arXiv:2303.02640v1 [cs.LG] 5 Mar 2023

通知

本作品已提交给 IEEE，供其考虑发表。版权可能在未通知的情况下转让，届时此版本可能不再可访问。

Swim：适用于运动控制任务的通用、高性能、高效率激活函数

Maryam Abdool[1] 和 Tony Dear[2]

摘要—**激活函数对深度学习算法的性能有着重要影响。尤其是在包括深度强化学习模型在内的更深层模型上，面对具有挑战性的任务时，Swish 激活函数往往优于 ReLU。尽管取得了这一进展，ReLU 仍是首选函数，部分原因在于其效率高于 Swish。此外，与计算机视觉和自然语言处理领域不同，深度强化学习和机器人学领域采用 Swish 等新型激活函数的意愿较低，仍在继续使用 ReLU 等更传统的函数。为解决这些问题，我们提出了 Swim，这是一种通用、高效且性能优于 Swish 的替代函数，并分析了其特性，同时从奖励达成和效率两方面解释其相较于 Swish 的高性能。我们重点测试 Swim 在 MuJoCo 的运动连续控制任务中的表现，因为这些任务具有更复杂的动力学，因此最能受益于高性能、高效率的激活函数。我们还将 Swim 与 TD3 算法结合使用，并结合机器人运动领域阐述这一选择。最后，我们得出结论：Swim 是连续控制运动任务中最先进的激活函数，并建议将其与 TD3 搭配，作为一种实用框架。**

I引言与相关工作 K

深度强化学习可以自动设计用于运动任务的复杂控制器 [9]。在深度强化学习兴起之前，一种常见的技术是为每项运动任务手动设计控制器；这一过程需要机器人的精确动力学模型，而要实现这一点可能并不容易 [9]。除了具有复杂的动力学特性外，运动问题还具有很高的自由度，这使控制器的设计更加困难 [5]。因此，我们采用深度强化学习，自动设计本文所介绍的 MuJoCo 运动连续控制任务的控制器。

在神经网络中，每个神经元都会使用输入权重和偏置对输入进行变换，但这种简单的独立模型无法学习复杂任务[17]。因此，我们引入非线性，使神经网络能够在各种场景下充当非线性函数逼近器，包括强化学习 [19]。实现这一点的方法是在神经网络的隐藏层中使用非线性激活函数，例如修正线性单元（ReLU），

[1]Maryam Abdool 任职于美国纽约州纽约市哥伦比亚大学计算机科学系，邮编 10027maryam.r.abdool@gmail.com[2]Tony Dear 任职于美国纽约州纽约市哥伦比亚大学计算机科学系，邮编 10027tony.dear@columbia.edu

神经网络 [15]。非线性是神经网络的一项重要特性，因为通用逼近定理表明，非线性激活函数可以任意精确地逼近任何连续函数[10]。这对将深度强化学习算法应用于连续控制和运动任务具有重要意义，因为它保证了神经网络具备学习复杂控制器的潜力。

TD3 是一种旨在学习复杂运动控制器的深度强化学习算法 [7]。作为一种深度学习算法，TD3 遵循标准做法，在神经网络的隐藏层中使用 ReLU（一种非线性激活函数）来建模复杂任务 [7]。TD3 与其他算法的区别体现在三个方面：演员-评论家方法、样本效率以及双评论家网络。基于这些原因，我们选择 TD3 而非其他算法。

