基于GameGAN的动态环境模拟

Seung Wook Kim[1,2,3*] Yuhao Zhou[2†] Jonah Philion[1,2,3] Antonio Torralba[4] Sanja Fidler[1,2,3*]
[1]NVIDIA [2]多伦多大学 [3]Vector Institute [4] 麻省理工学院
{seungwookk,jphilion,sfidler}@nvidia.com henryzhou@cs.toronto.edu torralba@mit.edu
https://nv-tlabs.github.io/gameGAN

摘要

对于任何机器人系统，模拟都是关键的一环。为正确地进行模拟，我们需要编写复杂的环境规则，如：动态代理的行为，以及每个代理的行为会如何影响其他代理的行为。在本文中，我们旨在通过简单地观察代理与环境的交互，以学习得到一个模拟器。我们专注于图形游戏，将其作为真实环境的代理。我们提出GameGAN，一种生成式模型，能够通过训练，理解游戏规则和键盘动作，学习如何在视觉上模仿特定游戏。代理按下一个按键，GameGAN就能借助精巧设计的生成式对抗网络，“渲染”接下来的屏幕内容。我们的方法具备超越当前方法的关键优势：我们设计了一个记忆模块，以构建环境的内部映射，让代理能够返回到早前访问过的位置，且具有高度的视觉一致性。此外，GameGAN能够分解图像中的静态和动态组件，使模型的行为更易于理解，且与需要对动态元素进行显式推理的下游任务相关。这使得许多有趣的应用程序成为了可能，例如交换游戏的不同组件，以构建前所未有的新游戏。

1. 前言

在部署到现实世界之前，人工代理需要在具有挑战性的模拟环境中进行广泛的测试。因此，设计完善的模拟器非常重要。传统上，这需要编写程序模型，以生成有效且多样化的场景，并由复杂的行为树来指定场景中的每个因素如何行动，以及对其他行为者（包括自代理）所做出的行为作何反应。但是，编写包含大量不同场景的模拟器非常耗时，并且需要技术水平高超的图形专家。而通过观察现实世界的动态目标来学习模拟，是最具可扩展性的发展方向。

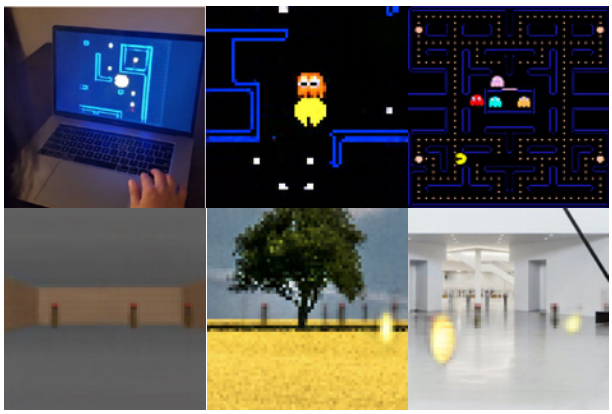


图1：如果您看左上图的那一位，可能会认为她在玩岩谷彻的《吃豆人》游戏，但事实并非如此！她实际上正在玩的是由GAN生成的新版本《吃豆人》。本文中，我们将介绍GameGAN，它能够通过对很多游戏回合的观察，学习如何复制游戏。此外，我们的模型能够分解背景与动态对象，让我们能够调换下行中间及右侧图像中所示的组件，以创建新游戏。

当前的大量研究都致力于学习行为模型[2, 33, 19, 4]。但是，这通常需要进行大量监督工作，例如访问代理的真实轨迹。我们旨在让模拟器通过简单地观察代理与环境之间的交互来学习。为简化问题，我们将其视为2D图像生成问题。已有观察到的图像帧序列，以及代理采取的相应行动，我们希望模拟图像的创建，就像是从能够对代理的行动做出反应的真实动态环境中“渲染”出来的一样。

为实现这一目标，我们把关注重心放在了图形游戏之上，将其作为真实环境的代理。它代表着更简单且更受控的环境。我们的目标是通过使用训练好的模型(Learned model)，在视觉上模仿游戏，以在测试时替换图形引擎。这是一个极具挑战性的问题：不同的游戏有着不同数量的组件，物理动态方面也不尽相同。此外，许多游戏都要求环境长期保持一致。例如，设想有一款游戏，其中一个代理要在迷宫中进行导航。

*相应的联系方式：{seungwookk,sfidler}@nvidia.com
†YZ在NVIDIA实习期间参与了此项目。

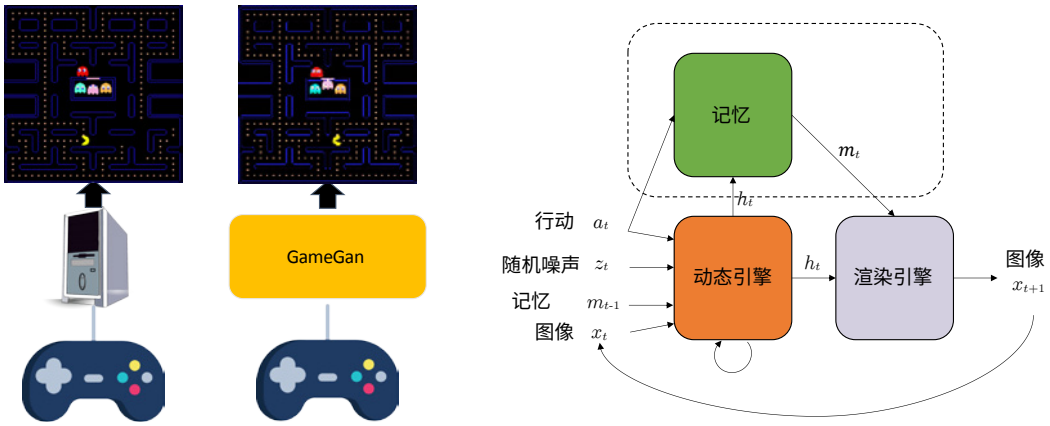


图2：GameGAN概述：我们的目标是用神经网络代替游戏引擎。GameGAN由三个主要模块组成。**动态引擎**由RNN实现，并包含每经过时间t就会更新的世界状态。它能够选择性地写入和读取外部**记忆模块**$\mathcal{M}$。最后，**渲染引擎**用于解码输出图像。所有模块都是神经网络，且经过端到端训练。

当代理离开某个位置，随后又返回时，会希望场景看起来与之前一致。在可视化SLAM中，检测环路闭合（返回到先前位置）的挑战性已经广为周知，更不用说生成一个一致环境的难度了。最后，还有一点也很重要，那就是在游戏中，预定义行为和随机行为通常并存，且对随机行为进行建模是特别困难的。

在本文中，我们将介绍GameGAN，这是一种生成式模型，可用于模拟特定游戏。GameGAN会在训练过程中了解游戏规则和键盘操作，并旨在通过对动作的调节（如代理按下按键）来预测下一帧。它能够直接从大量的图像和动作对中学习，而无需访问底层游戏逻辑或引擎。我们在近期推出的旨在解决类似问题的世界模型(World Model)[13]方面取得了一些进展。通过利用生成式对抗网络[10]，我们就能够产出更高质量的结果。此外，不同于[13]使用简单的条件解码器，GameGAN具有设计精良的架构。特别是，我们提出了一个新的记忆模块，其有助于模型构建环境的内部映射，从而使代理能够返回先前访问过的、具有高度视觉一致性的位置。此外，我们还将介绍一种专门设计的解码器，该解码器能够学习如何分解图像中的静态和动态组件，使模型的行为更易于理解，还让我们能够通过交换不同的组件，对当前游戏进行修改。

我们在《吃豆人》和VizDoom环境的修改版上，对GameGAN [20]进行了测试，并建议通过一些综合任务，进行定量和定性评估。我们进一步引入了一项“回家”任务，以测试训练好的模拟器的长期一致性。请注意，GameGAN支持多种应用程序，例如将给定游戏从一个操作系统转移到另一操作系统，而无需重写代码。

2. 相关研究工作

生成式对抗网络：在GAN中[10]，生成器(generator)和鉴别器(discriminator)会进行对抗博弈，帮助生成器产出近乎真实的输出。为控制生成的输出结果能够达到预期，会提供类别标签[28]、图像[18, 25]、字幕[34]或遮罩[32]作为生成器的输入。例如[39]借助循环一致性损失[41, 21]，将源样式迁移到目标视频，来合成新视频。请注意，这相较于我们研究工作中考虑的问题更为简单，因为已提供目标视频的动态内容，且仅需更改视觉样式。在本文中，我们仅考虑生成动态内容本身。我们采用GAN框架，并以用户操作作为生成未来图像帧的条件。据我们所知，我们的这项研究是首个使用基于动作信息的GAN模拟游戏模拟器的研究。

视频预测：我们的研究工作与根据给定先前帧序列来预测未来帧的视频预测任务相似。有一些研究[36, 6, 31]通过训练循环编码器来解码未来帧。对于大多数方法，训练都会存在重建损失，导致预定义流程生成了模糊的图像帧，且通常无法很好地处理随机行为。这些错误通常会随着时间的推移而累积，并导致预测质量低下。在将生成图像缩放到更高分辨率方面，Action-LSTM模型[6, 31] 取得了成功，但它无法处理《吃豆人》等环境中复杂的随机性。最近，[13, 8]提出了基于VAE的框架，以期处理任务的随机性。但是，生成的视频模糊，生成的帧也倾向于忽略某些细节。早前的一些研究[9, 24, 37, 7]应用了GAN损失函数。[9]采用对抗损失，分解不同视频中的内容与姿势。在[24]中，

