World Models（完整版 ~90–120 min）

在梦境中学习：基于世界模型的强化学习

Ha & Schmidhuber, 2018

版本说明 & 使用建议

- 本稿是 **完整版 / Workshop 版本**：
  - 包含：
    - 动机与宏观架构（核心）
    - VAE / MDN 的关键数学公式（进阶）
    - 训练流程伪代码 + 代码结构（工程）
    - 复现实验结果与失败案例（经验）
    - 附录中的公式速查与目录说明
- 建议用法：
  - 30–40 min 只讲标记为 **核心主线** 的部分（可配合 `World_Models_Presentation_short.md`）
  - 90–120 min Workshop 可带着大家推公式、看代码、讨论实验细节
- 结构基本与原始版本一致，只额外加入了本页说明

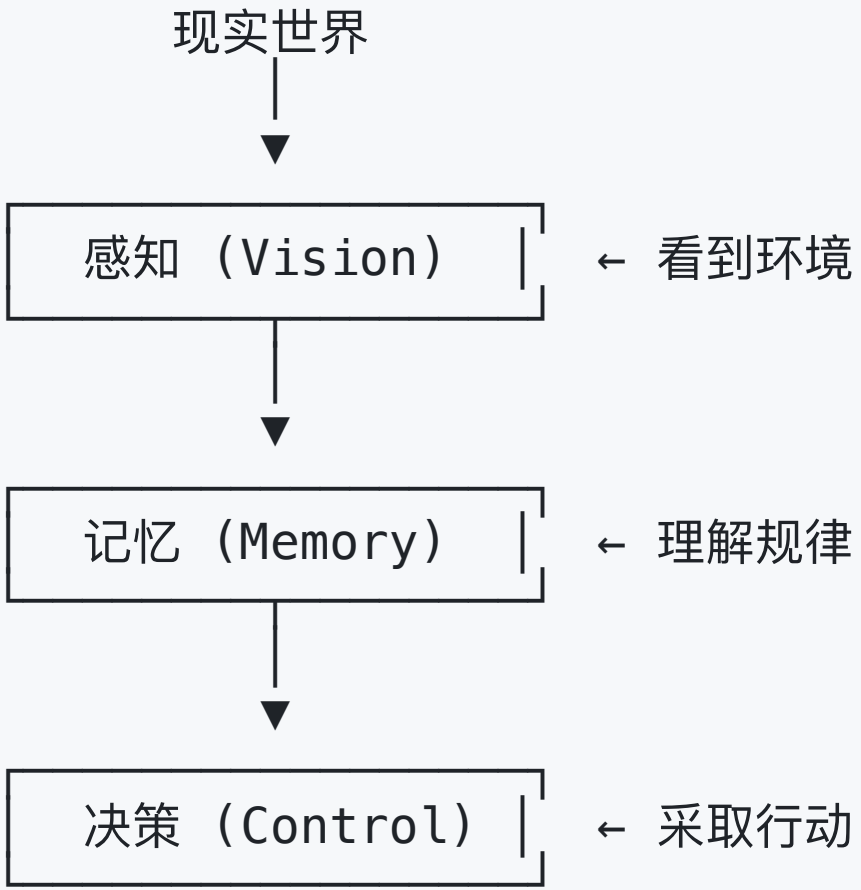
目录

> ★ = 30 min 精简版必讲内容

Part 1

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

World Models 的核心问题

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 的解决方案

> 学习一个环境模型，在"梦境"中训练策略

- 只需少量真实数据学习世界模型
- 在想象中生成无限训练数据

苏格拉底式提问：从 DQN 到 World Models

Q1. 如果每次和真实环境交互都要花很多钱/时间，你还会怎么设计 RL?
Q2. 如果有人送你一个完美模拟器，你会不会**先在模拟器里学，再上真机**?
Q3. 那能不能连模拟器也一起学出来?

