World Models（精简版 ~30 min）

在梦境中学习：基于世界模型的强化学习

Ha & Schmidhuber, 2018

本次分享版本说明

- **面向对象**：有基本深度学习/RL 背景的工程师/研究者
- **目标**：在 30 分钟内搞清楚：
  i. World Models 想解决什么问题?
  ii. V-M-C 架构到底在做什么?
  iii. **核心数学**：VAE (ELBO + 重参数化) 和 MDN (混合高斯)
  iv. 训练流程和实验结论是什么?
  v. 为什么后来需要 Dreamer?
- **与完整版的区别**：省略代码实现细节和 Workshop 互动环节

目录（精简版）

一分钟 Feynman 版解释

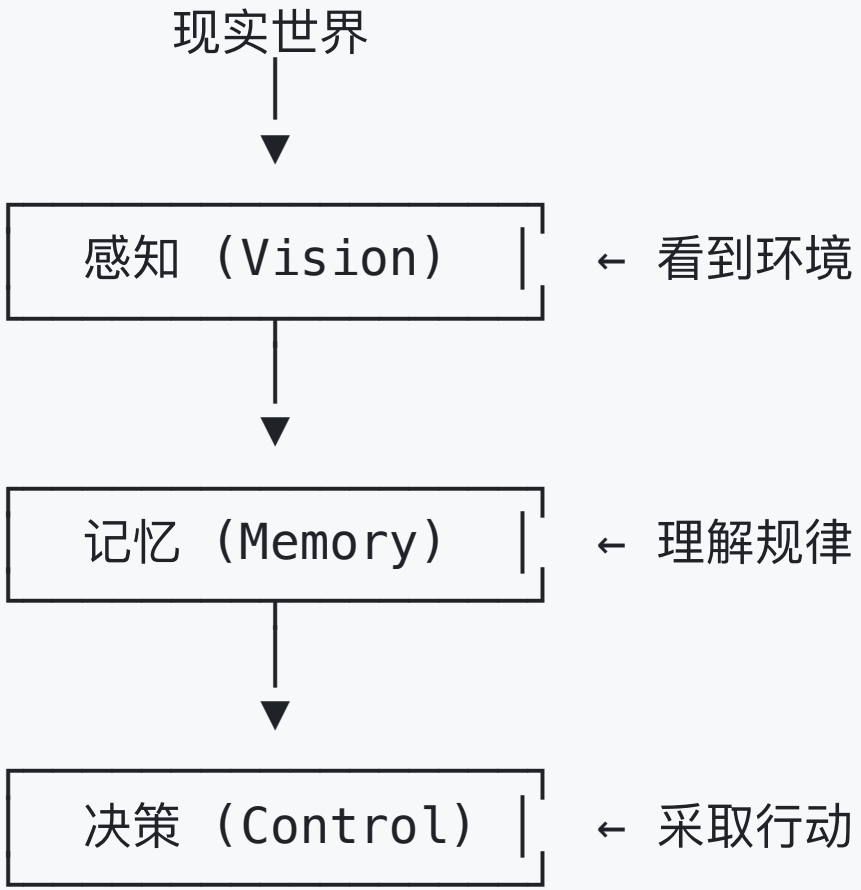
> 假设你要向一个**刚学完高一数学**的同学解释 World Models，可以这样说：

1. 我们先学会一个"压缩并预测"游戏世界的模型：看到当前画面，猜下一帧会怎样。
2. 然后只在这个学到的"脑内模拟器"里反复试错，寻找高奖励的操作策略。
3. 最后把在模拟器里学到的策略拿回真实环境，如果模型足够准，现实中也能跑得很好。

Part 1

动机与核心思想

人类如何学习?



关键洞察：我们可以在"脑内模拟"中预演，而不必每次都真实尝试

传统 RL 的困境

问题	影响
样本效率低	需要数百万次真实交互
真实交互昂贵	机器人损耗、时间成本
探索危险	自动驾驶不能随意试错

World Models 的核心想法

> **学习一个环境模型，在"梦境"中训练策略**

- 少量真实数据学习世界模型
- 在想象中生成大量训练数据
- 策略迁移到真实环境

苏格拉底式追问：为什么要世界模型?