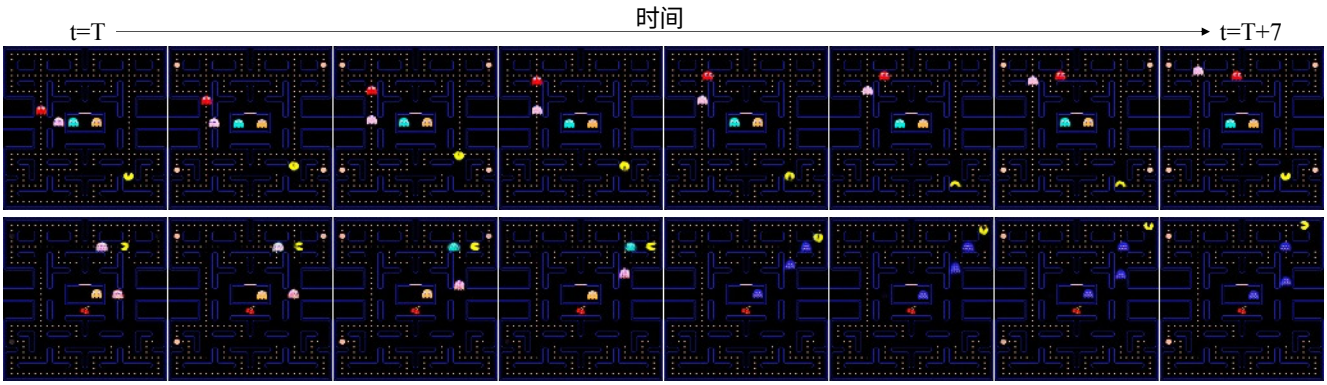


图3：人类玩基于《吃豆人》官方版训练的GameGAN的截图[1]。GameGAN学会了生成具有视觉一致性的模拟，并能够很好地学习游戏的动态性。在下行截图中，玩家吃下一粒强化道具后，将鬼魂变成了紫色。请注意，在吃豆人吃下强化道具前，鬼魂会追赶吃豆人，但在吃豆人吃下强化道具后，鬼魂则会转而逃跑。

VAE-GAN [23]结合模型，用于生成视频的下一帧。我们的模型与这些研究的不同之处在于，除生成下一帧外，GameGAN还学习了环境的内在动态元素。[12]通过解析已知图像帧，学出一种基于规则的、能够找到最为可能的下一图像帧的搜索引擎。GameGAN进一步学习了图像帧中的所有内容，以及直接生成图像帧的方法。

世界模型(World Model)：在基于模型的强化学习中，人们通过与环境的交互，学习动态模型。世界模型[13]则利用习得的模拟环境来训练RL代理。近期，世界模型已用于在并发研究中生成Atari游戏[1]。与这些模型的主要区别在于架构设计：记忆模块能够更好地实现长期一致性，设计完善的解码器能够分解游戏中的静态和动态组件。

3. GameGAN

我们感兴趣的是训练能够模拟环境预定义和随机性特征的游戏模拟器。我们还尤为关注图像空间中的行动制约性模拟器，其中包含一个自主的代理，该代理在时间t处，根据给定的行为$a_t \sim \mathcal{A}$进行移动，并生成新的观测值x_{t+1}。我们假设还有一个随机变量$z_t \sim \mathcal{N}(0; I)$对应于环境中的随机性。基于图像$x_{1:t}$的历史记录以及$a_t$和$z_t$，GameGAN会预测下一幅图像$x_{t+1}$。GameGAN由三个主要模块组成：**动态引擎**（第3.1节）作为内部状态变量，以a_t和z_t作为输入，更新当前状态。对于需要长期一致性的环境，我们可以选择性地使用外部**记忆模块**（第3.2节）。最后，**渲染引擎**（第3.3节）根据动态引擎的状态，生成输出图像。它可以由简单的卷积解码器来实现，也可以与记忆模块耦合，以分解静态和动态元素，同时确保长期一致性。我们使用对抗的损失函数，以及已有工作提出的时间周期损失函数（第3.4节），对GameGAN进行训练。不同于某些研究通过序列化的训练来确保稳定性 [13]，GameGAN是端到端训练的。我们会在补充材料中提供有关每个模块的更多详细信息。

3.1. 动态引擎

GameGAN必须学习环境的各方面如何根据给定的用户操作进行更改。例如，它需要了解某些动作是不可能实现的（例如穿墙），以及其他目标会因该动作而做出何种行为。我们将学习这种事件的主要组件称为**动态引擎**（参见图2中的插图）。它需要结合过去的历史，以产出一致的模拟。因此，在Chiappa等人[6]的设计启发下，我们选择将其实现为动作制约性LSTM [15]：

$$v_t = h_{t-1} \odot \mathcal{H}(a_t, z_t, m_{t-1}), s_t = \mathcal{C}(x_t) \quad (1)$$

$$i_t = \sigma(W^{iv} v_t + W^{is} s_t), f_t = \sigma(W^{fv} v_t + W^{fs} s_t), \\ o_t = \sigma(W^{ov} v_t + W^{os} s_t) \quad (2)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W^{cv} v_t + W^{cs} s_t) \quad (3)$$

$$h_t = o_t \odot \tanh(c_t) \quad (4)$$

其中h_t、a_t、z_t、c_t、x_t是时间步t时的隐藏状态、动作、随机变量、单元状态和图像。m_{t+1}是上一步中检索到的记忆向量（如使用记忆模块），i_t、f_t、o_t为输入、忘记和输出门。a_t、z_t、m_{t-1}和h_t被代入v_t中，s_t是图像

[1]PAC-MAN™& ©BANDAI NAMCO Entertainment Inc.

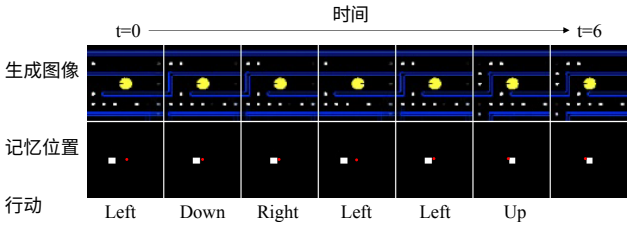


图4：看管式记忆位置a的可视化：标注红点的位置为中心，有助于可视化。请注意，我们要学习记忆位移，因此用户操作并不总是与记忆位移的方式一致。在这种情况下，“*Right*”将a移向左，“*Left*”将a移向右。当采取无效措施时，它也学会了不移动。

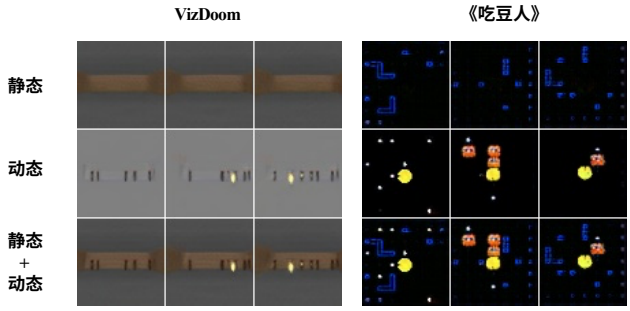


图5：该示例显示了如何使用GameGAN，在VizDoom和《吃豆人》游戏中分解静态和动态元素。静态组件通常包括环境对象，例如墙壁。动态元素通常是可随游戏进行而改变的对象，例如食物和其他非玩家角色。

x_t的编码。$\mathcal{H}$是MLP，$\mathcal{C}$是卷积编码器，W是权重矩阵，⊙表示哈达玛积(Hadamard product)。引擎负责维护LSTM的标准状态变量h_t和c_t，其中包含时间t时环境各方面的信息。它基于a_t、z_t、m_{t-1}和x_t计算得出状态变量。

3.2. 记忆模块

假设我们有意模拟一个环境，有一个代理在其中进行导航。这需要长期的一致性，即当代理稍后返回同一位置时，模拟场景（例如建筑物、街道）不应发生变化。对于RNN等典型模型而言，这是一项艰巨的任务，因为1）模型需要记住其在隐藏状态下生成的每个场景，且2）设计可实现如此长期一致性的损失函数并非易事。我们建议使用由神经图灵机（Neural Turing Machine，NTM）激励的外部记忆模块[11]。

该记忆模块有一个记忆块$\mathcal{M} \in \mathbb{R}^{N\times N\times D}$，且在时间$t$，看管位置$a_t \in \mathbb{R}^{N\times N\times D}$。$\mathcal{M}$包含$N$×$N$个$D$维向量，其中$N$是块的空间宽度和高度。直观地，$a_t$是自我中心式代理所处的当前位置。$\mathcal{M}$ 通过随机噪声~N（0，I）进行初始化，a_o通过0s进行初始化，除中心位置（N/2, N/2）设置为1。在每个时间步，记忆模块都会计算：

$$w = \text{softmax}(\mathcal{K}(a_t)) \in \mathbb{R}^{3\times 3} \quad (5)$$

$$g = \mathcal{G}(h_t) \in \mathbb{R} \quad (6)$$

$$\alpha_t = g \cdot \text{Conv2D}(\alpha_{t-1}, w) + (1-g)\cdot \alpha_{t-1} \quad (7)$$

$$\mathcal{M} = \text{write}(\alpha_t, \mathcal{E}(h_t), \mathcal{M}) \quad (8)$$

$$m_t = \text{read}(\alpha_t, \mathcal{M}) \quad (9)$$

其中$\mathcal{K}$、$\mathcal{G}$和$\mathcal{E}$是小型MLP。w是依赖于当前动作习得的位移内核，该内核用于a_{t-1}的位移。在某些情况下，这种位移是不应该发生的（例如，遇到死胡同就无法再继续前进）。在h_t的帮助下，我们还习得了一个控制变量g∈[0,1]，其能够确定a是否应位移。习得的$\mathcal{E}$能够从隐藏状态中提取要写入的信息。最后，类似其他神经记忆模块[11]，写入和读取操作会对由α指定的记忆位置进行软访问。借助此基于位移的记忆模块，可在强制执行局部移动时，使模型不受块$\mathcal{M}$大小的限制。因此，我们可在测试时使用任意大小的块。图4演示了习得的记忆移位。由于模型可自由地将动作分配给不同的内核，因此习得的位移并不总是与人类的行为相对应。我们可以看到，*Right*被分配为左移，因此*Left*被分配为右移。通过控制变量g，它还习得了在给出无效操作（例如穿墙）时不发生位移。

在我们的案例中，强化长期一致性是指记住生成的静态元素（例如背景），并在需要时适当地对其进行检索。因此，使用记忆模块的好处就在于内部存储静态信息。除新颖的周期损耗（第3.4.2节）外，我们还在渲染引擎架构中引入了归纳偏置（第3.3节），以助力分解静态和动态元素。

3.3. 渲染引擎

（神经）渲染引擎负责在给定状态ht下渲染模拟图像x_{t+1}。它可以简单地用标准的转置卷积层实现。但是，我们还引入了一种特殊的渲染引擎架构（图6），通过习得如何产出分解场景，以确保长期一致性。在第4节中，我们将比较每种架构的好处。