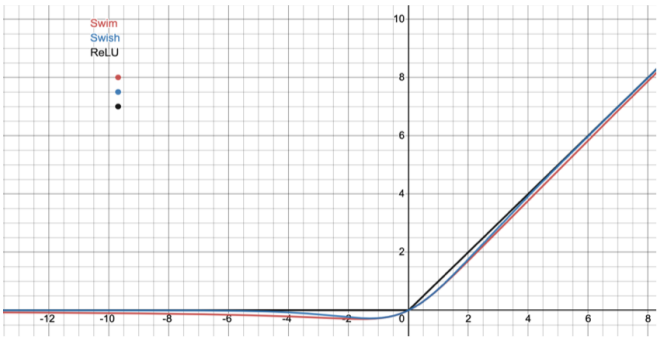
首先，在TD3的演员-评论家神经网络架构中，策略网络被称为演员，因为它用于选择动作；而估计的价值函数网络被称为评论家，因为它会评价演员执行的动作 [19]。演员-评论家方法的优势在于，它可以应用于具有连续动作空间的问题，即MuJoCo的连续控制任务，而Q学习方法无法直接应用于这些问题；对于具有无限动作集合的连续控制任务，Q学习方法必须在无限集合中进行搜索以选择动作，而演员-评论家方法将策略与价值函数分离，从而减少了为每个动作选择所需的大量计算 [19]。其次，TD3基于确定性策略梯度方法，因此具有较高的样本效率 [18]。确定性策略梯度的形式简单，即动作价值函数的期望梯度 [18]。由于形式简单，确定性策略梯度的估计效率远高于其随机对应方法 [18]。最后，TD3使用两个评论家网络而非一个，以解决DDPG和其他Q学习算法中存在的过高估计偏差问题；当对带噪声的价值估计进行最大化时，就会出现这一问题 [7], [20]。

尽管设计新算法是一个重要问题，例如我们采用 TD3 算法来建模控制器，但人们却忽视了对新型非线性激活函数的研究，转而设计更复杂的深度强化学习算法。因此，鉴于研究表明，并非所有非线性激活函数的表现都相同，我们将研究重点放在激活函数领域。例如，人们认为 ReLU 相较于 Sigmoid 更有

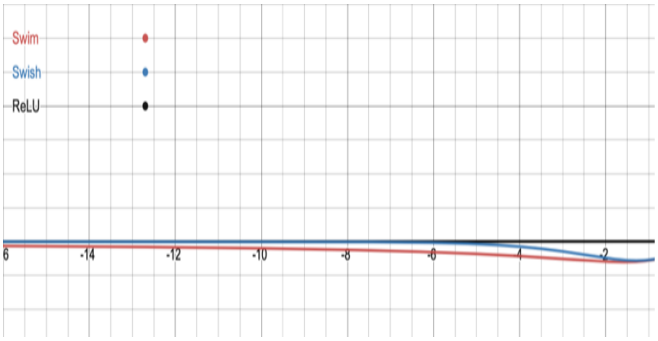
提交至IEEE/RSJ智能机器人与系统国际会议（IROS）审阅的稿件

优势，因为它能降低梯度消失问题出现的可能性，而且是一种更简单、更高效的激活函数 [8]。同样， [16] 推测，Swish 往往优于包括 ReLU 在内的常见基线函数，因为它是非单调的。然而，由于需要进行指数运算，Swish 的效率低于 ReLU；这也是 ReLU 被广泛使用的原因之一，此外它也更为人熟知。如前所述，原始 TD3 实现中选择使用 ReLU 就是一个显著的例子 [11], [7]。

由于这些局限性，我们提出了 Swim，这是一种非单调、平滑的激活函数，其数学定义为$f(x) = x2(\frac{kx}{\sqrt{1+k^2x^2}} + 1)$。Swim 比 Swish 效率更高，也更有可能取得优异表现，因此也比 ReLU 更有可能取得优异表现；如前所述， [16] 和 [6] 已表明，在具有挑战性的任务中，Swish 往往在更深的模型（包括深度强化学习模型）上优于 ReLU。我们在不牺牲任何理想特性的前提下，实现了高效率和优异性能，例如 Swish 凭借平滑性优于基线函数。此前的研究则与此形成对比，例如 [11]设计的 H-Swish，它是 Swish 的高效但不平滑版本。归根结底，我们力求实现优异性能和高效率，因为更好的激活函数意味着更好的运动控制器。鉴于前文提到的复杂动力学和耗时特性，优异性能和效率是运动领域的重要指标。最后，我们还推测，Swim 的成功可能源于其在深度强化学习和运动控制领域所具备的某些内在特性。



(a)



(b)