引导答案（讨论用）

- DQN: 把环境当作"只能在线调用的黑盒"，想学好只能多试。
- World Models: 先把这个黑盒"白盒化"成 `model(s, a) -> (s', r)`，再在里面大量试错。
- 好处:
  - 真实世界只承担"采样世界模型"的工作
  - 大部分策略优化都转移到 GPU 上的"梦境"中完成

World Models 核心公式

$$\text{World Model: } P(s_{t+1}, r_t | s_t, a_t)$$

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    (免费)
```

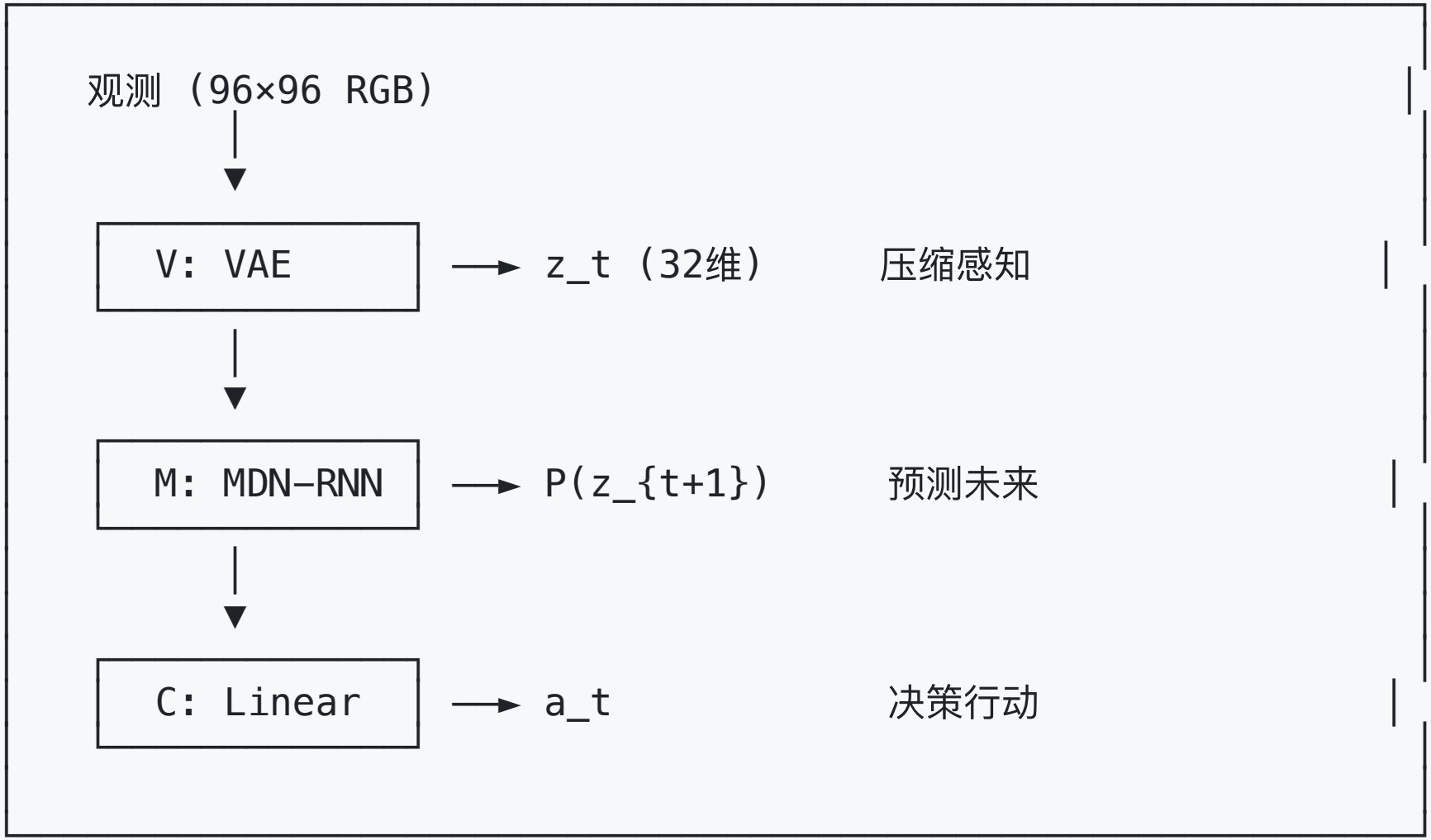
训练流程

1. 收集少量真实数据
2. 训练世界模型
3. 在世界模型中训练策略
4. 策略迁移到真实环境

Part 2

架构设计：V-M-C

三组件架构


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 = 12,288 维
          │
          ▼ 卷积编码器
    [32→64→128→256 channels]
          │
          ▼
输出: z ∈ R^32 (32 维)

压缩比: 384:1
```

为什么用 VAE 而不是普通 AutoEncoder?