问题 1

如果每一次尝试都很贵（机器人、自动驾驶），还能像 DQN 一样疯狂试错吗?

问题 2

如果不给你真实环境，只给你一个高拟真的"模拟器"，你能在里面先练吗?

问题 3

那下一步：我们能让智能体**自己学出**这个"模拟器"吗?

讨论思路

- 现实世界 ≈ 一个昂贵的黑盒函数：`env(s, a) -> (s', r)`
- 世界模型 = 去学习一个便宜的"白盒近似"：`model(s, a) -> (s', r)`
- 一旦有了白盒，就可以：
  - 便宜地生成更多训练数据
  - 在不同假设下做"如果我这样做，会发生什么？"的推演

World Model 的形式化

$$P(s_{t+1}, r_t \mid s_t, a_t)$$

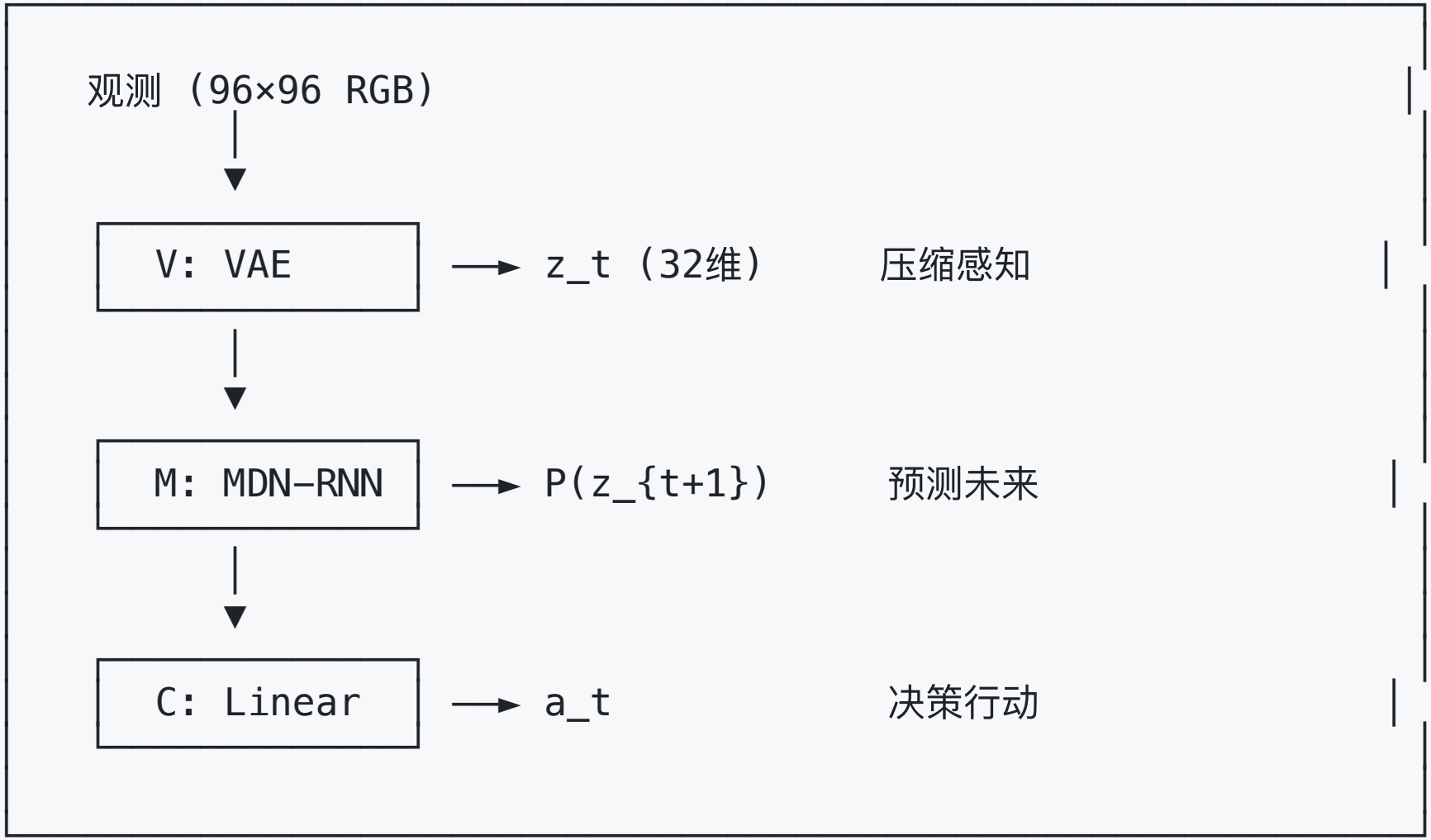
给定当前状态和动作，预测下一状态和奖励

```
真实环境:  s_t → [环境] → s_{t+1}, r_t    (昂贵)
              ↓
世界模型:  s_t → [模型] → s_{t+1}, r_t    (便宜)
```