图1：(a) Swim、Swish 和 ReLU 的曲线。(b) Swim、Swish 和 ReLU 在负值处的放大曲线。

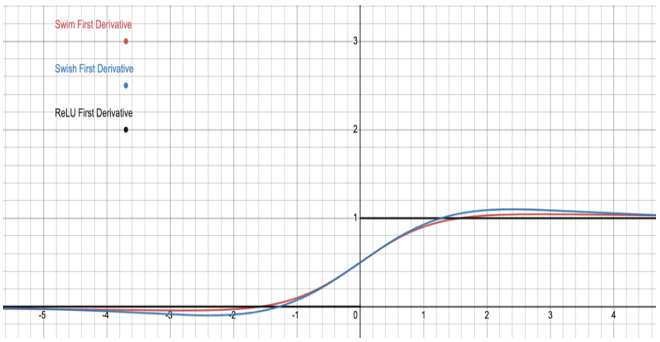
II. SWIM：分析

新提出的激活函数 Swim 及其一阶导数定义如下：

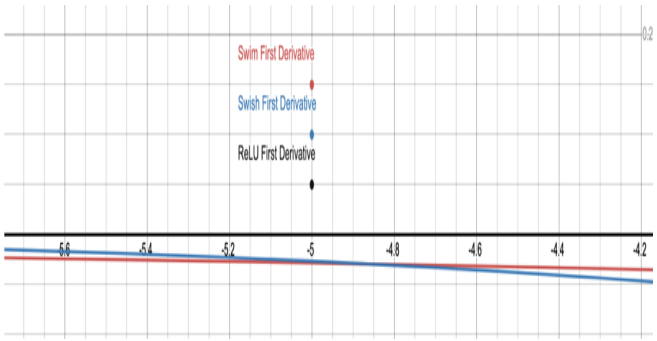
$$f(x) = \frac{x}{2}\left(\frac{kx}{\sqrt{1+k^2x^2}} + 1\right),$$
$$f'(x) = \frac{1}{2}\left(\frac{kx(k^2x^2+2)}{(\sqrt{1+k^2x^2})^3} + 1\right) \quad (1)$$

其中k 是一个可以调节或在训练前设定的常数。我们选择k，使其能够支持我们的分析：Swim 的性能优于 Swish，是因为其固有属性，而不是因为对函数进行了常数缩放；通过改变 β（同样是一个可学习参数），也可以对 Swish 进行类似的缩放。出于这一原因，我们将k = 0.5 设定为近似 Swish 函数（$x\frac{1}{1+e^{-\beta x}}$）在 β = 1 时的取值；原始 Swish 论文即使用该参数运行并报告了结果 [16]。这也意味着该k 常数可能并非最优设置，但我们采取这一折衷以支持我们的分析。这两个函数与 ReLU（$max(0,x)$）之间的差异 [15], 可见图1。

同样，图2展示了 Swim、Swish 和 ReLU 的一阶导数曲线。



(a)



(b)

图2：(a) Swim、Swish 和 ReLU 的一阶导数曲线。(b) Swim、Swish 和 ReLU 在负值处的一阶导数放大曲线。

在下一小节中，我们将把 Swim、Swish 和 ReLU 的行为及其一阶导数纳入我们的

分析。我们重点比较 Swim（对于$k = 0.5$）与 Swish，因为 [16]和 [6]已表明，Swish/SiLU 在各种任务中的表现优于 ReLU 和其他基线函数。

A. Swim 的性质

与 Swish 一样，Swim 没有上界、有下界、非单调且平滑。在这四种性质中，非单调性和平滑性是使 Swish 和 Swim 区别于 ReLU 及其他基线激活函数的性质。如图1所示，非单调性使 Swim 和 Swish 能够在较小的负输入下产生负值。其对梯度的影响如图2所示，在这些较小的负输入下，Swim 和 Swish 的一阶导数大于零。这有助于提高表达能力并改善梯度流 [16]。类似地，Swim 和 Swish 的平滑性对应于平滑的损失景观，使其更易于优化 [16]。