- VAE 的潜在空间**规整**，便于采样
- 可以在潜在空间**插值**，生成平滑过渡

M: Memory Model (MDN-RNN)

作用：学习环境动态，预测下一状态

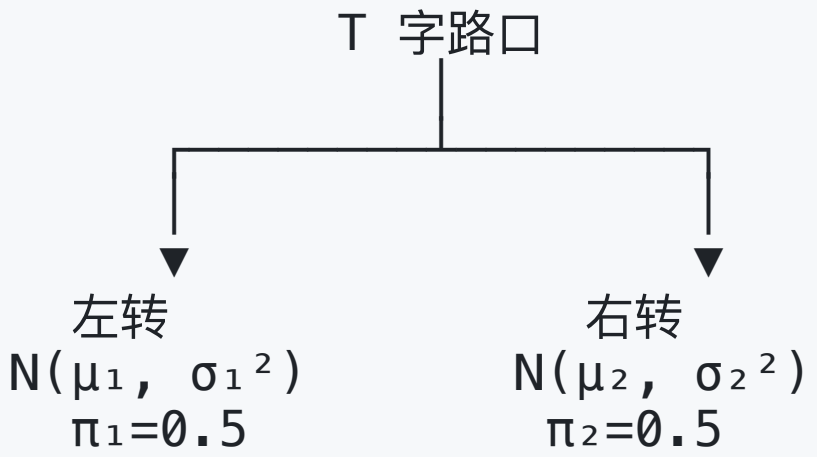
```
输入: (z_t, a_t, h_{t-1})
          │
          ▼ LSTM
    h_t = LSTM(z_t, a_t, h_{t-1})
          │
          ▼ MDN
输出: P(z_{t+1}) = Σ πᵢ · N(μᵢ, σᵢ²)
```

为什么用 MDN (Mixture Density Network)?

场景：赛车在 T 字路口，下一步可能左转或右转

- 单一高斯：预测"直走"（两个方向的均值）❌
- **MDN**：输出两个高斯，准确表达分叉 ✓

MDN 图解



单一高斯预测：$\mu = (\mu_1 + \mu_2)/2$ → 直走（错误！）

MDN 预测：$P(z) = 0.5 \cdot N(\mu_1, \sigma_1^2) + 0.5 \cdot N(\mu_2, \sigma_2^2)$

↓

准确表达两种可能的未来

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
```

为什么用如此简单的 Controller?

> *"We deliberately use a small controller to prevent it from memorizing the game."*
> — Ha & Schmidhuber, 2018

- **防止过拟合**世界模型的缺陷
- 强迫学习**真正有用的策略**

Part 3

数学原理

VAE 数学推导

目标：最大化数据的对数似然

$$\log p(x) \geq \mathbb{E}_{q(z|x)}[\log p(x|z)] - D_{KL}(q(z|x)\|p(z))$$

损失函数 (ELBO)

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

KL 散度闭式解

$$D_{KL} = -\frac{1}{2}\sum_{i=1}^{d}(1 + \log\sigma_i^2 - \mu_i^2 - \sigma_i^2)$$

重参数化技巧

问题：采样操作不可微，梯度无法传播

```
z ~ N(μ, σ²)  ←  采样操作，无法求梯度
```

解决：重参数化

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