Part 2

架构设计：V-M-C

三组件总体架构


观测 (96×96 RGB)
V: VAE
z_t (32维)
压缩感知
M: MDN-RNN
P(z_{t+1})
预测未来
C: Linear
a_t
决策行动

V: Vision Model (VAE)

作用：将高维图像压缩到低维潜在空间

- 输入：64×64×3 图像 → 输出：$z \in \mathbb{R}^{32}$
- 压缩比约 384:1

为什么用 VAE 而不是普通 AE?

- 概率潜在空间，更规整，方便采样和插值
- 适合在 latent 空间 roll out

VAE：给 12 岁小朋友的解释

- 想象你把一张 64×64 的游戏截图压缩成一张**小卡片**，上面只写下"对开车有用的信息"：
  - 赛道弯在哪里、车在什么位置、速度大概多快
- VAE 在做两件事：
  i. 学会把图片变成小卡片（Encoder）
  ii. 再把小卡片还原成图片（Decoder），要求肉眼看起来还像原图
- 训练好之后：
  - 我们不再直接处理图片，而是处理小卡片 `z`
  - 世界模型只需要预测"小卡片怎么变"，问题变得更简单

VAE 数学原理

损失函数 (ELBO)

$$\mathcal{L} = \underbrace{\|x - \hat{x}\|^2}_{\text{Recon}} + \beta \cdot \underbrace{D_{KL}(q(z \mid x) \| \mathcal{N}(0, I))}_{\text{KL}}$$

重建损失 + KL 散度正则化

重参数化技巧（关键！）

问题：$z \sim \mathcal{N}(\mu, \sigma^2)$ 采样操作不可微

解决：$z = \mu + \sigma \cdot \epsilon, \quad \epsilon \sim \mathcal{N}(0, 1)$

步骤	说明
$\epsilon \sim \mathcal{N}(0, 1)$	随机源（不需要梯度）

M: Memory Model (MDN-RNN)

作用：学习环境动态，预测未来潜在状态

- 输入：(z_t, a_t, h_{t-1})
- 模型：LSTM + MDN (K 个高斯混合)
- 输出：$P(z_{t+1})$ 的分布 + 终止信号 done

为什么要 MDN? 未来可能是多模态的（T 字路口左/右转）

MDN 数学原理

混合高斯分布

$$p(z_{t+1}) = \sum_{k=1}^{K} \pi_k \cdot \mathcal{N}(z_{t+1} \mid \mu_k, \sigma_k^2)$$

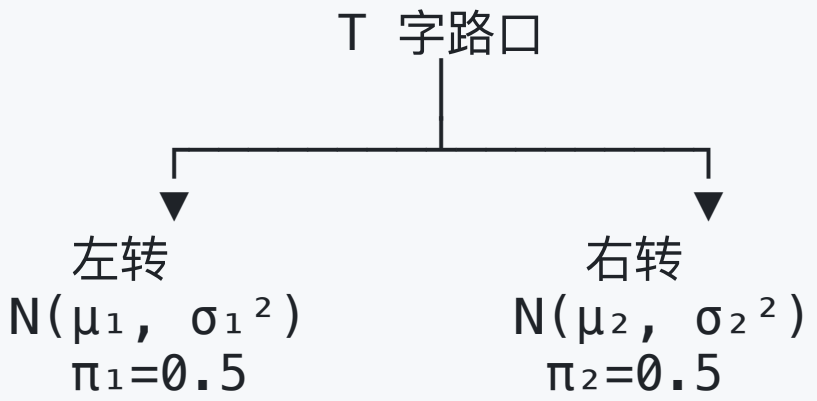
- π_k：混合权重，$\sum_k \pi_k = 1$（由 softmax 保证）
- μ_k, σ_k：第 k 个高斯的均值和标准差