如图1所示，Swim 的负输入相比 Swish 以更慢的速率趋于饱和。对于 $k = 0.5$（Swim）和 $\beta = 1$（Swish），Swim 在所有负输入下都保持这一性质，同时从下方限制 Swish 函数；需要注意的是，这两个函数在负值区域的局部最小值均约为 0.3。从本质上说，这使 Swim 能够针对负输入产生更多且幅度更大的负值，而不会导致梯度爆炸，也不会放弃趋近于零的函数所带来的正则化效应 [8]。尽管可以降低 Swish 的 β parameter 参数值，使其匹配 Swim 的缓慢饱和速率，但这无法与使 Swish 趋近于线性函数这一后果相互分离；根据某些任务的需求，这种后果可能同样不理想且不利，因为它会丧失非线性激活函数的性质。

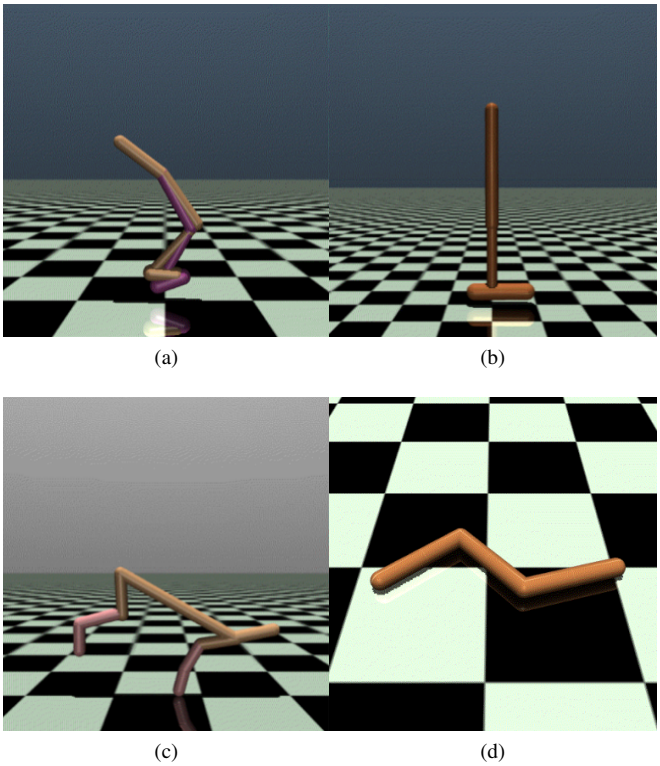
在深度离策略时序差分算法（即 TD3）的背景下，Swim 的平滑饱和特性尤为重要，因为神经网络没有采用特征归一化技术。这会导致激活函数不以零为中心（未经归一化），容易在迅速饱和的点出现梯度消失问题，而 Swish 就是这种情况。换言之，Swim 通过让负输入缓慢饱和，缓解了时序差分算法中的归一化缺失问题。正如 [1]所指出的，部分原因在于训练评论家网络时，动作值函数会被评估两次（$Q(s, a)$ 和 $Q(s', \pi(s'))$），从而产生来自不同分布的动作。此外，观测空间也不一定经过归一化；我们进行测试所用的 MuJoCo 运动连续控制基准任务便是如此。

与 Swish 不同，Swim 不含指数函数 [2]，因此不太容易发生溢出。如公式1所示，Swim 及其一阶导数由平方根和二次项构成。此外，Swim 的计算成本也低于 Swish。尽管这两种函数渐近等价，但在实践中，平方根和二次函数的计算速度快于指数函数，而构成 Swim 函数及其导数的平方根倒数整体而言，计算速度甚至比指数函数更快 [3]。据认为，John Carmack 编写了平方根倒数函数的快速实现 [13],；在 x86 架构上，[3] 表明，他基于平方根倒数的激活函数比 [4]的基于指数的激活函数 ELU 更快。此外，英特尔发布了向量函数的 CPU 性能数据，包括平方根倒数和指数/Exp 函数，[3]已将其汇总，如表 I 所示。根据 x86 架构的不同，表 I 显示，平方根倒数的速度约比 Exp 函数快 1.2 至 2.2 倍。