专用渲染引擎将向量c = {c^1, …, c^k}的列表作为输入。在本研究中，我们令K = 2，c = {m_t, h_t}。每个向量c^k对应一种类型的实体，并经历以下三个阶段（参见图6）。首先，将c^k馈入卷积网络，以生成属性映射$A^k \in \mathbb{R}^{H_1\times H_1\times D_1}$和对象映射$O^k \in \mathbb{R}^{H_1\times H_1\times 1}$。它还被馈入线性层，

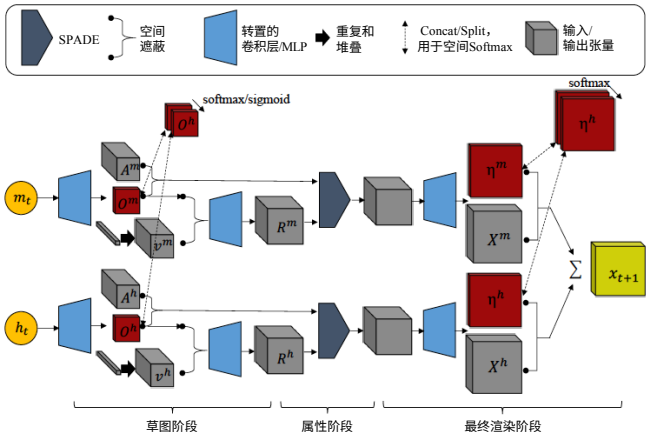


图6: 用于分解静态和动态组件的**渲染引擎**。详见第3.3节。

以获得第k个组件的类型向量$v^k \in \mathbb{R}^{D1}$。将所有组件的O串联，并通过S型曲线馈送，以确保$0 \le O^k[x][y] \le 1$，或通过一个空间softmax函数，以确保对于所有x, y的值，均有$\sum_{k=1}^{K}[x][y] = 1$。在每个位置，生成的对象映射乘以类型向量v^k，并传入卷积，以生成$R^k \in \mathbb{R}^{H2\times H2\times D2}$。这是第$k$个类型对象位置的示意图。但是，每个对象可能具有不同的属性，例如不同的样式或颜色。因此，它会经历一个属性阶段，其中张量由一个SPADE层[32, 16]进行了变换，其中屏蔽的属性映射$O^k \odot A^k$被指定为情境信息。它进一步传入几个转置的卷积层，最后进行类似于草图阶段的注意力(attention)流程，在此阶段，级联组件通过空间softmax函数，以获得精细蒙版(fine mask)。直觉告诉我们，在绘制单一对象后，需要决定要绘制对象的"深度"顺序，以解决遮蔽问题。让我们以η^k表示精细蒙版，以X^k表示最终张量。此后，通过对所有组件求和，$x = \sum_{k=1}^{K} \eta^k \odot X^k$，获得最终图像。因此，我们的神经渲染引擎架构有助于借助时间周期损失函数，从记忆向量和隐藏状态中提取不同的信息（第3.4.2节）。我们还将在补充材料的第A.5节中介绍容量更高、能够产出更高画质图像的版本。

3.4. 训练GameGAN

对抗训练已成功用于图像和视频合成任务。GameGAN利用对抗训练，可以学习环境动态元素，并产出具有时间连贯性的逼真模拟器的输出。对于某些需要长期一致性的情况，我们提出了一个时序性周期损失函数，分解静态和动态组件，以习得如何记住其生成的内容。

3.4.1 对抗损失函数

它由三个主要部分组成：单一图像鉴别器、动作制约性鉴别器和时间鉴别器。

单一图像鉴别器：为确保生成的每帧都是真实的，单一图像识别器和GameGAN模拟器会进行对抗性博弈。

动作制约性鉴别器：GameGAN必须忠实地反映代理所采取的行动。我们给出了三对动作条件鉴别器：(x_t, x_{t+1}, a_t), (x_t, x_{t+1}, a_t) 和$(\hat{x}_t, \hat{x}_{t+1}, a_t)$。$x_t$表示真实图像，$\hat{x}_t$ 表示生成的图像，而 $\bar{a}_t \in \mathcal{A}$为负面行为样本 $\bar{a}_t \neq a_t$。鉴别器的工作是判断两个帧在动作方面是否一致。 因此，要欺骗鉴别器，GameGAN必须生成能够反映行动的近乎真实的预测图像帧。

时间鉴别器：环境中的不同实体可能表现出不同的行为，且在部分观察的状态下也会出现或消失。为模拟时间上一致的环境，在生成下一状态时，必须考虑过去的信息。因此，我们将时间鉴别器实施为3D卷积网络。它需要几帧作为输入，并确定其为真实序列还是生成的序列。

由于已知条件GAN架构[27]能够习得简化分布，而忽略隐代码[40, 35]，因此我们添加了信息正则化[5]，将潜在代码z_t和对(x_t, x_{t+1})之间的交互信息$I(z_t, \phi(x_t, x_{t+1}))$最大化。为帮助行为制约式鉴别器，我们添加了一个正则项，以将a_t和$a_t^{pred} = \psi(x_{t+1}, x_t)$之间的交叉熵损失最小化。$\phi$和$\psi$都是MLP，除最后一层外，均与动作制约性鉴别器共享层。最后，我们发现通过在图像和特征空间中添加少量重建损失项，有助于稳定训练（对于特征空间，我们减小了生成帧与真实帧的单一图像鉴别器特征之间的差距）。详情请见补充材料。

3.4.2 周期损失函数

基于RNN的生成器能够追踪近期状态，以生成连贯的图像帧。但是，它很快就会忘记遥远的过去发生的事情，因为它只是被鼓励生成逼真的下一个观察结果。为确保静态元素的长期一致性，我们利用记忆模块和渲染引擎，将静态元素与动态元素分离。

经过一些时间步T之后，用动态引擎的输出信息填充于记忆块$\mathcal{M}$。使用记忆位置历史记录a_t，我们就能检索出记忆向量$\hat{m}_t$，如果位置a_t处的内容已被修改，则该向量可能与m_t不同。现在，$c = \{\hat{m}_t, 0\}$被传至渲染引擎

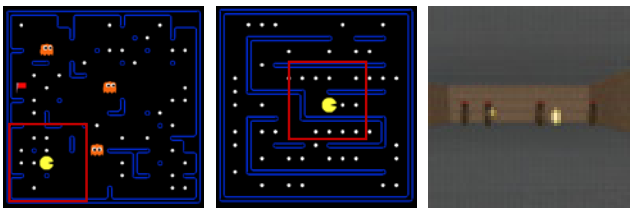


图7：本研究的数据集中的样本。对于吃豆人和吃豆人迷宫，训练数据包括部分观察到的状态，如红色框所示。左：吃豆人，中间：吃豆人迷宫，右：VizDoom

以生成$X^{\hat{m}_t}$，其中0是零向量，$X^{\hat{m}_t}$是与$\hat{m}_t$对应的输出组件。我们使用以下损失方程：

$$L_{cycle} = \sum_{t}^{T} ||X^{m_t} - X^{\hat{m}_t}|| \tag{10}$$

由于动态元素（例如《吃豆人》中的移动重影）不会随时间改变，因此建议引擎将静态元素放入记忆向量中，以减少L_{cycle}。由此可实现长期一致性。

为防止模型试图从简而忽略记忆组件，我们使用了一个正则项，其可将隐藏状态向量中精细蒙版min$\sum\eta^h$中所有位置的总和最小化，以使X^{mt}必须包含内容。另一个不太重要的解决方案是，如果学得的所有动作的移位内核都无法出现在对方的相反方向。在这种情况下，$\hat{m}_t$和m_t将始终相同，因为永远不会再访问相同的记忆位置。因此，我们以相对应的$\hat{a}$对动作a施加约束（例如上下），$\hat{a}$的移位内核$K(\hat{a})$等于水平和垂直翻转的$K(a)$。由于大多数需要长期一致性的模拟器都涉及导航任务，因此找到相对应的对象并非难事。

3.4.3 训练计划

GameGAN进行的是端到端训练。我们通过预热阶段，在前几次将真实帧输入动态引擎，然后将真实帧的数量缓慢减少到1（始终提供初始帧x_0）。我们分别使用18和32个图像帧，在《吃豆人》和VizDoom环境下训练GameGAN。

4. 实验

我们进行了定性和定量实验，主要采用了以下四个模型：1)动作-LSTM：该模型只基于重构损失函数训练，本质上与[6]类似，2)世界模型[13]，3)GameGAN-M：该模型没有记忆模块，但包含简单的渲染引擎，以及4)GameGAN：具有记忆模块和渲染引擎，可拆分动态和静态元素的模型。试验基于以下三个数据集开展（图7）：

吃豆人：我们使用了修改版的《吃豆人》游戏[2]，其中吃豆人代理从完整的14x14环境中观察到以自我为中心的7x7网格。环境是针对每一关随机生成环境。对于模拟器质量的测试，这可谓理想环境，因其既具有预定义的（例如，游戏规则和视点移动），又具有高度随机性的组成部分（例如，游戏中食物和墙壁的布局；带有重影的游戏动态）。闯关情节的图像为84x84，动作空间为$\mathcal{A} = \{left, right, up, down, stay\}$。提取长度大于或等于18的45K闯关情节，并将40K用于训练。训练数据是通过使用受过训练的DQN [30]代理生成的，该代理以高熵值观察整个环境，以探索多样化动作序列。 每一关都包含一系列以吃豆人为中心的7x7网格，以及动作。

吃豆人迷宫：该游戏与《吃豆人》类似，但没有鬼影，且游戏中的墙是通过迷宫生成算法来随机生成的，因此结构更完善。使用的数据量与《吃豆人》相同。

Vizdoom: 我们遵循Ha和Schmidhuber [13]的实验设置，使用VizDoom平台[20]的$takecover$模式。训练数据为通过随机策略提取的10K闯关情节，其中的图片为64x64，动作空间为$\mathcal{A} = \{left, right, stay\}$