损失函数（负对数似然）

$$\mathcal{L} = -\log \sum_{k=1}^{K} \pi_k \cdot \mathcal{N}(z_{t+1} \mid \mu_k, \sigma_k^2)$$

> 使用 **logsumexp** 技巧保证数值稳定性

MDN 图解：为什么需要多模态



单一高斯：μ = (μ₁+μ₂)/2 → 预测"直走" ✗

MDN：P(z) = 0.5·N(μ₁,σ₁²) + 0.5·N(μ₂,σ₂²) ✓

温度参数 τ：控制梦境"难度"

采样时的温度调节

$$\pi'_k = \frac{\pi_k^{1/\tau}}{\sum_j \pi_j^{1/\tau}}, \quad z \sim \mathcal{N}(\mu_k, \tau \cdot \sigma_k^2)$$

τ	效果	用途
τ < 1	更确定，走"熟悉的路"	简单模式
τ = 1	标准设置	默认
τ > 1	更随机，出现"意外情况"	**噩梦模式**

> **论文关键发现**：在 τ > 1 的"噩梦"中训练的 agent 更鲁棒！

C: Controller（线性策略）

作用：基于感知和记忆做出决策

```
# 输入: [z_t, h_t] = [32, 256] = 288 维
# 输出: a_t = 3 维 (steering, gas, brake)

action = W @ [z, h] + b   # 线性变换

# 参数量: 288 × 3 + 3 = 867 个
```

> 论文：故意保持 Controller 极简，避免它记住世界模型的漏洞

- 防止过拟合世界模型误差
- 强迫学"大方向"的鲁棒策略
- 适合用 CMA-ES 这种无梯度进化算法

V-M-C 架构自测：你能复述数据流吗?

如果不看图，请尝试回答：

1. 智能体现在的**眼睛**看到一帧画面 x_t，第一步去哪?
2. 怎么结合**过去的记忆** h_{t-1}?
3. **大脑**在做决策时，到底用了哪些信息?
4. 决策做完动作 a_t 后，谁负责**想象**下一帧?

参考答案

✅ **V (Encoder)**: $x_t \rightarrow z_t$ (压缩)

✅ **M (RNN)**: $[z_t, a_{t-1}, h_{t-1}] \rightarrow h_t$ (更新记忆)

✅ **C (Controller)**: $[z_t, h_t] \rightarrow a_t$ (基于当前感知+记忆做决策)

✅ **M (MDN)**: $h_t \rightarrow P(z_{t+1})$ (预测未来)

Part 3

训练流程 & 梦境 Rollout

训练流程：用"学开车"做类比

故事场景：你要在一个空荡荡的停车场学会漂移。

1. **Stage 1 (数据收集): 瞎开一通**
    - 你闭着眼乱踩油门、乱打方向，让人在旁边录像。
    - 目标：收集"我做了动作X，车身发生了什么变化Y"的原始素材。
2. **Stage 2 (训练世界模型): 回家看录像 & 脑补**
    - 回家躺在床上，回看录像（训练 VAE）。
    - 然后在脑子里总结规律（训练 RNN）："如果我当时猛打左舵，车头应该往左甩"。
3. **Stage 3 (梦境训练): 脑内模拟赛**

四阶段训练概览


Stage 1: 数据收集
随机策略在真实环境中 Rollout，收集 (图像，动作)
Stage 2a: 训练 VAE
图像 → z，最小化重建损失 + KL 正则
Stage 2b: 训练 MDN-RNN
(z, a) 序列 → 预测 z_{t+1} 分布
Stage 3: 在梦境中训练 Controller
固定 V+M，用 CMA-ES 在世界模型中进化策略