表 I：向量平方根倒数和指数函数的 CPU 性能（每元素时钟周期）

向量函数单人精度 (EP)	英特尔 Xeon E5-2699 v3 (Haswell AVX2)	英特尔 Xeon E5-2699 v4 (Broadwell AVX2)	英特尔 Xeon 铂金 8180 (Skylake AVX-512)
平方根倒数	0.66	0.64	0.24
Exp	0.81	0.89	0.52
Exp/InvSqrt	1.2×	1.4×	2.2×

III. 方法



(a) (b) (c) (d)

图 3：（a）Walker2d-v2。（b）Hopper-v2。（c）HalfCheetah-v2。（d）Swimmer-v2。

如图 3 所示，我们在 MuJoCo 的四个基准运动控制环境上进行实验：(a) Walker2d-v2、(b) Ant-v2、(c) Hopper-v2 和 (d) HalfCheetah-v2。我们使用 TD3 算法和 Swim 激活函数来解决这些环境中的问题。我们还报告了这些实验的性能和效率。

A. 预备知识

我们将每个环境建模为一个马尔可夫决策过程 (MDP)。在每个离散时间步t，给定状态$s \in S$，智能体根据其策略$\pi: S \to A$选择一个动作$a \in A$。随后，智能体会获得奖励r以及环境的新状态s'。接下来，目标是找到参数为 ϕ 的最优策略 π_ϕ。

我们选择双延迟深度确定性策略梯度算法 (TD3) [7] 来学习最优策略。TD3 基于深度确定性策略梯度 (DDPG) 算法 [12], , 由一个演员和两个评论家网络组成，其中动作价值函数$Q^{\phi_1}(s,a)$和$Q^{\phi_2}(s,a)$充当两个评论家神经网络的近似器。TD3 通过最小化均方贝尔曼误差来学习两个 Q 函数Q^{ϕ_1}和Q^{ϕ_2}。

在每个时间步，贝尔曼误差损失函数中的目标根据以下方程更新：

$$y = r + \gamma \min_{i=1,2} Q_{\theta_i'}(s', \pi_{\phi'}(s') + \varepsilon),$$
$$\varepsilon \sim clip(N(0,\sigma), -c, c) \tag{2}$$

其中，γ是折扣因子，θ_i'是目标 Q 网络的参数，ϕ'是目标演员网络的参数，ε是裁剪噪声，而（$-c$，c）是用于裁剪目标策略噪声的超参数值。也就是说，使用两个目标值中较小的那个来更新目标，这两个目标值由 Q 函数给出。

我们使用作者以 PyTorch 编写的 TD3 原始实现，因为原论文也着重于在 MuJoCo 控制任务上运行 TD3，其中包括本文所使用的任务。因此，我们也采用相同的最优超参数，只在实验中更改激活函数。

B. 应用激活函数

我们在 PyTorch 中自行实现 Swim 激活函数，并将其用于演员和评论家网络的所有隐藏层。应用 Swish 激活函数时，我们也采用相同的方法，不过 Swish 使用的是 PyTorch 中底层的数值稳定 Sigmoid torch 函数。我们没有实现数值稳定版本，而是使用 torch 函数直接翻译公式1中所写的 Swim 函数。数值稳定函数或许能提升 Swim 的性能，但我们将其留待未来研究。此外，为提高精度，我们使用双精度而非单精度，并在搭载 ARM架构的机器上运行实验。

如第一节所述，我们仅将 Swim 与 Swish 进行比较，因为 [16] 和 [6]已表明，在各种任务中，Swish/SiLU 的表现优于 ReLU 及其他基线函数。我们还选择 Swim 作为该问题的激活函数，原因已在第一节和第二节A中说明；此外，简单的运动学控制器模型，例如双曲正切(x) [21], , 不足以作为该问题的激活函数，因为运动控制任务呈现出复杂动力学。

C. 评估

按照深度强化学习领域的标准做法，我们使用性能曲线比较 Swim 与 Swish 的性能。我们还测量演员网络的推理时间，以比较这两种激活函数的效率，从而有效地将效率评估与性能评估分离。