```
          ε ~ N(0,1)      （采样，但不需要梯度）
                │
                ▼
z = μ + σ · ε          （可微操作）
    ↑   ↑
    │   └── 来自 encoder
    └────── 来自 encoder
```

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，$\sum_k \pi_k = 1$
- μ_k：第 k 个高斯的均值
- σ_k：第 k 个高斯的标准差

损失函数（负对数似然）

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

温度参数 τ

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	应用场景
τ < 1	更确定，权重更尖锐	容易训练
τ = 1	标准设置	默认
τ > 1	更随机，权重更平滑	训练更鲁棒的策略

论文发现：在 τ > 1 的"噩梦"中训练的 agent 更鲁棒！

Part 4

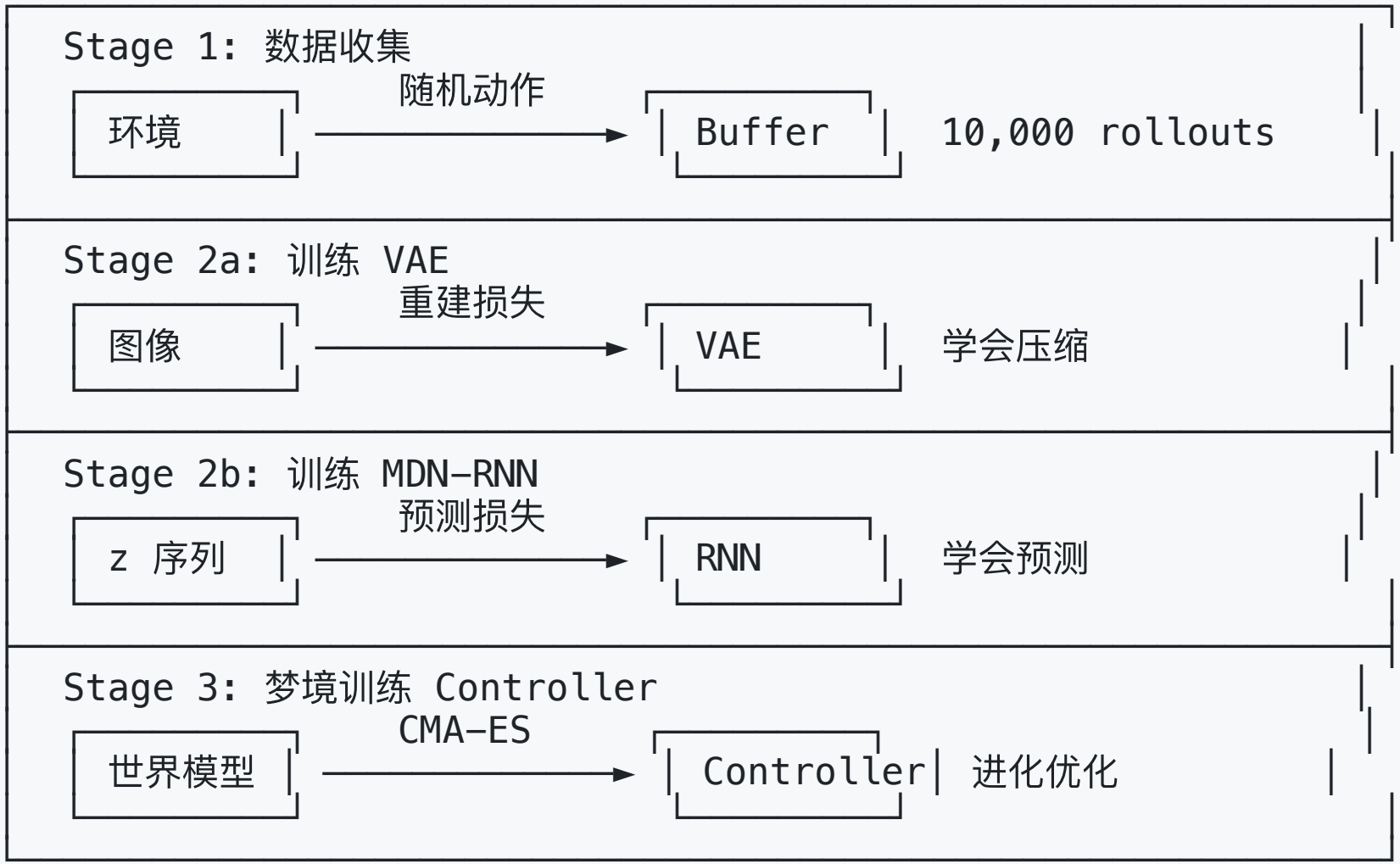
训练流程

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


Stage 1: 数据收集
环境
随机动作
Buffer
10,000 rollouts
Stage 2a: 训练 VAE
图像
重建损失
VAE
学会压缩
Stage 2b: 训练 MDN-RNN
z 序列
预测损失
RNN
学会预测
Stage 3: 梦境训练 Controller
世界模型
CMA-ES
Controller
进化优化

Stage 1: 数据收集

