Lecture Notes for

Neural Networks and Machine Learning

Q-Learning

Course Retrospective

World Models

Logistics and Agenda

- Logistics
  - Final Paper Due at end of Finals (**May 11**)
  - This is last lecture!!
  - *I will post other lecture slides for those interested*
- Agenda
  - Review Value Iteration
  - Q-Learning
  - Deep Q-Learning
  - Class Retrospective

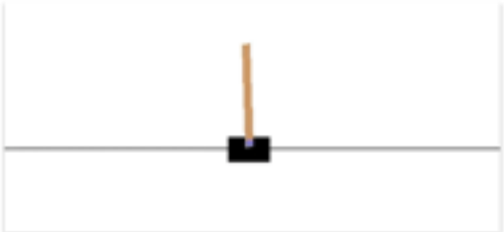
Last Time

```python
import gym


if __name__ == "__main__":
    env = gym.make("CartPole-v0")

    total_reward = 0.0
    total_steps = 0
    obs = env.reset()

    while True:
        action = env.action_space.sample()
        obs, reward, done, _ = env.step(action)
        total_reward += reward
        total_steps += 1
        if done:
            break
```



Action Space: One input, [0, 1] pull left or pull right

Obs Space: Dynamic state variables (continuous and four dimensional)

End: When more than 15 degrees off or too far from center

Reward: +1 for each time step

Value Iteration (Value Based)

- **Direct**:
  - Initialize V(s) to all zeros
  - Take a series of random steps, then follow policy
  - Perform value iteration: $V(s) \leftarrow \max_{a \in A} \sum_{\tilde{s} \in S} p_{a,s\to\tilde{s}} \cdot (r_{s,a,\tilde{s}} + \gamma V(\tilde{s}))$
  - Repeat until V(s) stops changing
- **Q-Function Variant**:
  - Initialize Q(s,a) to all zeros
  - Take a series of random steps, then follow policy
  - Perform value iteration: $Q(s,a) \leftarrow \sum_{\tilde{s} \in S} p_{a,s\to\tilde{s}} \cdot (r_{s,a,\tilde{s}} + \gamma \max_{a'} Q(\tilde{s}, a'))$
  - Repeat until Q is not changing

Need to estimate $p_{a,s\to s'}$ Via observed **Transitions**

With infinite time and exploration, this update will **Converge to Optimal Policy**

Value Iteration
Reinforcement Learning

M. Lapan Implementation for and Frozen Lake

Using Cross Entropy on the Frozen Lake

The setup of the lake is as follows: Observations space, integer, based on the square you select. There are holes, frozen spaces, and a goal. The reward only happens at the end, otherwise the reward is zero.

```
SFFF
FHFH
FFFH
HFFG
```

To encode this observation space, we will convert the integer value (1-16) into a one hot encoded categorical value.

The action space is defined as moving left(1), right(2), up(3), down(4), which will be the output of the network.

Follow Along:
08a_Basics_Of_Reinforcement_Learning.ipynb

Value Iteration (Review)

- **Direct Value**:
  - Initialize V(s) to all zeros
  - Take a series of random steps, then follow policy
  - Perform value iteration: $V(s) \leftarrow \max_{a \in A} \sum_{\hat{s} \in S} p_{a,s \to \hat{s}} \cdot (r_{s,a,\hat{s}} + \gamma V(\hat{s}))$
  - Repeat until V(s) stops changing

Need to estimate $p_{a,s \to s'}$
Via observed **Transitions**

- **Q-Function Variant**:
  - Initialize Q(s,a) to all zeros
  - Take a series of random steps, then follow policy
  - Perform value iteration: $Q(s,a) \leftarrow \sum_{\hat{s} \in S} p_{a,s \to \hat{s}} \cdot (r_{s,a,\hat{s}} + \gamma \max_{a'} Q(\hat{s},a'))$
  - Repeat until Q is not changing

With infinite time and exploration, this update will
Converge to Optimal Policy

Value Iteration Reinforcement Learning

M. Lapan Implementation for
and Frozen Lake

"Finish"

Follow Along:
08a_Basics_Of_Reinforcement_Learning.ipynb

Value Iteration Limitations

- Q and V can get really big for **large states and action spaces**
- Transition matrix can get gigantic for large state and action spaces
  - We will solve this by dropping the transition probabilities in Q function update
- This Variant is known as Q-Learning
- (*not addressing yet…*) Q-table needs infinite inputs when the state spaces are **continuous**
  - We will solve this by using a neural network to **approximate** the Q function

Tabular Q-Learning Algorithm

- In update, **ignore the transition probability**, making use of the iterative nature of Q, Bellman Update:

$$Q(s_t, a_t) = r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 \ldots$$
$$Q(s_t, a_t) = r_0 + \gamma (r_1 + \gamma^2 r_2 + \gamma^3 r_3 \ldots)$$
$$Q(s_t, a_t) = r_0 + \gamma \max_a Q(s_{t+1}, a)$$

$$Q(s, a) = r_{s,a} + \gamma \max_{a' \in A} Q(s', a')$$

- For stability, add momentum to the **Bellman update** equation

$$Q(s, a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \cdot [r_{s,a} + \gamma \max_{a' \in A} Q(s', a')]$$