IV. 结果

对于性能指标，我们在每个环境中使用所选激活函数运行 TD3，进行 100 万个时间步；每隔 5000 个时间步，在不添加探索噪声的情况下记录连续 10 个回合的平均奖励。随后，我们报告每个环境在 5 个随机种子下的性能曲线。我们还报告这 5 个种子中的最大平均回报，以比较不同激活函数的结果。

对于效率指标，我们计算每个环境前 100 次迭代的推理时间，并报告结果。

A. 性能

表 II：在 5 次试验、每次 100 万个时间步中取得的最大平均回报。 ± 表示试验结果的一个标准差。

环境	Swish		Swim		改进
Walker2d-v2	3774.668 149.779	±	4387.442 453.526	±	16%
Hopper-v2	3450.055 129.203	±	3431.081 110.394	±	0%
HalfCheetah-v2	10519.091 872.693	±	11130.364 652.716	±	5%
Swimmer-v2	132.838 8.146	±	134.503 6.806	±	1%

学习曲线见图4，最大平均回报列于表 II。结果表明，在所有任务中，Swim 的最终性能和学习速度均与 Swish 持平或优于 Swish。根据曲线和两种激活函数的最大平均回报，Swim 在 4 个环境中的 3 个表现优于 Swish：Walker2d-v2、HalfCheetah-v2 和 Swimmer-v2；不过，在 Walker2d-v2 和 HalfCheetah-v2 环境中的提升更为显著，提升幅度分别为 16% 和 5%。

结果支持第二节中提出的假设和分析；对于$k = 0.5$，预计 Swim 不会始终优于 Swish，因为选择该常数的目的，是使 Swim 在 $\beta = 1$时近似于 Swish。探索k的其他取值，或许能够带来更高的性能。

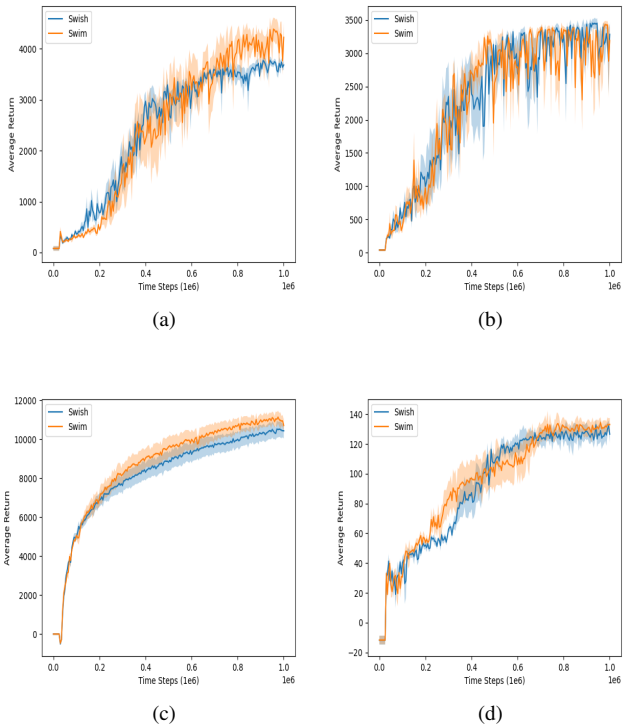


图4：MuJoCo 连续控制任务的学习曲线：(a) Walker2d-v2、(b) Hopper-v2、(c) HalfCheetah-v2 和 (d) Swimmer-v2。阴影区域表示 5 次试验中平均评估值的半个标准差。

然而，在某些环境中，Swim 仍然优于 Swish。正如第二节所述，我们推测这是因为 Swim 的负值饱和速度比 Swish 慢，从而缓解了 TD3 和其他深度强化学习算法缺少特征归一化技术这一问题；此外，我们的环境中的观测空间也未经归一化。