```
for episode in range(10000):
    obs = env.reset()

    for step in range(1000):
        # 随机动作探索
        action = env.action_space.sample()

        # 存储数据
        frames.append(preprocess(obs))
        actions.append(action)

        # 环境步进
        obs, reward, done, _ = env.step(action)

        if done:
            break
```

收集约 300 万帧数据

Stage 2: 训练世界模型

2a: 训练 VAE

```
for epoch in range(10):
    for batch in data_loader:
        recon, mu, logvar, z = vae(batch)
        loss = recon_loss + kl_loss
        loss.backward()
        optimizer.step()
```

2b: 训练 MDN-RNN

```
for epoch in range(20):
    for z_seq, a_seq, z_next in sequences:
        pi, mu, sigma, _, _, _ = rnn(z_seq, a_seq)
        loss = mdn_nll(pi, mu, sigma, z_next)
        loss.backward()
```

Stage 3: 梦境训练

```
# CMA-ES 进化优化
for generation in range(300):
    # 生成候选 Controller
    population = cmaes.ask()  # 64 个候选

    fitness = []
    for params in population:
        controller.set_params(params)

        # 在梦境中评估
        rewards = [dream_rollout(controller)
                   for _ in range(16)]
        fitness.append(mean(rewards))

    # 更新分布
    cmaes.tell(population, fitness)
```

关键：整个评估在世界模型中进行，无需真实交互！

梦境 Rollout

```
def dream_rollout(controller):
    z = vae.encode(random_frame)  # 初始状态
    hidden = None
    total_reward = 0

    for step in range(1000):
        # Controller 决策
        h = hidden[0] if hidden else zeros
        action = controller([z, h])

        # 世界模型预测
        pi, mu, sigma, reward, done, hidden = rnn(z, action, hidden)

        # 采样下一状态
        z = sample_mdn(pi, mu, sigma)
        total_reward += reward

        if done > 0.5:
            break

    return total_reward
```

CMA-ES vs 梯度下降

为什么用 CMA-ES?

方面	梯度下降	CMA-ES
梯度需求	需要通过采样反传	无需梯度
长轨迹	梯度爆炸/消失	直接评估终点
随机性	高方差	多次评估平均
参数量	任意	适合 <1000

CMA-ES 类比

```
梯度下降: 盲人摸着斜坡走 (需要感知坡度)
CMA-ES:    往几个方向扔石头，看哪个滚得最远
```

Part 5

代码实现

代码结构

```
3_car_racing_world_model.py
│
├── Config                  # 配置参数
│
├── ConvVAE                 # Vision Model
│   ├── encode()            # 图像 → (μ, σ)
│   ├── reparameterize()    # 重参数化采样
│   └── decode()            # z → 图像
│
├── MDNRNN                  # Memory Model
│   ├── forward()           # 序列预测
│   ├── mdn_loss()          # 负对数似然
│   └── sample()            # 从 MDN 采样
│
├── Controller              # 线性策略
│   └── get_action()        # [z, h] → action
│
├── SimpleCMAES             # 进化算法
│   ├── ask()               # 生成候选
│   └── tell()              # 更新分布
│
└── CarRacingWorldModel     # 主类
    ├── collect_data()
    ├── train_vae()
    ├── train_rnn()
    ├── train_controller()
    └── evaluate_real()
```

ConvVAE 实现