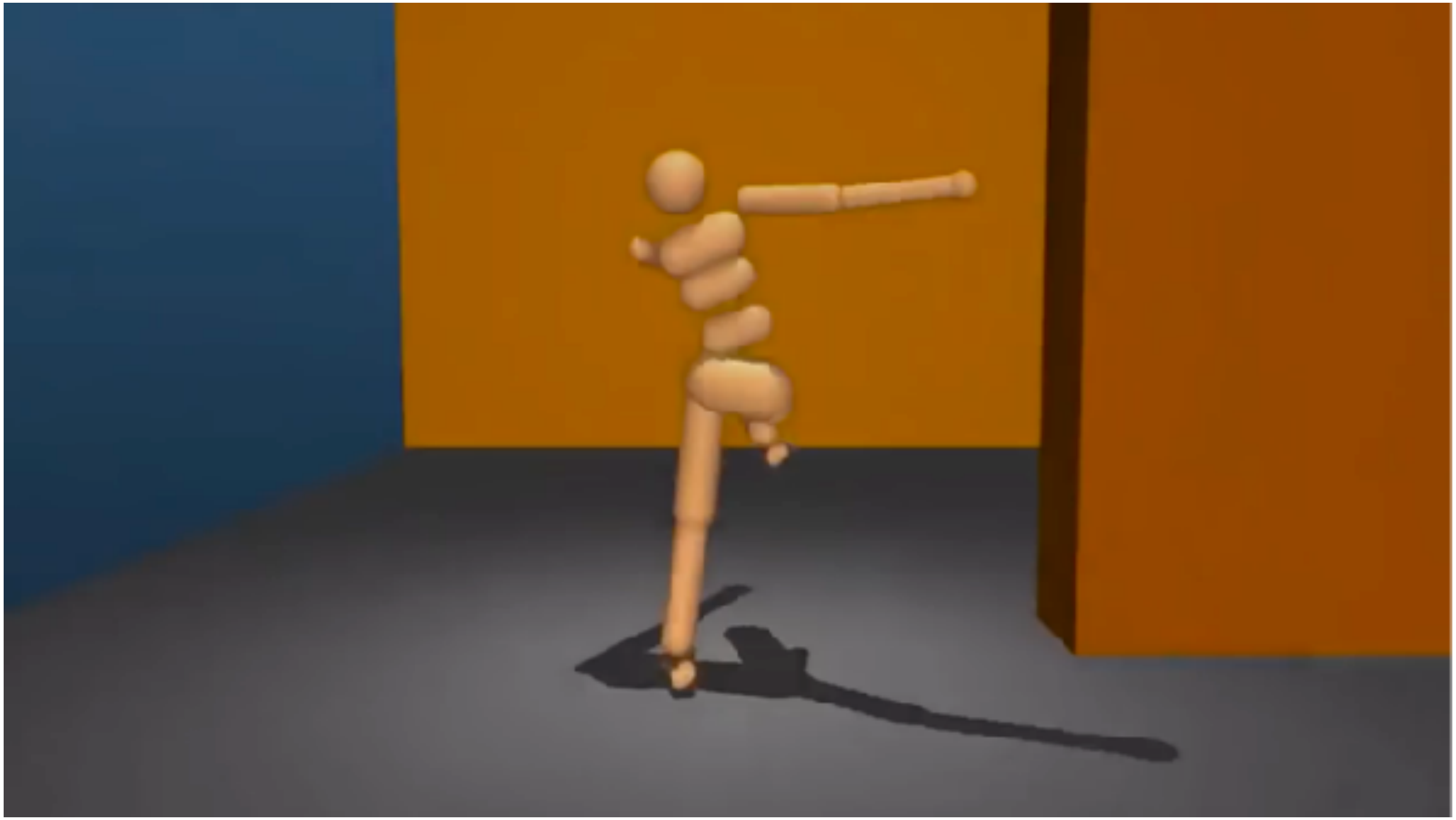
- Algorithm, start with empty $Q(s,a)$:
  - Sample (with rand) from environment, (s, a, r, s')
  - Make Bellman Update with Momentum
  - Repeat until desired performance

Tabular Q-Learning Reinforcement Learning

M. Lapan Implementation for and Frozen Lake

Follow Along:
08a_Basics_Of_Reinforcement_Learning.ipynb

Deep Q-Learning



Q-Learning with a Neural Network

- Want to approximate $Q(s,a)$ when the state space is potentially large. Given s_t (could be continuous), we want the network to give us a row of actions from $Q(s,a)$ table that we can choose from:

[… other states…]
→ [$Q(s_t,a_1)$, $Q(s_t,a_2)$, $Q(s_t,a_3)$, … $Q(s_t,a_A)$] ←
[… other states…]

- How to train network to be Q? Make a loss function which incentives the actual Q-function behavior we desire from a sampled tuple (s, a, r, s')

$$\mathscr{L} = \left[Q(s,a) - [r_{s,a} + \gamma \max_{a' \in A} Q^*(s',a')] \right]^2$$

from current network params

from older network params (better stability)

Periodically Update
Params of Q^* from Q

$$\mathscr{L} = \left[Q(s,a) - [r_{s,a}] \right]^2$$

if no next state (env is done)

But we need more power!