Stage 1: 数据收集（真实环境）

```
for episode in range(10000):
    obs = env.reset()
    for step in range(1000):
        action = env.action_space.sample()  # 随机探索
        frames.append(preprocess(obs))
        actions.append(action)
        obs, reward, done, _ = env.step(action)
        if done:
            break
```

- 约 1 万条轨迹，累计数百万帧
- 只做一次真实收集，后续训练主要在"梦境"里进行

Stage 2: 训练世界模型

- **2a：训练 VAE（图像 → z）**
  - 损失 = 重建误差 + KL 散度
  - 得到压缩的潜在表示 z_t
- **2b：训练 MDN-RNN（z, a → z'）**
  - 在 z 序列上训练序列模型
  - 输出下一步潜在状态的分布 $P(z_{t+1})$

> 得到一个可以"在 latent 空间中模拟未来"的世界模型

Stage 3: 梦境中训练 Controller

```
for generation in range(300):
    population = cmaes.ask()  # 采样一批候选参数
    fitness = []
    for params in population:
        controller.set_params(params)
        rewards = [dream_rollout(controller)
                   for _ in range(16)]
        fitness.append(mean(rewards))
    cmaes.tell(population, fitness)  # 更新分布
```

- 所有评估都在世界模型里完成，无需真实环境
- 通过进化搜索找到在梦境中表现最好的线性策略

梦境 Rollout（核心直觉）

```
def dream_rollout(controller):
    z = vae.encode(random_frame)  # 初始潜在状态
    hidden = None
    total_reward = 0
    for step in range(1000):
        h = hidden[0] if hidden else zeros
        action = controller([z, h])
        pi, mu, sigma, reward, done, hidden = rnn(z, action, hidden)
        z = sample_mdn(pi, mu, sigma)  # 在 latent 中前进一步
        total_reward += reward
        if done > 0.5:
            break
    return total_reward
```

> 整个轨迹都发生在 latent 空间的"梦境"中

CMA-ES vs 梯度下降（直觉）

方面	梯度下降	CMA-ES
梯度需求	需要可微的动态模型	无需梯度
长轨迹	易梯度爆炸/消失	只看最终得分
随机性	高方差	多次评估平均
适用参数量	任意	小于 ~1k 最合适

> 类比：
>
> - 梯度下降：盲人摸着斜坡走
> - CMA-ES：往几个方向扔石头，看哪个滚得最远

思考题：如果梦境是错的?

思想实验

假设你在脑海中认为："只要踩刹车，车速就会变快"（错误的物理规律）。

你在梦里训练出的策略会是什么样?

推演结果

- 你会拼命踩刹车，因为在你的梦里这能拿高分。
- **现实中**：车停了，比赛输了。
- **结论：Agent 会利用世界模型的漏洞作弊。**
  - 这就是 **Dream-Reality Gap** 的根源。
  - 也是为什么我们需要偶尔回到真实世界校准模型。

Part 4

实验结果 & 局限

论文参数 & 复现实验

组件	参数	值
数据	Rollouts	10,000
	Max steps	1,000
VAE	Latent dim	32
	LR	1e-4
MDN-RNN	Hidden	256
	Gaussians K	5
CMA-ES	Population	64
	Generations	300

CarRacing 结果对比（论文）

方法	得分
随机策略	~0
纯梦境训练	~900
梦境 + 微调	**906**
人类水平	~900

> 在完全基于想象训练的前提下，CarRacing 可达到接近人类水平

CartPole 实验的反例

方法	Dream	Real	Gap
Simple WM (LSTM)	~103	~17	**6x**
Full WM (MDN-LSTM)	~208	~9.6	**22x**
DQN (baseline)	-	~193	-

为什么失败?