4.1. 定性评估

图8显示了不同模型在《吃豆人》数据集上的表现。对于动作-LSTM，其训练伴有重建损失，由于无法捕获多模式未来分布，会产生模糊的图像，且误差会迅速累积。世界模型[13]为VizDoom生成了逼真的图像，对于具有高度随机性的《吃豆人》环境，模拟仍存在困难。特别是，它有时会遭受较大的意外中断（例如$t = 0$至$t = 1$）。另一方面，GameGAN能够生成时间上一致且逼真的清晰图像。GameGAN仅由几个卷积层组成，可大致匹配世界模型的参数量。我们还提供一版生成能够生成更高画质图像的GameGAN，详见补充材料第A.5节。

分解静态和动态元素： 我们的GameGAN与记忆模块经过训练，可将静态元素与动态元素分解开来。图5显示了如何将《吃豆人》环境中的墙壁和VizDoom环境中的房间与诸如鬼魂和火球之类的动态对象分解。这样，我们就能创建一个有趣的环境，其中每个元素都可以与其他对象交换。可将敌人放置在用户喜欢的地方，而非分布在VizDoom中丧气的房间里，也可以让马里奥在房间中四处奔跑（图9）。不必修改原始游

[2]http://ai.berkeley.edu/project_overview.html

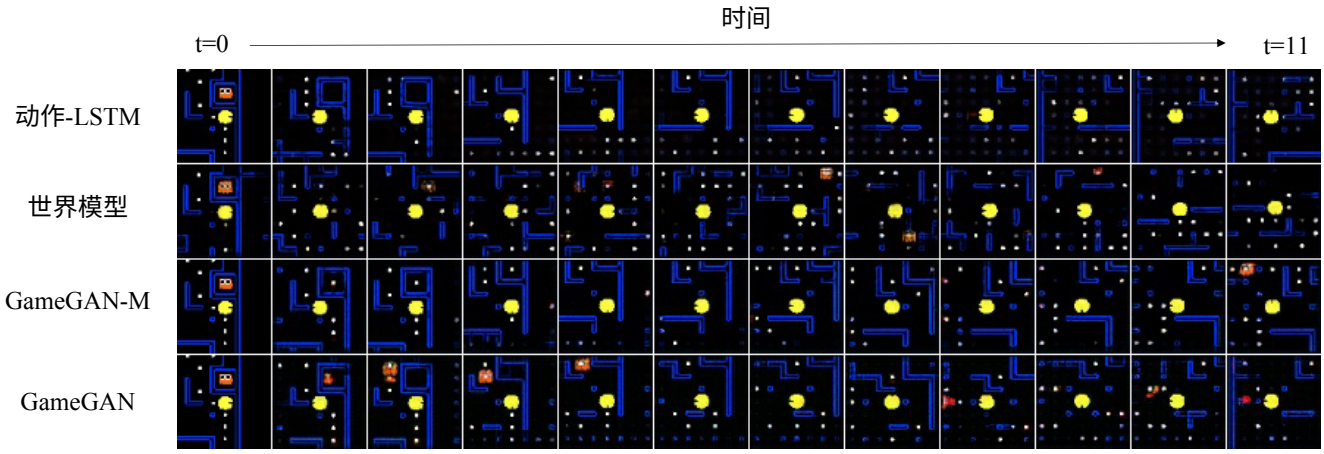


图8：在同一初始屏幕开始运行模型。训练时带有重建损失动作-LSTM生成的框架缺失细节（例如食物）。世界模型很难保持时间上的一致性，导致偶尔会出现明显的不连续性。GameGAN可产出连贯的模拟画面。

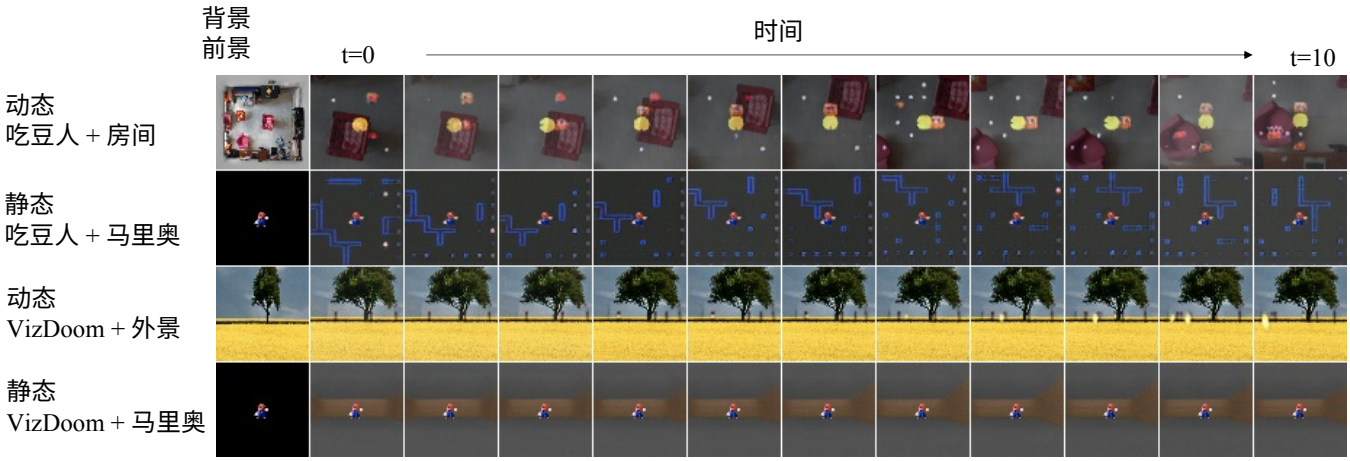


图9: GameGAN应用于《吃豆人》和VizDoom，背景/前景与随机图像交换

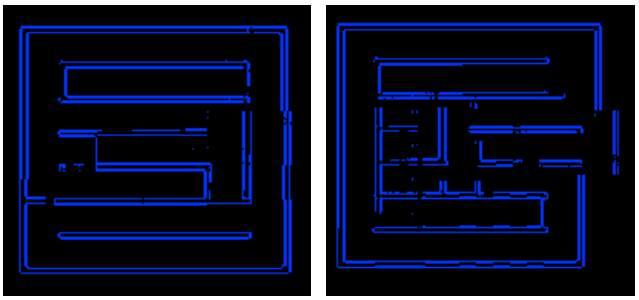


图10：在GameGAN模型上通过吃豆人代理遍历生成的迷宫。大多数迷宫都是现实可行的。右图显示了无法正确闭环的错误情况。

戏代码，就能够交换背景。我们的方法是将游戏视为黑匣子，并学习如何复制游戏，从而轻松对其进行修改。解耦的模型也为现有模型开启了未来的至多可能。

一个有趣的方向是学习多个解耦模型，并交换某些组件。当动态引擎习得环境规则，渲染引擎习得渲染图像时，只要简单地从一个模型的隐藏状态中习得线性变换，以利用另一个模型的渲染引擎就可以了。

吃豆人迷宫生成：针对吃豆人迷宫，GameGAN在每个时间步都会生成一个局部网格，可将其连接以生成完整的迷宫。它能够生成逼真的墙体，且由于环境足够小，因此GameGAN还能掌握映射的粗略尺寸，并在大多数情况下能够正确绘制矩形边界。图10的右下角显示了一个失败案例，该案例未能实现闭环。

4.2. 任务1: 训练RL代理

对环境质量的定量测量可谓一项挑战，因为未来是多模式的，绝无定式。一种测量方法是通过在模拟环境中训练强化学习代理，并在实际环境中对受过训练

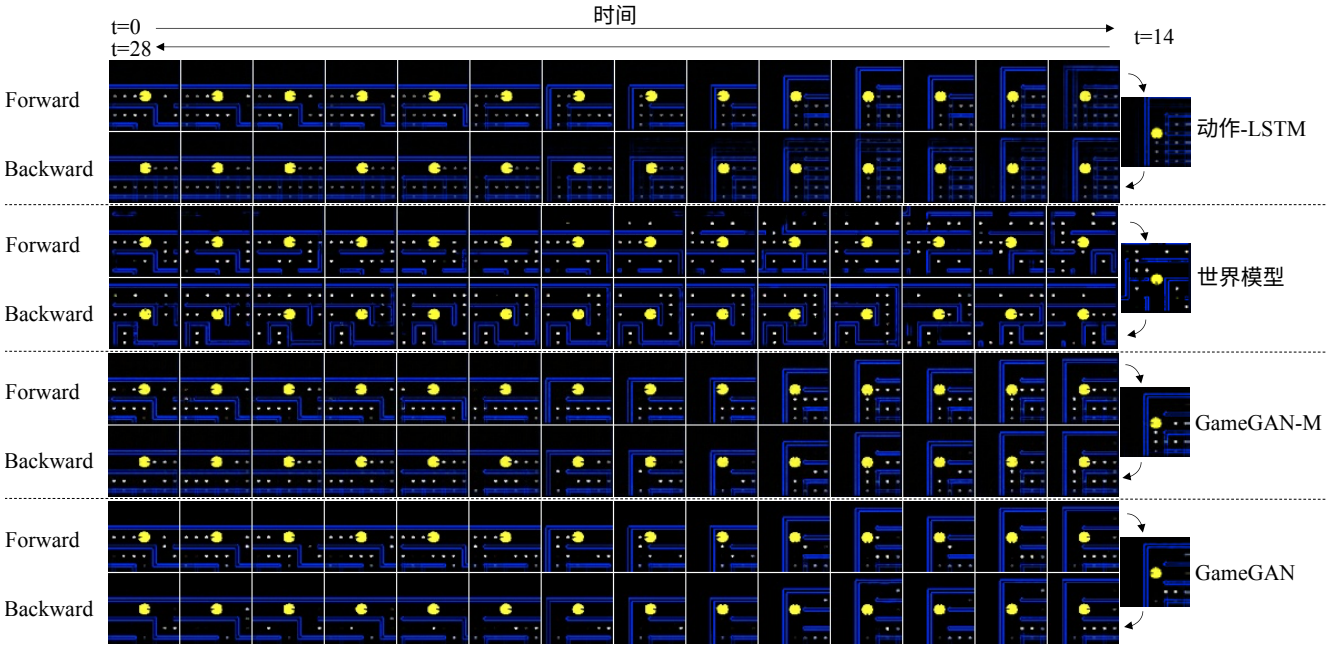


图11：“回家”任务部署。每组的Forward显示的是从初始位置到目标位置的路径。每组的Backward显示的是返回到初始位置的路径。 只有完整的GameGAN模型才能成功回到初始位置。