```python
class ConvVAE(nn.Module):
    def __init__(self, latent_size=32):
        # Encoder: 4 层卷积
        self.encoder = nn.Sequential(
            nn.Conv2d(3, 32, 4, stride=2),   # 64→32
            nn.Conv2d(32, 64, 4, stride=2),  # 32→16
            nn.Conv2d(64, 128, 4, stride=2), # 16→8
            nn.Conv2d(128, 256, 4, stride=2) # 8→4
        )
        self.fc_mu = nn.Linear(4096, 32)
        self.fc_logvar = nn.Linear(4096, 32)

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return mu + eps * std  # 关键：可微采样
```

MDN-RNN 实现

```
class MDNRNN(nn.Module):
    def __init__(self):
        self.lstm = nn.LSTM(35, 256)  # 输入: z(32) + a(3)

        # MDN 输出层
        self.fc_pi = nn.Linear(256, 5)        # 5 个混合权重
        self.fc_mu = nn.Linear(256, 5*32)     # 5×32 个均值
        self.fc_sigma = nn.Linear(256, 5*32) # 5×32 个标准差

    def forward(self, z, action, hidden):
        x = torch.cat([z, action], dim=-1)
        out, hidden = self.lstm(x, hidden)

        pi = F.softmax(self.fc_pi(out), dim=-1)
        mu = self.fc_mu(out).view(-1, 5, 32)
        sigma = torch.exp(self.fc_sigma(out))

        return pi, mu, sigma, hidden
```

Controller 实现

```python
class Controller:
    def __init__(self, input_dim=288, action_dim=3):
        # 极简：仅 867 个参数
        self.W = np.random.randn(3, 288) * 0.1
        self.b = np.zeros(3)

    def get_action(self, state):
        raw = self.W @ state + self.b

        # CarRacing 动作映射
        return [
            np.tanh(raw[0]),      # steering: [-1, 1]
            sigmoid(raw[1]),      # gas: [0, 1]
            sigmoid(raw[2])       # brake: [0, 1]
        ]
```

Part 6

实验结果

论文参数设置

组件	参数	值
数据	Rollouts	10,000
	Max steps	1,000
VAE	Latent dim	32
	Learning rate	0.0001
MDN-RNN	Hidden size	256
	Gaussians (K)	5
CMA-ES	Population	64
	Generations	300
	Eval rollouts	16

CarRacing 复现结果

实验对比

设置	Dream Fitness	Real Reward	状态
Quick (50 rollouts)	~107	~13	完成
Paper (10000 rollouts)	进行中	进行中	**~6 天 ETA**
论文报告	-	**~900**	参考

复现进度（实时更新）

- **数据收集**: 940/10000 rollouts (9.4%)
- **存储**: 9 chunks × ~83MB = 750MB（内存优化后）
- **预计总时间**: ~4-6 天

论文核心结果

CarRacing-v0

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

VizDoom (Take Cover)

- 在梦境中训练的 agent 表现良好
- 但发现了"作弊"问题：agent 学会利用世界模型的缺陷

Part 7

局限与展望

互动预测：CartPole 为什么会失败?

场景设定

- CartPole (倒立摆) 是一个极简单的任务（状态只有 4 维）。
- CarRacing 是复杂的视觉任务（几千维像素）。
- World Models 在 CarRacing 上成功了，但在 CartPole 上却**惨败**。

请大家猜猜原因?

1. 是因为 CartPole 只有 4 维，太简单不需要压缩?
2. 是因为随机策略收集的数据有问题?
3. 还是因为模型预测的微小误差在长序

> ***提示***：*想象你在平衡一根筷子。如果你的眼睛（感知）或者大脑（预测）有一丁点延迟或误差，筷子会怎样?*

实验洞察：CartPole 的教训

CartPole 上的 World Model 实验

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 CartPole 上 World Model 失败?

1. **数据质量差**: 随机策略平均仅 ~22 分
2. **状态空间敏感**: 虽然只有 4 维，但动态对初始条件极度敏感
3. **原论文设计**: VAE 针对高维图像，低维状态反而不适合

World Models 的局限性

1. 分阶段训练

```
问题: V, M, C 分开训练，无法联合优化

V (VAE)     ──训练──►  固定
M (RNN)     ──训练──►  固定
C (Controller) ──────►  只优化 C

如果 V 或 M 有缺陷，C 只能"适应"，无法改进它们
```

2. 简单 Controller 的表达力限制

- 867 参数的线性策略
- 复杂任务可能不够

World Models 的局限性 (续)

3. 世界模型误差累积