1. 用随机策略收集的数据质量差
2. CartPole 对初始条件和微小误差极其敏感
3. 原设计偏向高维图像任务，低维状态未必适配

> 启示：World Models 不是银弹，要与任务匹配

World Models 的主要局限

1. **分阶段训练**
   - V、M、C 分开训练，不能端到端联合优化
   - V/M 中的缺陷无法通过策略学习被反向纠正
2. **简单 Controller 的表达力有限**
   - 867 参数的线性策略
   - 对复杂任务可能不足
3. **世界模型误差累积**
   - 长轨迹 roll out 时误差逐步放大
4. **"作弊"问题**
   - Agent 学会利用模型缺陷获取高分，而非学习真实世界规律

Part 5

另一条路线与后续发展

另一条路线：Decision Transformer

核心思想：把 RL 变成序列建模

- **传统 RL (World Models)**: 学习 P(s'|s,a)，在想象中规划
- **Decision Transformer**: 学习 P(a | R̂, s)，条件生成动作
  - R̂ = Return-to-Go (从当前到结束的累积奖励)

Credit Assignment 类比

方法	类比	信息传播
TD Learning	RNN	逐步反向传播，误差累积
Decision Transformer	Transformer	Self-Attention 直接全局连接

World Models vs Decision Transformer

维度	World Models	Decision Transformer
核心	学习世界如何运转	学习好轨迹长什么样
是否学世界模型	是	否
能否超越数据	是（想象中探索）	否（受限于数据）
稀疏奖励	困难	自然处理

本质区别：

- World Models: "理解世界" → 想象 → 决策
- Decision Transformer: "模仿成功" → 条件生成

从 World Models 到 Dreamer

```
World Models (2018)
        |
        ▼
PlaNet (2019)
  - RSSM: 确定性 + 随机性双路径
  - MPC 规划
        |
        ▼
Dreamer (2020)
  - Actor-Critic 在想象中训练
  - 端到端联合优化
        |
        ▼
DreamerV2/V3 (2021/2023)
  - 离散 latent
  - 多任务通用世界模型
```

Dreamer 相对 World Models 的改进

维度	World Models	Dreamer
训练方式	分阶段（先 V/M 再 C）	端到端联合训练
控制器	线性 + CMA-ES	Actor-Critic + 梯度
状态表示	纯随机 z	RSSM：确定性 h_t + 随机 z_t
优化目标	最终 episodic reward	TD 目标 + Entropy Regularization

> 整体上：从"先建好世界再在梦里搜索" → "一边学世界一边学行为"

总结

核心要点回顾

World Models 的贡献

1. 提出了 "在梦境中训练" 的可行路径
2. 给出了 V-M-C 的通用架构模板
3. 强调了简单 Controller 对鲁棒性的作用
4. 对后续 PlaNet / Dreamer 系列有直接影响

关键技术组件

- VAE：压缩高维观测
- MDN-RNN：建模多模态未来
- CMA-ES：无梯度策略优化
- 温度参数 τ：控制梦境难度与随机性

一句话总结

> 学习一个世界模型，在"梦"里训练策略，再迁移回现实。

Q&A & 讨论问题

可以深入讨论的问题：

1. **数据收集**：随机策略 vs 专家策略，哪个更好？为什么论文用随机？
2. **模型容量**：线性 Controller 只有 867 参数，这是优点还是限制？
3. **Dream-Reality Gap**：如何减小梦境和现实的差距？
4. **应用场景**：哪些领域适合 World Models？（机器人？游戏？自动驾驶？）

关键公式速查

模块	公式
VAE 损失	$\mathcal{L} = \|x - \hat{x}\|^2 + \beta \cdot D_{KL}$
KL 散度	$D_{KL} = -\frac{1}{2}\sum(1 + \log\sigma^2 - \mu^2 - \sigma^2)$
重参数化	$z = \mu + \sigma \cdot \epsilon,\ \epsilon \sim \mathcal{N}(0, 1)$
MDN	$p(z) = \sum_k \pi_k \cdot \mathcal{N}(z \mid \mu_k, \sigma_k^2)$
温度采样	$\pi'_k \propto \pi_k^{1/\tau},\ \sigma' = \sqrt{\tau} \cdot \sigma$
Controller	$a = \text{activation}(W \cdot [z, h] + b)$

参考资料

- **论文**: https://arxiv.org/abs/1803.10122
- **官网**: https://worldmodels.github.io/
- **代码**: `world_models/experiments/3_car_racing_world_model.py`
- **完整版**: `World_Models_Presentation_long.md` (90 min, 含代码和互动)