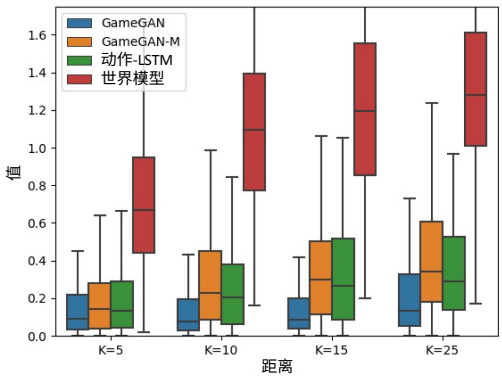


图12：“回家”任务指标的箱形图。越低越好。作为参照，同一关中一对随机选择的帧的得分为1.17 ± 0.56。

的代理进行测试。模拟环境应足够接近真实环境，才能在真实环境中表现良好。它必须习得真实环境中存在的动态元素、规则和随机性。更好的模拟器（与实际环境非常相似）中的代理得分也会更高。我们注意到，这与基于模型的RL密切相关。由于GameGAN内部没有表示游戏得分的机制，因此我们训练了外部分类器。向分类器提供N个先前的图像帧和当前的动作，以产生输出（例如Win / Lose）。

吃豆人：对于此任务，吃豆人代理必须通过吃食物（+0.5奖励）和夺取旗帜（+1奖励）来获得高分。被鬼魂吞噬或用到最大步数(40)时扣1个奖励。请注意，这是一项具有挑战性的部分观察的强化学习任务，在该任务中，代理会观察7x7网格。使用具有LSTM组件的A3C [29]对代理进行训练。

VizDoom：我们使用协方差矩阵自适应演化策略[14]来训练RL代理。我们采用与[13]相同的设置，使用了相应的模拟器。

	吃豆人	VizDoom
随机策略	-0.20 ± 0.78	210 ± 108
动作-LSTM[6]	-0.09 ± 0.87	280 ± 104
世界模型[13]	1.24 ± 1.82	1092 ± 556
GameGAN $-M$	1.99 ± 2.23	724 ± 468
GameGAN	1.13 ± 1.56	765 ± 482

表1：数字为平均分数±标准偏差，值越高越好。对于《吃豆人》游戏，在实际环境中训练的代理达到3.02±2.64，可认作上限。而对于VizDoom，分数达到750时则可认为问题得以解决。

表1显示了结果。对于所有实验，分数都是经过100个测试环境计算得出的，我们报告的是平均分数以及标准偏差。在动作-LSTM模拟器中训练的代理，其性能类似于具有随机策略的代理，这表明模拟场景与真实状况相差甚远。对于《吃豆人》，GameGAN-M表现最佳，而GameGAN和世界模型得分相似。对于VizDoom，

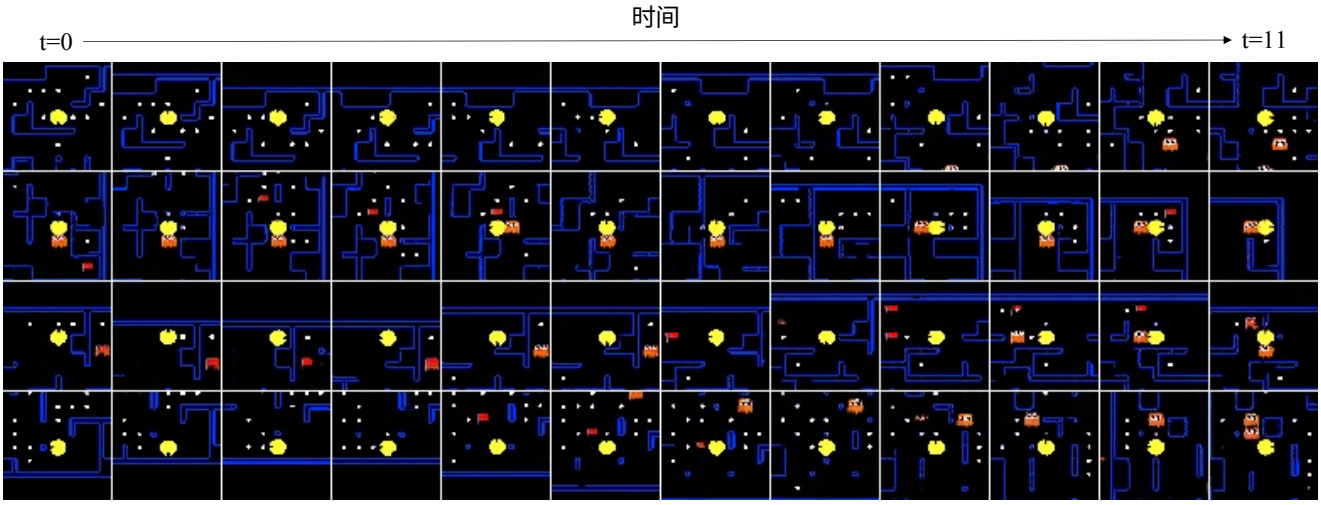


图13：第A.5节中描述的采用更高容量GameGAN的《吃豆人》展示。 画质明显优于图8所示的简单GameGAN模型。

对于VizDoom，分数达到750时则可认为问题闯关成功。而GameGAN则能够闯关成功。请注意，尽管世界模型的得分更高，但是GameGAN是首个基于GAN框架进行训练，并成功闯关的研究工作。而且，GameGAN可以进行端到端的训练，不像世界模型是通过连续训练以提高稳定性。有趣的一点发现是，在吃豆人环境下，GameGAN的性能低于GameGAN-M。这是由于在训练模型时需要额外的复杂性，其中环境无需长期一致性就可获得更高分数。我们发现，在训练记忆模块的同时对GAN目标进行优化的难度更大，这归因于RL代理会利用环境的缺陷来伺机作弊。此种情况下，我们发现GameGAN有时无法阻止代理穿墙，而GameGAN-M却近乎完美。这就使得RL代理发现了一种喜欢撞墙的政策，在实际环境中，这通常会导致游戏过早结束。在下一部分中，我们将展示特性情境下拥有长期一致性的好处。

4.3. 任务2：回家

该任务评估了吃豆人迷宫环境中模拟器的长期一致性。吃豆人从状态为s的随机初始位置$(x_A,\ y_A)$开始。给予K个随机动作$(a_1,\ ...,\ a_K)$，最后到达位置$(x_B,\ y_B)$。使用反向动作$(\hat{a}_K,\ ...,\ \hat{a}_1)$(例如 $a_k\ =\ Down, \hat{a}_K\ =\ Up$)，它会返回到初始位置$(x_A,\ y_A)$，处于状态$\hat{s}$。现在，我们可以测量 $\hat{s}$ 和s之间的距离d，对长期一致性进行评估（对于真实环境，$d\ =\ 0$）。由于除了墙壁以外的其他元素（例如食物）可能会发生变化，因此我们仅对$\hat{s}$和s的墙壁进行比较。因此，s是一个84x84的二进制图像，如果像素为蓝色，则其像素为1。我们将指标d定义为

$$d = \frac{\text{sum}(\text{abs}(s - \hat{s}))}{\text{sum}(s) + 1} \tag{11}$$

其中sum()为1的计数。因此，d表示变化的像素数与初始像素数之比。图12显示了结果。我们再一次在世界模型中观察到了偶尔出现的较大不连续性，这会严重影响性能。当K较小时，性能差异相对较小。这是因为其他模型也具有通过RNN实现的短期一致性。但是，随着K的增大，带有记忆模块的GameGAN稳步胜过其他模型，且差距也越来越大，这表明GameGAN能够有效利用记忆模块。图11显示了吃豆人迷宫环境中不同模型的部署。可以看出，无记忆模块的模型不记得之前生成的内容。这表明GameGAN不仅为游戏模拟器开启广阔的发展方向，还为能够模拟现实世界的通用环境模拟器开启了广阔的前景。

5. 结论

我们提出了GameGAN模型，该模型利用对抗训练来习得如何模拟游戏场景。GameGAN仅通过观察游戏规则及用户动作来训练GameGAN，且无需访问游戏的逻辑或引擎。GameGAN包含一个新的记忆模块，可确保长期一致性，且经过训练，可分解静态和动态元素。通过透彻的定量研究，GameGAN的建模能力得以证明。在未来的工作中，我们会对模型进行扩展，以捕获更复杂的现实环境。

致谢

感谢Bandai-Namco Entertainment Inc.提供了官方版《吃豆人》以用于训练。还要感谢Ming-Yu Liu和Shao-Hua Sun，与二位的探讨让我们获益良多。

参考文献

[1] Anonymous. Model based reinforcement learning for atari. In *Submitted to International Conference on Learning Representations*, 2020. under review. 3

[2] Randall D Beer and John C Gallagher. Evolving dynamical neural networks for adaptive behavior. *Adaptive behavior*' 1(1):91–122, 1992. 1

[3] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018. 15

[4] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krhenbhl. Learning by cheating. In *Conference on Robot Learning (CoRL)*, pages 6059–6066, 2019. 1

[5] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016. 5, 15

[6] Silvia Chiappa, Sébastien Racaniere, Daan Wierstra, and Shakir Mohamed. Recurrent environment simulators. *arXiv preprint arXiv:1704.02254*, 2017. 2, 3, 6, 8

[7] Aidan Clark, Jeff Donahue, and Karen Simonyan. Efficient video generation on complex datasets. *arXiv preprint arXiv:1907.06571*, 2019. 2

[8] Emily Denton and Rob Fergus. Stochastic video generation with a learned prior. *arXiv preprint arXiv:1802.07687*, 2018. 2

[9] Emily L Denton and vighnesh Birodkar. Unsupervised learning of disentangled representations from video. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4414–4423. Curran Associates, Inc., 2017. 2

[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 2, 15

[11] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014. 4, 13

[12] Matthew Guzdial, Boyang Li, and Mark O Riedl. Game engine learning from video. In *IJCAI*, pages 3707–3713, 2017. 3

[13] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*, pages 2450–2462, 2018. 2, 3, 6, 8

[14] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001. 8

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997. 3

[16] Minyoung Huh, Shao-Hua Sun, and Ning Zhang. Feedback adversarial learning: Spatial feedback for improving generative adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019. 5