- We need to do some **random actions** before following the policy or else we won't learn
- Also, we need to follow the policy more and more during training to get to better places in the environment
- **Epsilon-Greedy** Approach:
    - Start randomly doing actions with prob epsilon
    - Slowly make epsilon smaller as training progresses
- And also we need to have larger amounts of uncorrelated training batches so we will again use **experience replay**
- **Update schedule**: make Q and $Q*$ same every N steps

Deep Q-Learning Reinforcement Learning

M. Lapan Implementation for Frozen Lake and Atari!

$$\mathscr{L} = \left[Q(s,a) - [r_{s,a} + \gamma \max_{a' \in A} Q^*(s',a')] \right]^2$$

from current network params

from older network params (better stability)

$$\mathscr{L} = \left[Q(s,a) - [r_{s,a}] \right]^2$$

if no next state (env is done)

```
Follow Along:
08a_Basics_Of_Reinforcement_Learning.ipynb
```

Course Retrospective

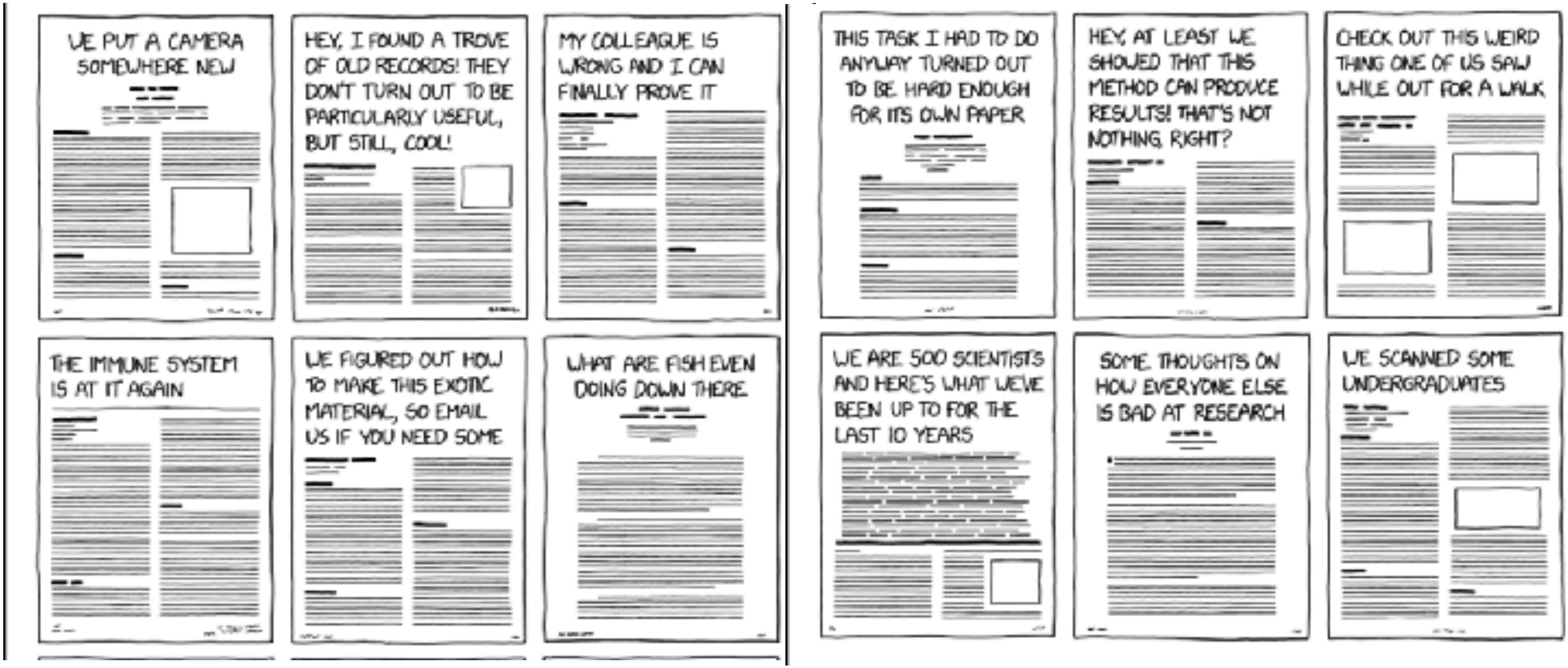
Day 1 of python: How can I learn python ?