B. 效率

表III：各环境前100次迭代的推理时间。± 表示迭代过程中的一个标准差。

环境	Swish（毫秒）	Swim（毫秒）	改进
Walker2d-v2	0.810 ± 0.066	0.687± 0.077	17.9%
Hopper-v2	0.720 ± 0.055	0.666 ± 0.056	8.1%
HalfCheetah-v2	0.826 ± 0.057	0.733 ± 0.064	12.7%
Swimmer-v2	0.737 ± 0.060	0.644± 0.045	14.4%

表III 报告了四种环境各自的推理时间。表中显示，Swim 在所有环境中的效率始终优于 Swish，提升幅度为 8.1% 至 17.9%。与最大平均回报的结果相反，效率在所有环境中的提升都很显著。

结果支持我们在第二节中的分析：我们推测 Swim 比 Swish 更高效，因为平方根倒数的计算开销低于指数函数。由于我们没有实现利用 Swim 特性的自定义反向传播过程，因此没有测量反向传播时间，尽管平方根倒数出现在 Swim 函数的一阶导数中，如公式1所示。正如 [3] 在其论文中所做的那样，这一特性可用于优化反向传播过程的效率。

第五节：讨论与未来工作

根据我们的分析和结果，Swim 的主要优势在于，与 Swish 相比，其计算复杂度更低；虽然 Swim 也达到了最先进的性能，并且有时在学习速度和奖励达成方面超过 Swish，但在所有任务中，其效率均显著优于 Swish。Swim 的低计算复杂度使其适合作为通用激活函数，因此我们呼吁未来的研究者在深度强化学习和机器人学领域之外的其他模型上测试并优化 Swim。对于深度强化学习和机器人学领域，Swim 尤其适用，因为它兼具高性能与高效率。运动连续控制任务能够受益于采用负向饱和输入激活函数的深度离策略时序差分学习算法，从而缓解缺乏特征归一化技术以及观测空间未归一化的问题；此外，运动连续控制任务也能受益于高效的激活函数，因为这类任务具有复杂动力学和较高的自由度，需要较长的训练时间，而高效激活函数可以加快运行速度。这些结果也引出了领域特定激活函数这一理念，而当前研究尚未聚焦于此；当前的研究趋势是设计可泛化激活函数，并在各种任务上进行测试，却没有探究哪些特性更有利于某些领域。我们希望未来能有更多关于领域特定激活函数的研究，尤其是在深度强化学习和机器人学领域，因为这些领域对激活函数的研究相对较少。

关于 Swim 仍有进一步的研究空间。在测试和优化方面，我们只在 CPU 上测试了我们的函数，尚未在 GPU 上进行测试。这也意味着，可以像 [14] 在其论文中所做的那样，通过基于 CUDA 的实现来优化 Swim，进一步提高效率。我们也只在 ARM架构 上测试了结果并报告效率，尽管 [3] 此前已报告了 x86 架构上平方根倒数的快速计算结果，英特尔也公布了平方根倒数的快速性能表现，我们已在本文中对此进行了报告。此外，可以像 [3] 在其论文中所做的那样，为 Swim 实现反向传播过程，利用平方根倒数提高其效率。在激活函数本身方面，我们没有实现数值稳定版本，不过我们报告的性能表明，Swish 在学习速度方面与 Swish 相当或更胜一筹

而且无需这样做即可达到相同或更高的奖励达成效果。我们推测，数值稳定版本可以进一步提升 Swim 的性能。我们还推测，采用不同的 k 值可能会提升 Swim 的性能。基于这些发现，我们计划通过在 GPU 和 ARM架构 上进行测试、编写基于 CUDA 的实现、为 Swim 设计自定义反向传播过程、开发数值稳定版本，以及探索k的不同取值，来探索并提升 Swim 的性能和效率。

附录

本研究论文所使用的代码和其他补充材料，包括我们对 Swim 的实现，可在 https://github.com/maryam-abdool/Swim- Control 获取