[17] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 15

[18] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016. 2

[19] Ajay Jain, Sergio Casas, Renjie Liao, Yuwen Xiong, Song Feng, Sean Segal, and Raquel Urtasun. Discrete residual flow for probabilistic pedestrian behavior prediction. *arXiv preprint arXiv:1910.08041*, 2019. 1

[20] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2016. 2, 6

[21] Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, and Jiwon Kim. Learning to discover cross-domain relations with generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1857–1865. JMLR. org, 2017. 2

[22] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 16

[23] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. *arXiv preprint arXiv:1512.09300*, 2015. 3

[24] Alex X. Lee, Richard Zhang, Frederik Ebert, Pieter Abbeel, Chelsea Finn, and Sergey Levine. Stochastic adversarial video prediction. *CoRR*, abs/1804.01523, 2018. 2

[25] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised image-to-image translation networks. *CoRR*, abs/1703.00848, 2017. 2

[26] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? *arXiv preprint arXiv:1801.04406*, 2018. 16

[27] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014. 5, 15

[28] Takeru Miyato and Masanori Koyama. cGANs with projection discriminator. In *International Conference on Learning Representations*, 2018. 2

[29] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016. 8

[30] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 6

[31] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2863–2871. Curran Associates, Inc., 2015. 2

[32] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2337–2346, 2019. 2, 5, 14, 15

[33] Chris Paxton, Vasumathi Raman, Gregory D Hager, and Marin Kobilarov. Combining neural networks and tree search for task and motion planning in challenging environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6059–6066. IEEE, 2017. 1

[34] Scott E. Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 1060–1069, 2016. 2

[35] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016. 5, 15

[36] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using lstms. In *International conference on machine learning*, pages 843–852, 2015. 2

[37] Sergey Tulyakov, Ming-Yu Liu, Xiaodong Yang, and Jan Kautz. Mocogan: Decomposing motion and content for video generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1526–1535, 2018. 2

[38] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016. 15

[39] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. Video-to-video synthesis. *CoRR*, abs/1808.06601, 2018. 2

[40] Dingdong Yang, Seunghoon Hong, Yunseok Jang, Tianchen Zhao, and Honglak Lee. Diversity-sensitive conditional generative adversarial networks. *arXiv preprint arXiv:1901.09024*, 2019. 5, 15

[41] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017. 2

补充材料

我们提供了有关模型架构（A部分），训练计划（B部分）和附图（C部分）的详细说明。

A. 模型架构

以下是第3节中描述的每个模块的架构详情。我们采用以下表示法来描述模块：

Conv2D(a,b,c,d): 2D卷积层，其输出通道大小为a，卷积核大小为**b**，stride为**c**，padding大小为**d**。

Conv3D(a,b,c,d,e,f,g): 3D卷积层，其输出通道大小为**a**，时间核大小为**b**，空间核大小为**c**，时间stride为**d**，空间stride为**e**，时间padding大小为**f**，空间padding大小为**g**。

T.Conv2D(a,b,c,d,e): 转置的2D卷积层，其输出通道大小为**a**，卷积核大小为**b**，stride为**c**，padding大小为**d**，输出padding大小为**e**。

Linear(a): 线性层，其输出大小为**a**。

Reshape(a): 重塑输入张量，使其输出大小为**a**。

LeakyReLU(a): 带泄露的修正线性单元(Leaky ReLU)函数，斜率为**a**.

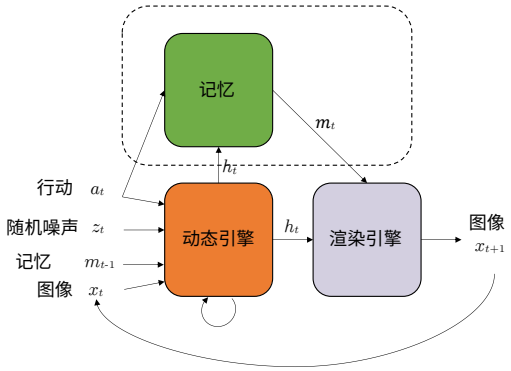


图14: GameGAN概览：a_t和z_t输入动态引擎在时间t输入x_t，更新隐藏状态。它可以选择性地写入和读取外部记忆模块（虚线框中）。最后，渲染引擎用于对输出图像x_{t+1}进行解码。整个系统的训练都是端到端的。 所有模块均为神经网络。

A.1. 动态引擎

输入动作$a \sim \mathcal{A}$是单点编码的向量。对于《吃豆人》的环境，我们定义$\mathcal{A} = \{left, right, up, down, stay\}$，其对应动作为$\hat{a}$（参见第3.4.2节），我们定义$\hat{left} = right$，$\hat{up} = down$，反之亦然。图像$x$的大小为84×84×3。对于VizDoom，$\mathcal{A} = \{left, right, stay\}$，$\hat{left} = right$，$\hat{right} = left$。图像$x$的大小为64×64×3。

在每个时间步t，从标准正态分布$\mathcal{N}(0; I)$中采样一个32维随机变量z_t。基于历史图像$x_{1:t}$以及a_t和z_t，GameGAN就能预测下一个图像x_{t+1}。

为完整起见，我们在此处重述动态引擎的计算顺序（第3.1节）。

$$v_t = h_{t-1} \odot \mathcal{H}(a_t, z_t, m_{t-1}) \tag{12}$$

$$s_t = \mathcal{C}(x_t) \tag{13}$$

$$i_t = \sigma(W^{iv} v_t + W^{is} s_t), f_t = \sigma(W^{fv} v_t + W^{fs} s_t), \\ o_t = \sigma(W^{ov} v_t + W^{os} s_t) \tag{14}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W^{cv} v_t + W^{cs} s_t) \tag{15}$$

$$h_t = o_t \odot \tanh(c_t) \tag{16}$$

其中h_t、a_t、z_t、c_t、x_t是时间步t时的隐藏状态、动作、随机变量、单元状态和图像。⊙表示哈达玛积。

动态引擎的隐藏状态是512维向量。因此，$h_t, o_t, c_t, f_t, i_t, o_t \in \mathbb{R}^{512}$。

$\mathcal{H}$首先会为每项输入计算嵌入，然后通过两层MLP[Linear(512), LeakyReLU(0.2), Linear(512)]级联并通过。如果隐藏单元的大小不同于512，则h_{t-1}也可以在等式13中的哈达玛积之前穿过线性层。

$\mathcal{C}$被实现为5层（对于64 x 64的图像）或6层（对于84 x 84的图像）的卷积网络，其后为线性层：

《吃豆人》	VizDoom
Conv2D(64, 3, 2, 0)	Conv2D(64, 4, 1, 1)
LeakyReLU(0.2)	LeakyReLU(0.2)
Conv2D(64, 3, 1, 0)	Conv2D(64, 3, 2, 0)
LeakyReLU(0.2)	LeakyReLU(0.2)
Conv2D(64, 3, 2, 0)	Conv2D(64,3, 2, 0)
LeakyReLU(0.2)	LeakyReLU(0.2)
Conv2D(64, 3, 1, 0)	Conv2D(64, 3, 2, 0)
LeakyReLU(0.2)	LeakyReLU(0.2)
Conv2D(64, 3, 2, 0)	Reshape(7*7*64)
LeakyReLU(0.2)	Linear(512)
Reshape(8*8*64)	
Linear(512)	

A.2. 记忆模块

为完整起见，我们在此处重述记忆模块的计算顺序（第3.2节）。

$$w = \text{softmax}(\mathcal{K}(a_t)) \in \mathbb{R}^{3\times 3} \tag{17}$$

$$g = \mathcal{G}(h_t) \in \mathbb{R} \tag{18}$$

$$\alpha_t = g \cdot \text{Conv2D}(\alpha_{t-1}, w) + (1-g) \cdot \alpha_{t-1} \tag{19}$$

$$\mathcal{M} = \text{write}(\alpha_t, \mathcal{E}(h_t), \mathcal{M}) \tag{20}$$

$$m_t = \text{read}(\alpha_t, \mathcal{M}) \tag{21}$$

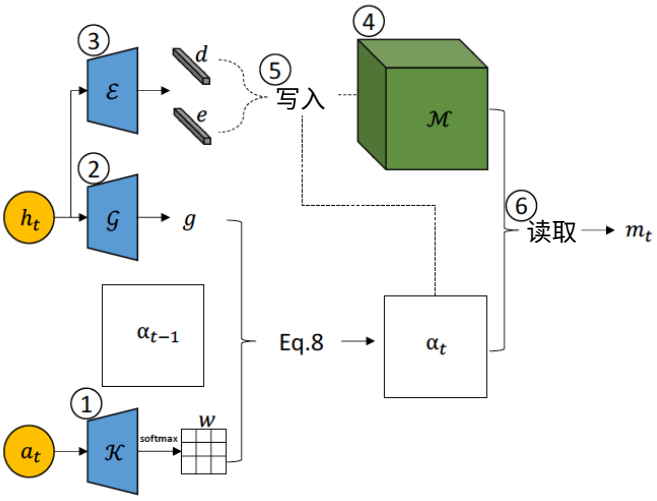


图15: 记忆模块。圆圈数字编号对应的说明见下文

① $\mathcal{K}$为双层MLP，可输出9维向量，该向量经过softmax和reshape，成为大小为3x3的核w：[Linear(512), LeakyReLU(0.2), Linear(9), Softmax(), Re-shape(3,3)].

② $\mathcal{G}$为双层MLP，可输出一个标量值，后跟Sigmoid激活函数，使得$g \in [0,1]$: [Linear(512), LeakyReLU (0.2), Linear(1), Sigmoid()].