Day 3 of python: machine learning engineer positions near me

Course Retrospective

- **Ethics**: The Guidelines, ConceptNet NumberBatch
- **Multi-task and Multi-modal**: ATLAS, Self-consistency
- **CNN Visualization**: Heatmaps, Grad-CAM, Circuits
- **CNN Fully Convolutional**: R-CNN, YOLO, Mask-RCNN, YOLACT and others
- **Style Transfer**: Gatys, FastStyle, WCT
- **GANs**: Vanilla to Wasserstein to BigGAN (and others)
- **RL**: CE, Value Iteration, Q-Learning, Deep Q-Learning
- What was good, **bad**, **ugly**? What could be **changed**?

Types of Scientific Papers



Thanks for a great semester!!!

Please fill out the course evaluations!!

Backup Lectures

World Models

The Problem

World Models

Can agents learn inside of their own dreams?

DAVID HA Google Brain Tokyo, Japan	JÜRGEN SCHMIDHUBER NNAISENSE Swiss AI Lab, IDSIA (USI & SUPSI)	March 27 2018	NIPS 2018 Paper	YouTube Talk	Download PDF

https://worldmodels.github.io

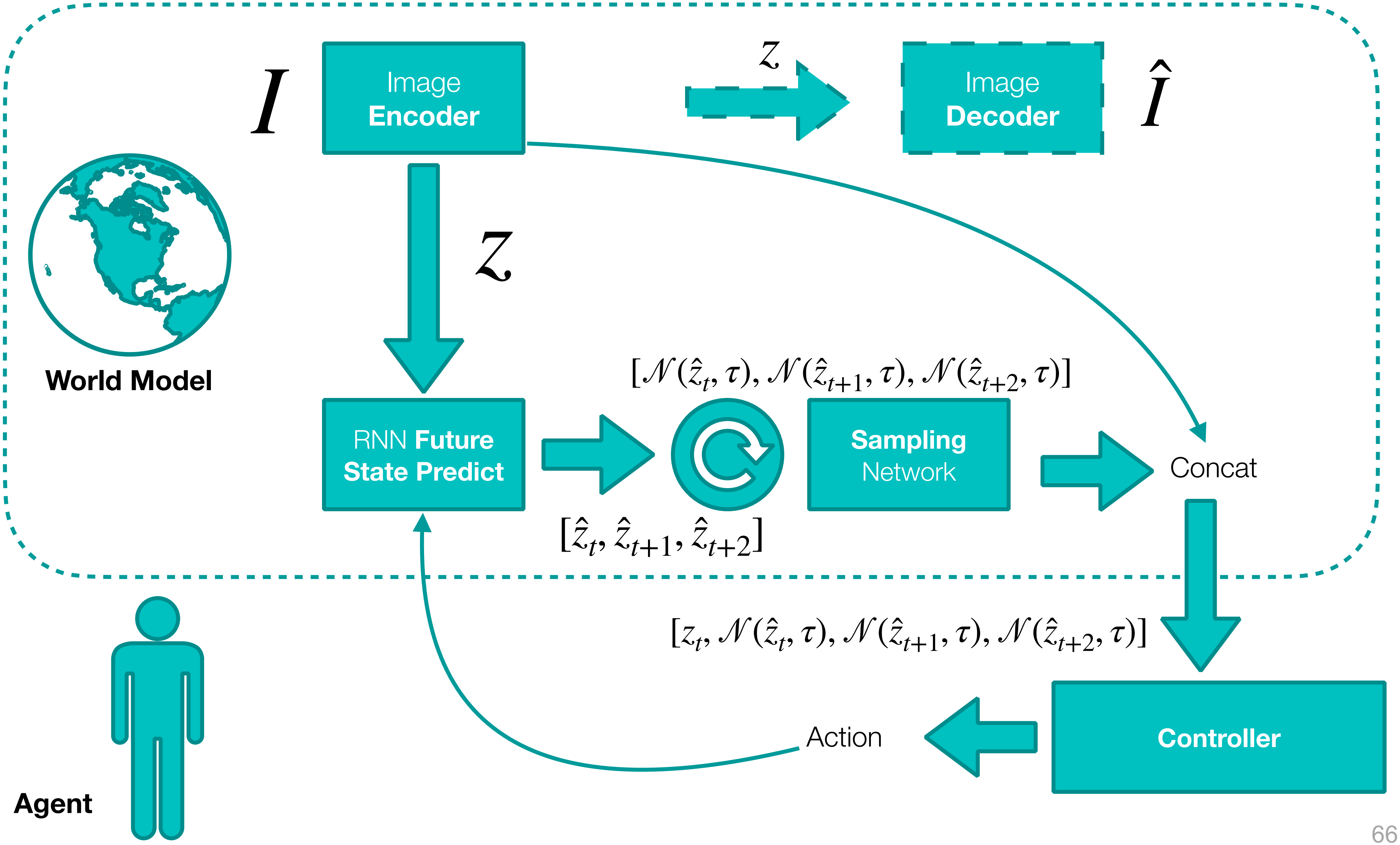
A Motivation

Agents can dream! What a time to be alive!