```
真实轨迹:    s₀ → s₁ → s₂ → s₃ → ...
                ↓      ↓      ↓
模型轨迹:    s₀ → s'₁ → s'₂ → s'₃ → ...
                 ε₁     ε₂     ε₃

误差累积: ε₃ ≈ ε₁ + ε₂ + ε₃  (越来越偏离真实)
```

4. "作弊"问题

> Agent 可能学会利用世界模型的缺陷来获得高分

例：在 VizDoom 中，agent 学会在特定区域"自杀"来逃避火球

后续发展

```
World Models (2018)
        |
        ▼
PlaNet (2019)
  - RSSM: 确定性 + 随机性双路径
  - MPC 规划替代学习 Controller
        |
        ▼
Dreamer (2020)
  - Actor-Critic 在想象中训练
  - 端到端联合优化
        |
        ▼
DreamerV2 (2021)
  - 离散潜在变量
  - Atari 上超越人类
        |
        ▼
DreamerV3 (2023)
  - 通用架构，无需任务特定调参
```

另一条路线：Decision Transformer

核心思想：把 RL 变成序列建模

```
传统 RL (World Models/Dreamer):
  学习 P(s'|s,a)，在想象中规划

Decision Transformer:
  把轨迹看成序列: τ = (R̂₁, s₁, a₁, R̂₂, s₂, a₂, ...)
  学习 P(aₜ | R̂ₜ, sₜ, 历史)

  R̂ₜ = Return-to-Go = 从 t 到结束的累积奖励
```

关键洞察：不预测"最优动作"，而是"想要 X 分时该怎么做"

Credit Assignment: TD vs DT

类比：RNN vs Transformer 的信息传播

World Models vs Decision Transformer

维度	World Models/Dreamer	Decision Transformer
核心思想	学习世界如何运转	学习好轨迹长什么样
学习目标	状态转移 $P(s' \mid s,a)$	动作生成 $P(a \mid \hat{R},s)$
是否学世界模型	是	否
能否 Planning	是（想象中规划）	否
能否超越数据	是（想象中探索）	否（受限于数据）
Credit Assignment	Bellman backup	Self-Attention
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

Dreamer 的改进

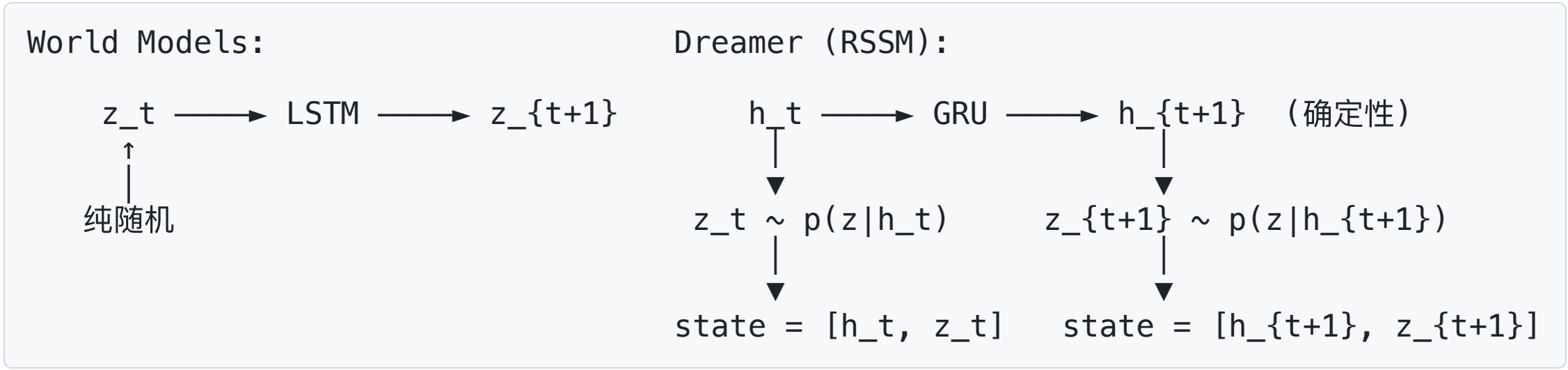
问题	World Models	Dreamer
训练方式	分阶段	端到端联合
Controller	线性 + CMA-ES	Actor-Critic + 梯度
状态表示	纯随机 z	RSSM (确定+随机)
优化目标	最终奖励	时序差分 (TD)

RSSM (Recurrent State-Space Model)

$$h_t = f(h_{t-1}, z_{t-1}, a_{t-1}) \quad \text{(确定性路径)}$$

$$z_t \sim p(z_t|h_t) \quad \text{(随机性路径)}$$

RSSM 架构图解



为什么 RSSM 更好?

- **确定性路径 h**：记住长期信息，避免"遗忘"
- **随机性路径 z**：建模不确定性和多模态
- **双路径结合**：既稳定又灵活

总结

核心要点

World Models 的贡献

1. **证明了"梦境训练"的可行性**
2. **提出了 V-M-C 架构框架**
3. **揭示了简单 Controller 的重要性**
4. **开创了 Model-Based RL 新范式**

关键技术

- **VAE**：压缩高维观测
- **MDN-RNN**：预测多模态未来
- **CMA-ES**：无梯度策略优化
- **温度参数**：控制梦境随机性

一句话总结

> **学习一个世界模型，在"梦"里训练策略，迁移到现实**

"我们不需要理解每一滴雨如何落下，
只需要知道：下雨了，打伞。"

—— 世界模型的哲学

Workshop 互动环节

(适用于 90 min 完整版)

动手练习 1：理解 VAE 重参数化

问题：为什么 `z = μ + σ * ε` 可以让梯度传播，而 `z ~ N(μ, σ²)` 不行？