③ ε为单层MLP，可生成擦除向量$e \in \mathbb{R}^{512}$和加法向量$d \mathbb{R}^{512}$：[Linear(1024), split(512)]，其中Split(512)将1024维向量分为两个512维向量。e还将执行Sigmoid激活函数。

④ 用512维随机噪声$N(0, I)$，将记忆块$\mathcal{M}$中的每个空间位置初始化。为提高计算效率，我们训练中使用的块的宽度和高度$N = 39$，下游任务使用$N = 99$。因此，我们使用39×39×512个块进行训练，并使用99×99×512个块进行实验。注意，基于移位的记忆模块架构允许在测试时使用任意大小的块。

⑤ 写入操作与神经图灵机[11]相似。对于每个位置$\mathcal{M}^i$，写入计算：

$$\mathcal{M}^i = \mathcal{M}^i(1 - \alpha_t^i \cdot e) + \alpha_t^i \cdot d \tag{22}$$

其中i表示块$\mathcal{M}$的空间x, y坐标。e是S型向量，当$e = 1$时，会擦除$\mathcal{M}^i$处的信息，且d会将新信息写入$\mathcal{M}^i$。注意，如果标量α_t^i为0（即位置未被关注），则记忆内容$\mathcal{M}^i$不变。

⑥ 读取操作定义为:

$$m_t = \sum_{i=0}^{N \times N} \alpha_t^i \cdot \mathcal{M}^i \tag{23}$$

其中$\mathcal{M}^i$表示位置i的记忆内容。

A.3. 渲染引擎

对于GameGAN-M的简单渲染引擎，我们首先将隐藏状态h_t传递至线性层，使其成为7×7×512张量，然后将其传递给5个转置卷积层，以生成3通道输出图像x_{t+1}:

《吃豆人》	VizDoom
Linear(512*7*7)	Linear(512*7*7)
LeakyReLU(0.2)	LeakyReLU(0.2)
Reshape(7, 7, 512)	Reshape(7, 7, 512)
T.Conv2D(512, 3, 1, 0, 0)	T.Conv2D(512, 4, 1, 0, 0)
LeakyReLU(0.2)	LeakyReLU(0.2)
T.Conv2D(256, 3, 2, 0, 1)	T.Conv2D(256, 4, 1, 0, 0)
LeakyReLU(0.2)	LeakyReLU(0.2)
T.Conv2D(128, 4, 2, 0, 0)	T.Conv2D(128, 5, 2, 0, 0)
LeakyReLU(0.2)	LeakyReLU(0.2)
T.Conv2D(64, 4, 2, 0, 0)	T.Conv2D(64, 5, 2, 0, 0)
LeakyReLU(0.2)	LeakyReLU(0.2)
T.Conv2D(3, 3, 1, 0, 0)	T.Conv2D(3, 4, 1, 0, 0)

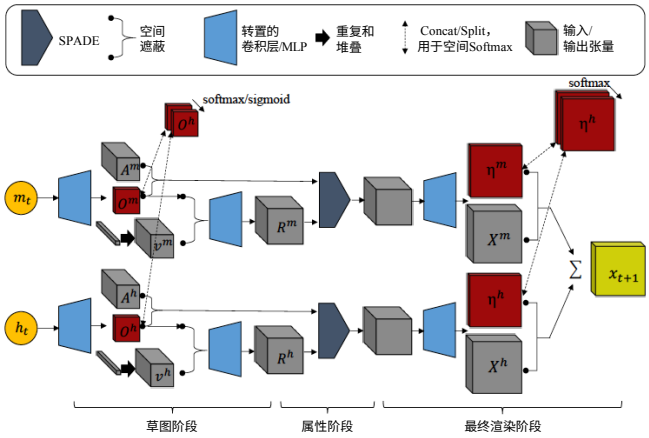


图16: 用于分解静态和动态组件的渲染引擎。圆圈数字编号对应的说明见下文。

对于以$c = \{c^1, ..., c^K\} = \{m_h, h_t\}$向量列表作为输入的专用解耦渲染引擎，实现了以下内容。具体圆圈数字编号见图16。

① 首先，通过将c^k传递到线性层，使其成为3×3×128张量，获得$A^k \in \mathbb{R}^{H_1 \times H_1 \times D_1}$ and $O^k \in {}^{RH_1 \times H_1 \times 1}$，并将该张量传递至两个转置卷积层，将过滤器大小设置为3，以生成R7×7×32+1张量（因此，$H_1 = 7, D_1 = 32$）：

《吃豆人》&Vizdoom
Linear(3*3*128)
Reshape(3,3,128)
LeakyReLU(0.2)
T.Conv2D(512, 3, 1, 0, 0)
LeakyReLU(0.2)
T.Conv2D(32+1, 3, 1, 0, 0)

我们将所得的张量在通道上分离，得到A^k和O^k。v^k是通过运行c^k，使c^k穿过一层MLP [Linear(32)，LeakyReLU(0.2)]，并将其沿空间维度堆叠以匹配A^k的大小而获得的。

② 粗略的草图张量R^k是通过将由O^k遮蔽的v^k（经过空间softmax或S形函数）穿过两个转置卷积层而获得的：

《吃豆人》	VizDoom
T.Conv2D(256, 3, 1, 0, 0)	T.Conv2D(256, 3, 1, 0, 0)
LeakyReLU(0.2)	LeakyReLU(0.2)
T.Conv2D(128, 3, 2, 0, 1)	T.Conv2D(128, 3, 2, 1, 0)

③ 我们采用SPADE[32]的结构，以实例归一化作为归一化模块。由O^k遮蔽的属性映射A^k用于生成SPADE层参数γ和β的语义映射。

④ 归一化张量通过两层转置卷积层：

Pacman	VizDoom
T.Conv2D(64, 4, 2, 0, 0)	T.Conv2D(64, 3, 2, 1, 0)
LeakyReLU(0.2)	LeakyReLU(0.2)
T.Conv2D(32, 4, 2, 0, 0)	T.Conv2D(32, 3, 2, 1, 0)
LeakyReLU(0.2)	LeakyReLU(0.2)

通过上文中的输出张量，可通过单一卷积层[Conv2D(1, 3, 1)]获得η^k，然后将其与其他组件合并，以实现空间softmax。我们还尝试了级联η，并将其传递到softmax之前的1×1卷积层中，但未见太大区别。类似地，X^k是通过单一卷积层 [Conv2D(3, 3, 1)]获得的。

A.4. 鉴别器

GameGAN中使用了几种鉴别器。为提高计算效率，我们首先使用共享编码器，对每帧进行编码：

《吃豆人》	VizDoom
Conv2D(16, 5, 2, 0)	Conv2D(64, 4, 2, 0)
BatchNorm2D(16)	BatchNorm2D(64)
LeakyReLU(0.2)	LeakyReLU(0.2)
Conv2D(32, 5, 2, 0)	Conv2D(128, 3, 2, 0)
BatchNorm2D(32)	BatchNorm2D(128)
LeakyReLU(0.2)	LeakyReLU(0.2)
Conv2D(64, 3, 2, 0)	Conv2D(256, 3, 2, 0)
BatchNorm2D(64)	BatchNorm2D(256)
LeakyReLU(0.2)	LeakyReLU(0.2)
Conv2D(64, 3, 2, 0)	Conv2D(256, 3, 2, 0)
BatchNorm2D(64)	BatchNorm2D(256)
LeakyReLU(0.2)	LeakyReLU(0.2)
Reshape(3, 3, 64)	Reshape(3, 3, 256)

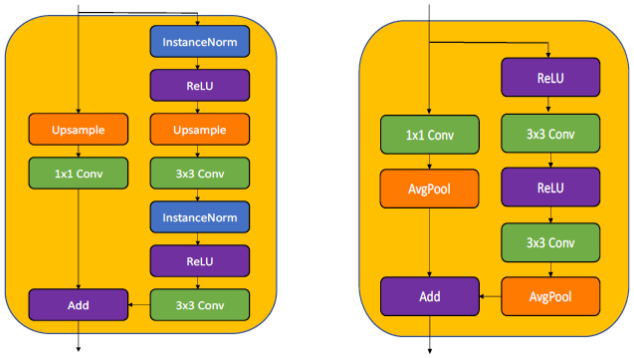
单帧鉴别器：单帧鉴别器的工作是判断给定的单帧是否为真实数据。对于基于补丁的(D_{patch})和全帧(D_{full})鉴别器，我们采用了两个简单的网络：

D_{patch}	D_{full}
Conv2D(dim, 2, 1, 1)	Conv2D(dim, 2, 1, 0)
BatchNorm2D(dim)	BatchNorm2D(dim)
LeakyReLU(0.2)	LeakyReLU(0.2)
Conv2D(1, 1, 2, 1)	Conv2D(1, 1, 2, 1)

其中《吃豆人》的dim为64，VizDoom的dim为256。D_{patch}给出3×3的对数，而D_{full}给出单一对数。

动作制约性鉴别器。我们为三对动作制约性鉴别器D_{action}赋予：D_{action}: (x_t, x_{t+1}, a_t)、$(x_t, x_{t+1}, \bar{a}_t)$ 和 $(\hat{x}_t, \hat{x}_{t+1}, a_t)$。$x_t$表示真实图像 $\hat{a}$ 表示生成的图像，且 $\bar{a}_t \in \mathcal{A}$，这是一个采样的负动作，使 $\bar{a}_t \neq a_t$。鉴别器的工作是判断两个帧相对于给定动作是否一致。首先，我们通过嵌入层[Linear(dim)]获得独热编码动作的嵌入向量，其中《吃豆人》的dim=32，VizDoom的dim=128。然后，将xt, xt+1两个帧按通道进行级联，并与一个卷积层[Conv2D(dim, 3, 1, 0), BatchNorm2D(dim), LeakyReLU(0.2), Reshape(dim)]合并。最后，将动作嵌入和合并帧特征级联，并馈入[Linear(dim), BatchNorm1D (dim), LeakyReLU(0.2), Linear(1)]，从而生成单一对数。

时间鉴别器。时间鉴别器$D_{temporal}$将几帧作为输入，即可判定其为真实序列还是生成的序列。我们实施了一个分层时间鉴别器，可在多个层级输出对数。第一层将时间范围内的所有帧级联，并执行以下操作：



(a) (b)

图17: 基于[3]的图15修改并重画的高容量模型的残差块。(a)渲染引擎块，(b)鉴别器块。

Conv3D(64, 2, 2, 1, 1, 0, 0)
BatchNorm3D(64)
LeakyReLU(0.2)
Conv3D(128, 2, 2, 1, 1, 0, 0)
BatchNorm3D(128)
LeakyReLU(0.2)

上文中的输出张量被馈送到两个分支中。 第一个是单层[Conv3D(1, 2, 1, 2, 1, 0, 0)]，它可产生一个对数。因此，该对数能够有效查看6帧，并判断其是否真实。第二个分支使用张量作为下一级的输入：

Conv3D(256, 3, 1, 2, 1, 0, 0)
BatchNorm3D(256)
LeakyReLU(0.2)