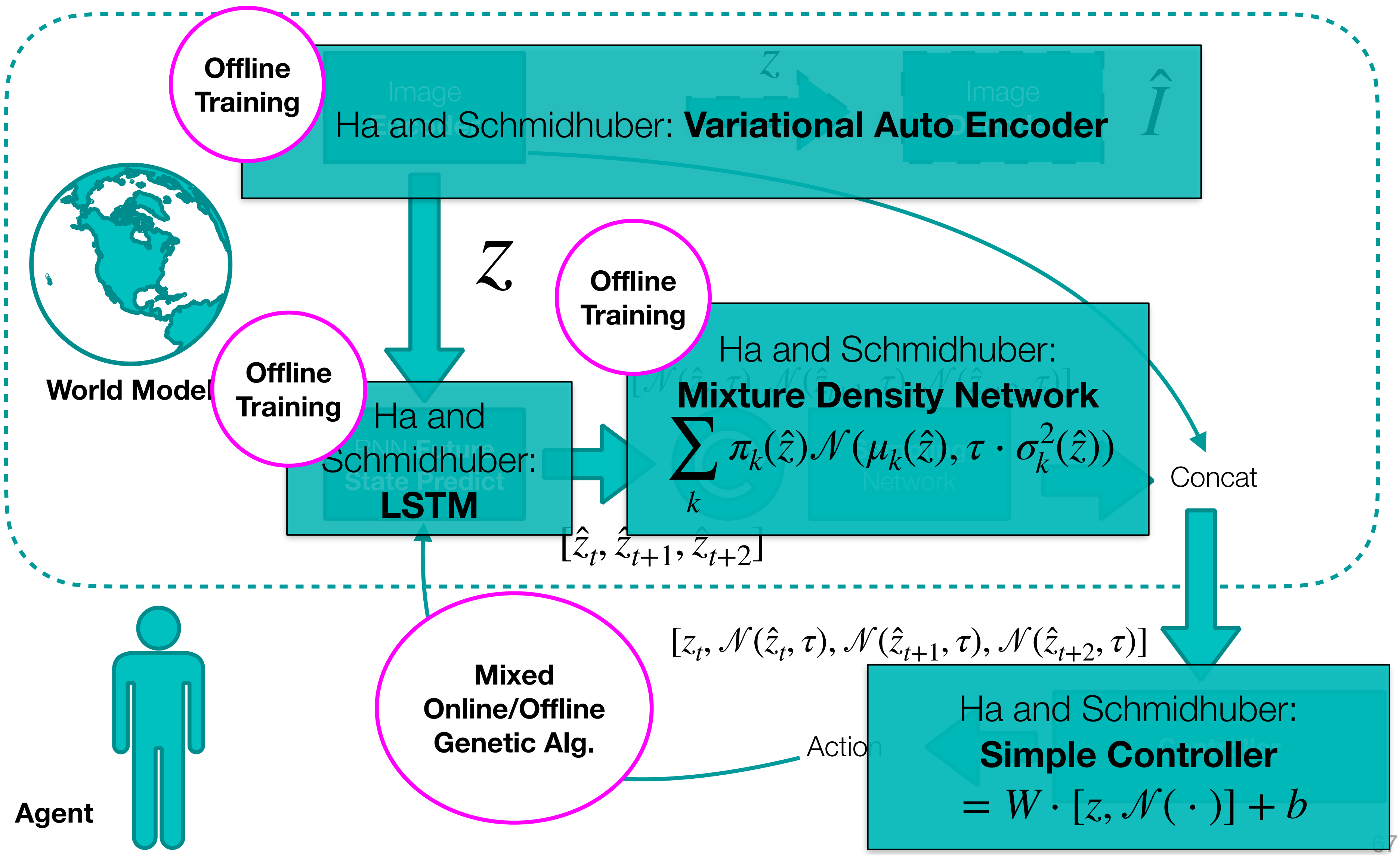
And academia can dream about driving the hype train!

Maybe we should be more careful about the way we describe what an agent does… because they don't dream. That's fluff.