```
# 不可微版本
z = torch.normal(mu, sigma)  # 采样操作，梯度断裂

# 可微版本（重参数化）
eps = torch.randn_like(sigma)  # 固定随机源
z = mu + sigma * eps           # 纯算术操作，梯度可传
```

讨论：画出计算图，标出梯度流向。

动手练习 2：MDN 多模态预测

场景：T 字路口，50% 概率左转，50% 概率右转。

问题：

1. 如果用单一高斯，预测结果是什么？
2. 如果用 2-component MDN，输出应该是什么？
3. 如何从 MDN 输出中采样下一状态？

讨论问题

1. **数据收集策略**
   - 随机策略 vs ε-greedy vs 专家数据
   - 哪种数据分布更有利于学习世界模型?
2. **Controller 复杂度**
   - 如果用神经网络替代线性 Controller 会怎样?
   - 为什么论文强调"简单"?
3. **实际应用**
   - 自动驾驶中能用 World Models 吗?
   - 机器人控制中的 Sim2Real 问题如何解决?

Q&A

参考 & 附录

参考资料

- 论文: https://arxiv.org/abs/1803.10122
- 官网: https://worldmodels.github.io/
- 代码: `experiments/3_car_racing_world_model.py`

附录：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	$p(z) = \sum_k \pi_k \cdot \mathcal{N}(z \mid \mu_k, \sigma_k^2)$
MDN 损失	$\mathcal{L} = -\log \sum_k \pi_k \cdot \mathcal{N}(z \mid \mu_k, \sigma_k^2)$
Controller	$a = \text{activation}(W \cdot [z, h] + b)$

附录：代码文件结构

```
world_models/
├── 01_world_models_concept.md     # 概念理解
├── 02_vae_math.md                 # VAE 数学原理
├── 03_rnn_mdn_math.md             # RNN/MDN 数学原理
├── 04_paper_review.md             # 论文精读
├── 05_code_walkthrough.md         # 代码精读
├── World_Models_Presentation.md   # 原始演示文稿（参考）
│
└── experiments/
    ├── 1_baseline_dqn.py          # DQN Baseline
    ├── 2_world_model_full.py      # CartPole World Model
    └── 3_car_racing_world_model.py # CarRacing (带 checkpoint)
```