现在，类似地，输出张量被馈送到两个分支中。第一个是单层[Conv3D(1, 2, 1, 1, 1, 0, 0)]，可产生一个能够有效查看18帧的对数。 第二个分支使用张量作为下一级的输入：

Conv3D(512, 3, 1, 2, 1, 0, 0)
BatchNorm3D(512)
LeakyReLU(0.2)
Conv3D(1, 3, 1, 1, 1, 0, 0)

从而得到一个具有32帧时间接收场大小的对数。

《吃豆人》环境使用两个级别（最多18帧），而VizDoom使用所有三个级别。

A.5 提升容量

A.3和A.4中描述的渲染引擎和鉴别器都只包含几个线性层和卷积层，这会限制GameGAN的表现力。基于图像生成GAN的最新进展[3, 32]，我们尝试了具有更高容量的残差块。在此，我们将重点介绍与前几节相比的主要区别。我们会发布代码，以提高研究的复现性。

渲染引擎：渲染引擎中的卷积层被图17a中的残差块替代。残差块遵循[3]的设计，不同之处在于，批量归一化层[17]被实例归一化层[38]代替。对于专用渲染引擎，仅具有两种对象类型（如A.3中所述）也可能是一个限制因素。通过将向量添加到向量列表中，可以轻松添加更多组件（例如，使c = $\{m_h, h_t^1, h_t^2, ...h_t^n\}$，其中$h_t = concat\ (h_t^2, ...h_t^n)$，因该架构对任意数量的组件均通用。但是，这将导致解码器数量为$length(\mathrm{c})$，我们通过让$v^k = MLP\ (h_t) \in \mathbb{R}^{H_1 \times H_1 \times 32}$，而非在空间维度上堆叠$v^k$，可放宽对动态元素的约束。

鉴别器：如果因鉴别器容量不足以至于很容易被愚弄，那么仅增加渲染引擎的容量，是无法生成更高画质的图像的。我们用图17b所示的一堆残差块代替了鉴别器的共享编码器。通过新编码器馈送的每一帧都会产生$N \times N \times D$个张量，其中VizDoom的$N = 4$，《吃豆人》的$N = 5$。单帧鉴别器实施为[Sum(N,N), Linear(1)]，其中Sum(N,N)在张量的$N \times N$空间维度上求和。动作制约性和时间鉴别器使用的架构与A.4中的类似。

图13显示的是基于更高容量GameGAN模型训练的《吃豆人》上的展示。它清楚地表明，更高容量的模型可产生更逼真的锐利图像。因此，它更适合于未来需要对现实环境进行高保真模拟的相关工作。

B. 训练计划

我们采用标准GAN公式[10]，该公式可在生成器G（在我们的GameGAN示例中）和鉴别器D之间进行最小-最大博弈。令$\mathcal{L}_{\mathrm{GAN}}$为所有GAN损失函数的总和（对于单帧，我们使用相等的权重、行动制约性和时间损失函数）。由于已知条件GAN结构[27]能够学习简单的分布而忽略隐码[40, 35]，因此我们添加了信息正则化[5] $\mathcal{L}_{\mathrm{Info}}$，将隐码$z_t$和对$(x_t, x_{t+1})$之间的互信息$I(z_t, \phi(x_t, x_{t+1}))$最大化。为帮助以动作制约性鉴别器，我们添加了一个术语，其能够将a_t和$a_t^{pred} = \psi(x_{t+1}, x_t)$之间的交叉熵损失$\mathcal{L}_{\mathrm{Action}}$最小化。$\phi$和$\psi$都是MLP，与动作制约性鉴别器共享各层，除最后一层。最后，我们发现在图像（$\mathcal{L}_{\mathrm{recon}} = \frac{1}{T}\sum_{t=0}^{T} ||x_t - \hat{x}_t||_2^2$）和特征空间（$\mathcal{L}_{\mathrm{feat}} = \frac{1}{T}\sum_{t=0}^{T} ||feat_t - \hat{feat}_t||_2^2$）中添加较小的L2重建损失，有助于稳定训练，$x$和$\hat{x}$分别为真实和生成的图像，$feat$和$\hat{feat}$分别为通过共享编码器获得的真实和生成的特征鉴别器。

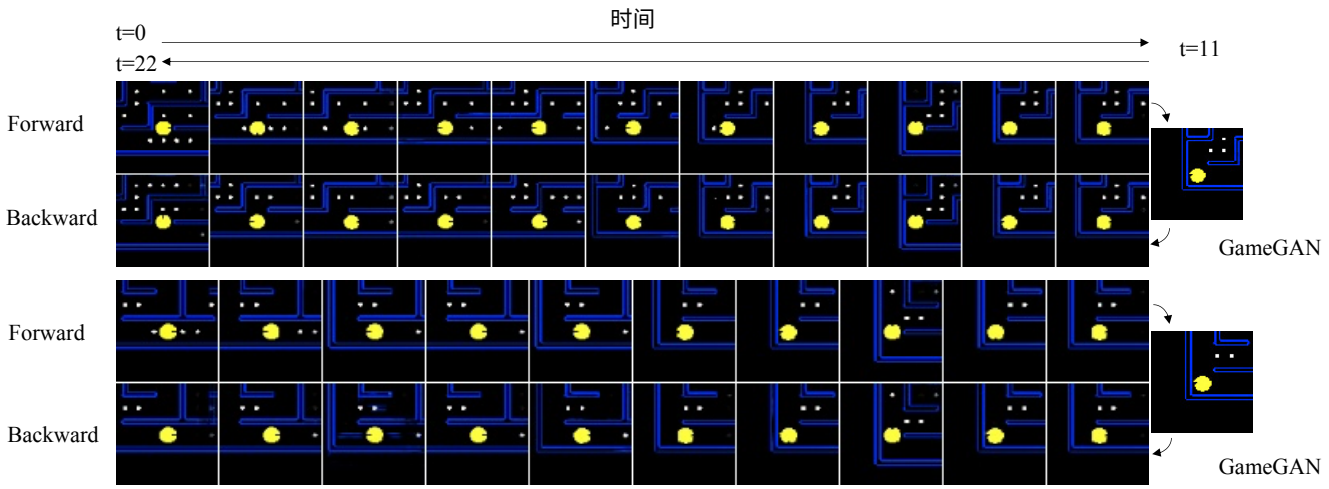


图18：其他“回家”任务的运行结果。 上一行显示的是从初始位置到目标位置的前进路径。 下一行显示的是从目标位置到初始位置的返回路径。

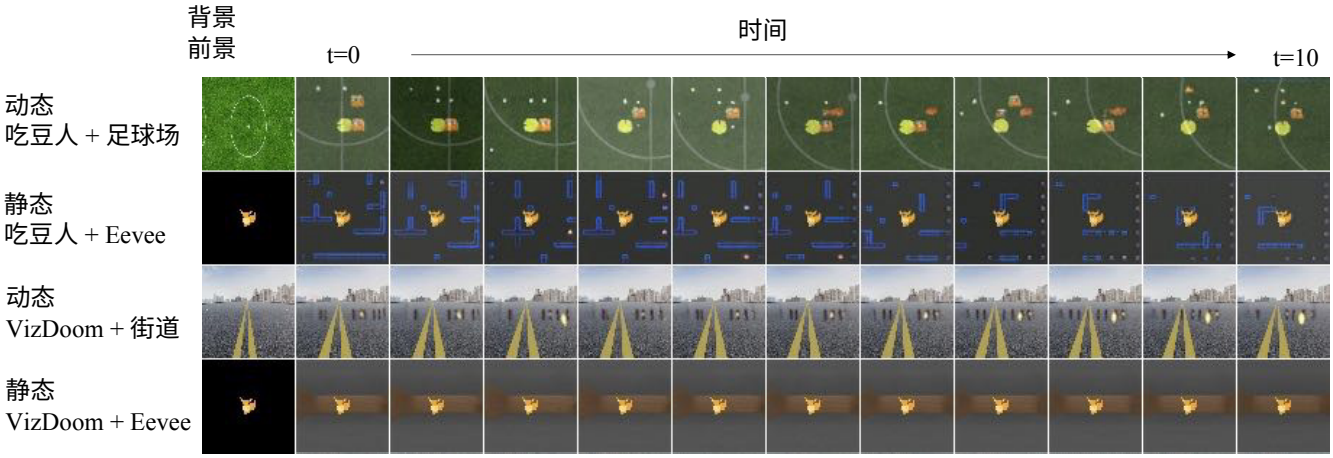


图19: 基于GameGAN的额外实验，将背景/前景与随机图像交换。

GameGAN能够优化:

$$\mathcal{L} = \mathcal{L}_{\text{GAN}} + \lambda_{\text{A}}\mathcal{L}_{\text{Action}} + \lambda_{\text{I}}\mathcal{L}_{\text{Info}} + \lambda_{\text{r}}\mathcal{L}_{\text{recon}} + \lambda_{\text{f}}\mathcal{L}_{\text{feat}} \tag{24}$$

当使用记忆模块和专用渲染引擎时，将$\lambda_{\text{c}}\mathcal{L}_{\text{cycle}}$（第3.4.2节）代入$\mathcal{L}$中，且$\lambda_{\text{c}} = 0.05$。我们不使用$\mathcal{L}_{\text{cycle}}$中的渐变更新渲染引擎，因为使用$\mathcal{L}_{\text{cycle}}$的目的是助力训练动态引擎和记忆模块，以实现长期一致性。我们设置$\lambda_{\text{A}} = \lambda_{\text{I}} = 1$和$\lambda_{\text{r}} = \lambda_{\text{f}} = 0.05$。在GameGAN的每个优化步骤之后，鉴别器即会更新。在对鉴别器进行训练时，我们添加了一个术语γL_{GP}，对于$\gamma = 10$的真实数据分布[26]，γL_{GP}是判别器梯度的罚值。

我们使用Adam优化器[22]，针对GameGAN和鉴别器的学习率为0.0001，批量(batch)大小为12。《吃豆人》和VizDoom环境中的GameGAN分别以18和32帧的序列进行训练。我们采用了一个预热阶段，将9（对于Pacman）/ 16（对于VizDoom）个实际帧馈入动态引擎，然后在第20个回合，将实际帧数线性减少，直至1。（初始帧始终为x_0）。