参考文献

[1] Aditya Bhatt、Max Argus、Artemij Amiranashvili 和 Thomas Brox。Crossnorm：用于离策略时序差分强化学习的归一化。*arXiv*预印本 *arXiv:1902.05605*，2019年。[2] Pierre Blanchard、Desmond J Higham 和 Nicholas J Higham。准确计算对数和指数与 Softmax 函数。*IMA*数值分析杂志，41(4):2311–2330，2021年。[3] Brad Carlile、Guy Delamarter、Paul Kinney、Akiko Marti 和 Brian Whitney。通过逆平方根线性单元（ISRLU）改进深度学习。*arXiv*预印本 *arXiv:1710.09967*，2017 年。[4] Djork-Arn´e Clevert 、 Thomas Unterthiner 和 Sepp Hochreiter。通过指数线性单元（ELU）实现快速、准确的深度网络学习。*arXiv*预印本 *arXiv:1511.07289*，2015年。[5] Yan Duan、Xi Chen、Rein Houthooft、John Schulman 和 Pieter Abbeel。连续控制深度强化学习基准测试。载于国际机器学习会议，第1329–1338页。PMLR，2016年。[6] Stefan Elfwing、Eiji Uchibe 和 Kenji Doya。用于强化学习中神经网络函数逼近的 Sigmoid 加权线性单元。神经网络，107:3–11，2018年。[7] Scott Fujimoto、Herke Hoof 和 David Meger。解决演员—评论家方法中的函数逼近误差。载于国际机器学习会议，第1587–1596页。PMLR，2018年。[8] Xavier Glorot、Antoine Bordes 和 Yoshua Bengio。深度稀疏整流神经网络。载于第十四届人工智能与统计国际会议论文集，第315–323页。JMLR研讨会与会议论文集，2011年。[9] Tuomas Haarnoja、Sehoon Ha、Aurick Zhou、Jie Tan、George Tucker 和 Sergey Levine。通过深度强化学习学会行走。*arXiv*预印本 *arXiv:1812.11103*，2018年。[10] Kurt Hornik、Maxwell Stinchcombe 和 Halbert White。多层前馈网络是通用逼近器。神经网络，2(5):359–366，1989年。[11] Andrew Howard、Mark Sandler、Grace Chu、Liang-Chieh Chen、Bo Chen、Mingxing Tan、Weijun Wang、Yukun Zhu、Ruoming Pang、Vijay Vasudevan 等。探索 MobileNetV3。载于*IEEE/CVF*计算机视觉国际会议论文集，第1314–1324页，2019年。[12] Timothy P Lillicrap、Jonathan J Hunt、Alexander Pritzel、Nicolas Heess、Tom Erez、Yuval Tassa、David Silver 和 Daan Wierstra。深度强化学习下的连续控制。*arXiv*预印本 *arXiv:1509.02971*，2015年。[13] Chris Lomont。快速逆平方根。技术报告 32，2003。，[14] Diganta Misra。Mish：一种自正则化的非单调神经激活函数。*arXiv*预印本 *arXiv:190*848550，2019*08681*，4 2 10。( ): – 。[15] Vinod Nair 和 Geoffrey E Hinton。修正线性单元改进受限玻尔兹曼机。载于*ICML*，2010年。[16] Prajit Ramachandran、Barret Zoph 和 Quoc V Le。探索激活函数。*arXiv*预印本 *arXiv:1710.05941*，2017年。[17] J¨urgen Schmidhuber。神经网络中的深度学习：综述。神经网络，61:85–117，2015年。

[18] David Silver、Guy Lever、Nicolas Heess、Thomas Degris、Daan Wier- stra 和 Martin Riedmiller。确定性策略梯度算法。载于国际机器学习会议，第387–395页。PMLR，2014年。[19] Richard S Sutton 和 Andrew G Barto。强化学习：导论。麻省理工学院出版社，2018年。[20] Sebastian Thrun 和 Anton Schwartz。将函数逼近用于强化学习时的问题。载于*1993*年连接主义模型暑期学校论文集，新泽西州希尔斯代尔。*Lawrence Erlbaum*，第6卷，第1–9页，1993年。[21] Dongdong Xie、Shenquan Wang 和 Yuenan Wang。基于改进运动学控制器算法的差速驱动移动机器人轨迹跟踪控制。载于*2018*年中国自动化大会（*CAC*），第2675–2680页。IEEE，2018年。