The Main Idea



Implementation

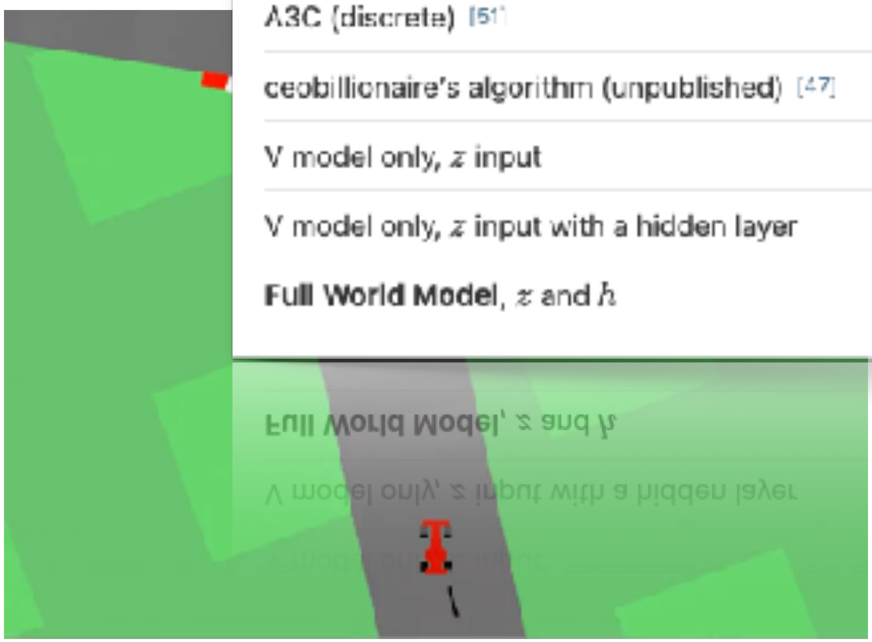


An Example, Racing

- Schmidhuber and Ha Methods:
  - Collect 10,000 rollouts from a random policy.
  - Train VAE (V) to encode each frame
  - Train
  - Evol
    cum

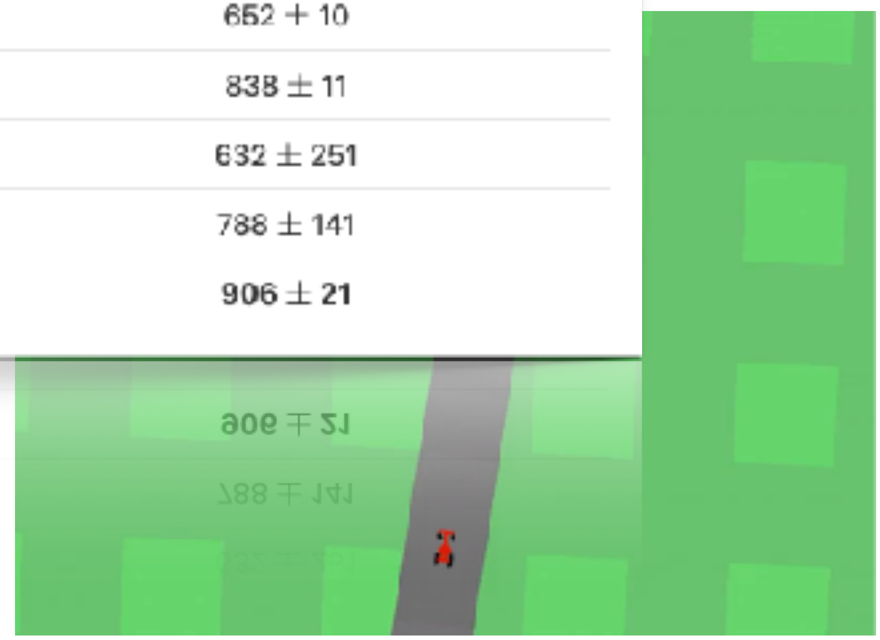
Model	Parameter Count
VAE	4,348,547
	422,368
	867

Method	Average Score over 100 Random Tracks
DQN [53]	343 ± 18
A3C (continuous) [52]	591 ± 45
A3C (discrete) [51]	652 ± 10
ceobillionaire's algorithm (unpublished) [47]	838 ± 11
V model only, z input	632 ± 251
V model only, z input with a hidden layer	788 ± 141
Full World Model, z and h	906 ± 21



