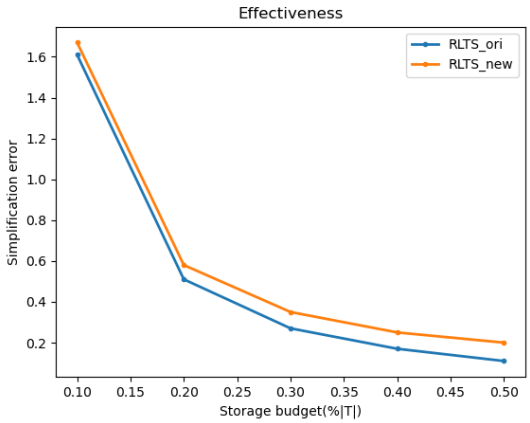

图 2: 简化误差

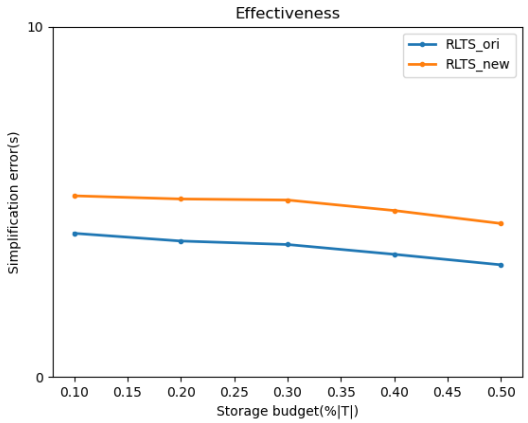

图 3: 运行时间

图 4: 实验结果

HyperParameter 表 1 探究了不同超参数 K 的影响，显示了在不同 K 值下 Geolife 的在线模式的结果。正如预期的那样，随着 k 的增长，运行时间变大。当 k 从 2 增长到 3 时，精度会变得更好，大 k

会使训练模型变得困难，而小 k 会错过一些潜在的候选点。这符合问题解决的直觉。因此，在实验中将 k 设置为 3，因为它在有效性和效率之间取得了良好的平衡。在复现过程中可见不同 K 值得影响基本符号原文。

Parameter	k=2	k=3	k=4	k=5
RLTS_ori_err	4.737	4.642	4.873	4.886
RLTS_new_err	4.828	4.554	4.921	4.725
RLTS_ori_time	0.489	0.543	0.564	0.574
RLTS_new_time	0.548	0.565	0.572	0.535

表 1: 超参数 K 的影响

表 2 探究了不同超参数 J 的影响，即在在线处理模式对 Geolife 可以跳过的最大点数。随着 J 的增加，其有效性会下降，但效率会提高。这是因为随着 J 的增大，RLTS-Skip 会倾向于跳过更多的点，这将导致可能的简化轨迹的空间变小，决定和采取行动的努力变少。在总览中选择 J=2 作为其他实验的默认设置，因为它在有效性和效率之间给出了合理的权衡。

J 大小	J = 0	J = 1	J = 2	J = 3	J = 4
SED 误差	4.938	5.622	6.127	6.834	10.241
运行时间	0.653	0.598	0.543	0.504	0.491
跳过比例	0%	3%	4%	5%	15%

表 2: 超参数 J 的影响

6 总结与展望

轨迹数据是一种捕捉移动物体（如车辆和行人）运动的数据类型，用于导航和交通预测等应用。它通常由 GPS 传感器连续产生，可能会占用大量的存储空间，而且难以查询。为了解决这些问题，轨迹简化经常被用来减少给定轨迹中的点的数量，同时使信息损失最小化。针对这一任务，人们提出了一种名为 RLTS 的基于强化学习的方法，它可以适应不同的动态，并在不同的误差测量中具有良好的通用性。通过对现实生活中的数据集的广泛实验，RLTS 在简化轨迹数据方面被证明是有效和高效的，特别是与现有算法相比。

在未来的研究中，一个有趣的方向是探索如何自适应地选择误差测量，以便更好地适应不同的应用场景。这需要考虑误差测量的各种因素，包括数据的分布、模型的复杂度和计算能力限制、目标任务的要求以及可能的偏差和估计误差的来源。有了这些信息，可以开发自适应策略来动态地选择最合适的误差测量方法。这种研究不仅有助于提高模型的准确性，而且还可以使模型更通用，能够在多种应用场景中使用。在简化轨迹问题中，强化学习方法可以用来尝试不同的策略。具体来说，可以使用各种算法来训练智能体，使其能够从经验中学习最优的策略。这些算法可以使用蒙特卡罗树搜索、蒙特卡罗策略搜索、贝尔曼倒数树或其他算法之一。可以尝试使用这些算法来确定最优的策略，并进一步探索如何有效地在简化轨迹问题中使用这些策略。这可能需要考虑算法的性能、可扩展性、计算复杂度以及在不同应用场景中的表现。通过这种方式，可以进一步改进简化轨迹问题的解决方案，从而帮助解决更复杂的问题。

参考文献

[1] POTAMIAS M, PATROUMPAS K, SELLIS T. Sampling trajectory streams with spatiotemporal criteria

[C]//18th International Conference on Scientific and Statistical Database Management (SSDBM'06). 2006: 275-284.

[2] MUCKELL J, OLSEN P W, HWANG J H, et al. Compression of trajectory data: a comprehensive evaluation and new approach[J]. GeoInformatica, 2014, 18(3): 435-460.

[3] LIN X, JIANG J, MA S, et al. One-pass trajectory simplification using the synchronous Euclidean distance[J]. The VLDB Journal, 2019, 28(6): 897-921.

[4] TRAJCEVSKI G, CAO H, SCHEUERMANNY P, et al. On-line data reduction and the quality of history in moving objects databases[C]//Proceedings of the 5th ACM international workshop on Data engineering for wireless and mobile access. 2006: 19-26.

[5] KATSIKOULI P, SARKAR R, GAO J. Persistence based online signal and trajectory simplification for mobile devices[C]//Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. 2014: 371-380.

[6] BELLMAN R. On the approximation of curves by line segments using dynamic programming[J]. Communications of the ACM, 1961, 4(6): 284.

[7] LONG C, WONG R C W, JAGADISH H. Trajectory simplification: On minimizing the direction-based error[J]. Proceedings of the VLDB Endowment, 2014, 8(1): 49-60.

[8] CHEN M, XU M, FRANTI P. A fast $o(n)$ multiresolution polygonal approximation algorithm for GPS trajectory simplification[J]. IEEE Transactions on Image Processing, 2012, 21(5): 2770-2785.

[9] CHEN Y, JIANG K, ZHENG Y, et al. Trajectory simplification method for location-based social networking services[C]//Proceedings of the 2009 international workshop on location based social networks. 2009: 33-40.

[10] ZHAO Y, SHANG S, WANG Y, et al. Rest: A reference-based framework for spatio-temporal trajectory compression[C]//Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining. 2018: 2797-2806.

[11] KONG W, LIAW C, MEHTA A, et al. A new dog learns old tricks: RL finds classic optimization algorithms[C]//International Conference on Learning Representations. 2018.

[12] WANG Y, TONG Y, LONG C, et al. Adaptive dynamic bipartite graph matching: A reinforcement learning approach[C]//2019 IEEE 35th International Conference on Data Engineering (ICDE). 2019: 1478-1489.

[13] MARCUS R, PAPAEMMANOUIL O. Deep reinforcement learning for join order enumeration[C]// Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management. 2018: 1-4.