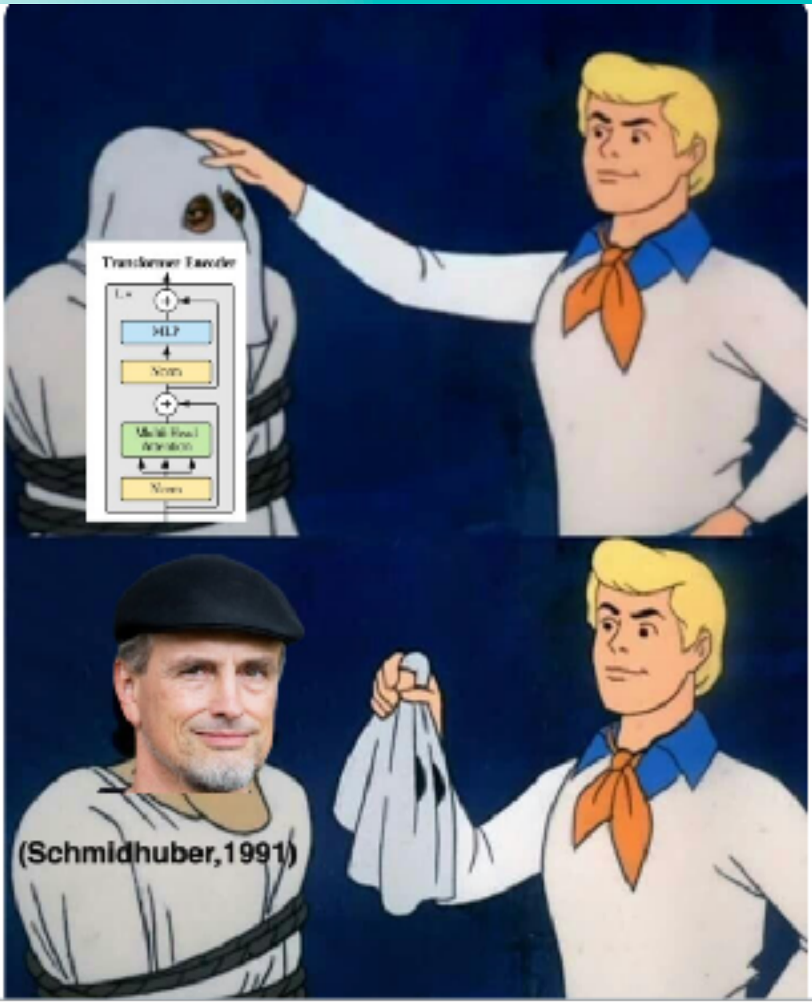
Only use VAE Encoding

https://worldmodels.github.io



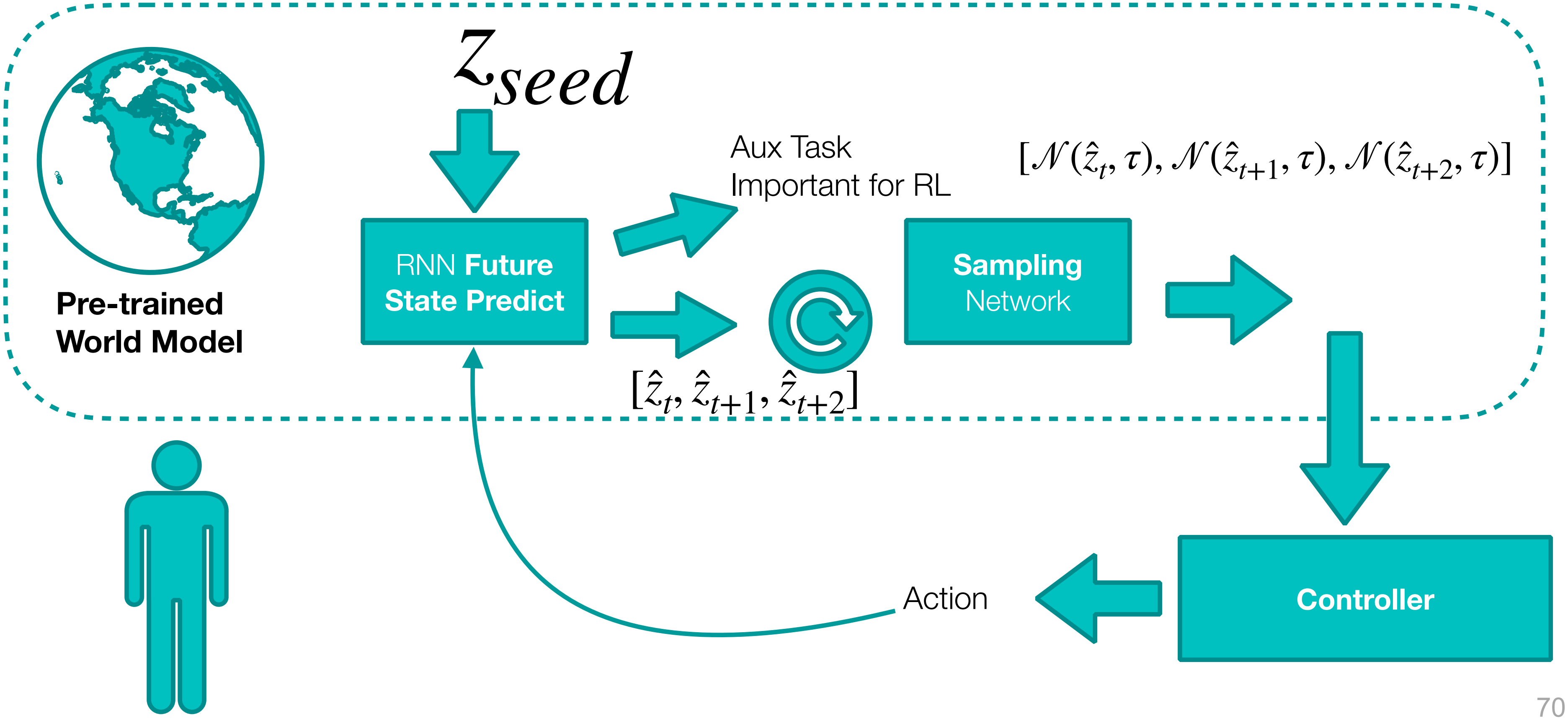
Full World Model

World Models II

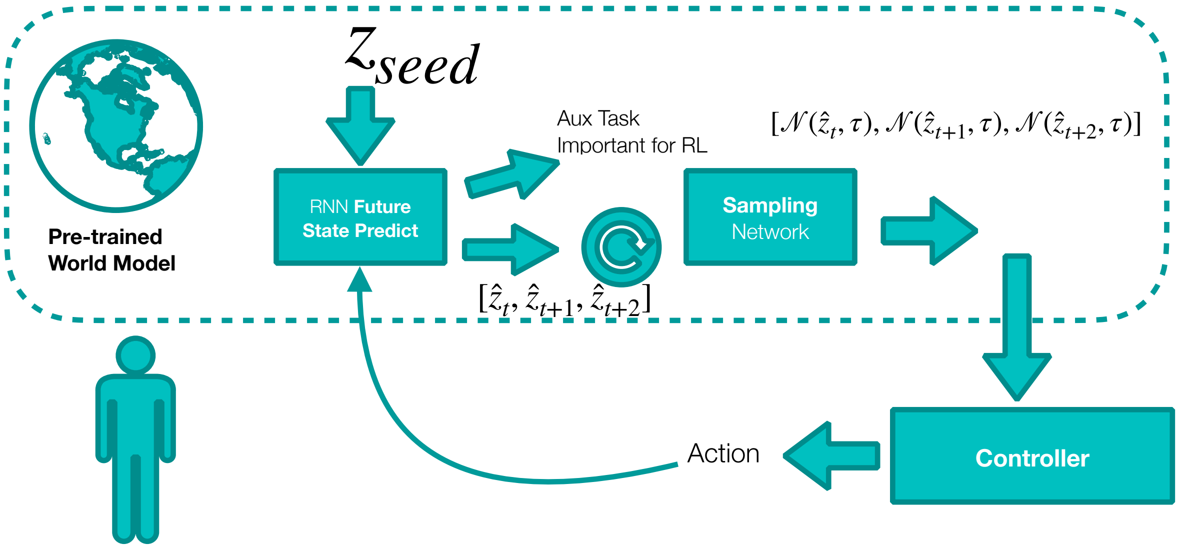


Can we learn without the environment?

- What if we sample from the world model to train our controller?



VizDoom Training Example



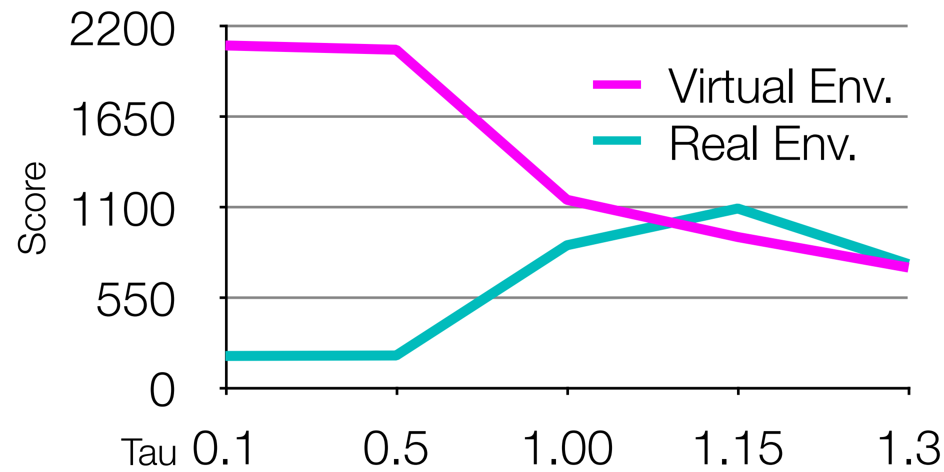
Model	Parameter Count
VAE	4,446,915
MDN-RNN	1,678,785
Controller	1,088

- Collect 10,000 rollouts from a random policy
- Train VAE (V) to encode each frame
- Train MDN-RNN to predict z and “if survived” in next frame
- Evolve Controller (C) to maximize the expected survival time inside the virtual environment.
- Use learned policy from on actual Gym environment
- Call it training inside a “dream” because marketing

Learned Policy

- Important to optimize the temperature control of the MDN

$$\sum_k \pi_k(\hat{z})\mathcal{N}(\mu_k(\hat{z}), \tau \cdot \sigma_k^2(\hat{z}))$$



Temperature	Score in Virtual Environment	Score in Actual Environment
0.10	2086 ± 140	193 ± 58
0.50	2060 ± 277	198 ± 50
1.00	1145 ± 690	868 ± 511
1.15	918 ± 546	1092 ± 556
1.30	732 ± 269	753 ± 139
Random Policy Baseline	N/A	210 ± 108
Gym Leaderboard [34]	N/A	820 ± 58

More Complex Models

- Random Policy makes it hard to exploit "hard to get to" regions of the state space
- Solution: Iterative algorithm
  - Initialize *M*, *C* with random model parameters
  - Rollout to actual environment *N* times. Agent may learn during rollouts. Save all actions and observations during rollouts
  - Train *M* to model $P(x_{t+1}, r_{t+1}, a_{t+1}, d_{t+1} \mid x_t, a_t, \hat{z}_t)$ and train *C* to optimize expected rewards in *M*
  - Repeat rollout of new policy if not converged
- Leave that investigation to future work…

Lecture Notes for

Neural Networks and Machine Learning

World Models and Course Retrospective

Next Time:

None!

Reading: Nope