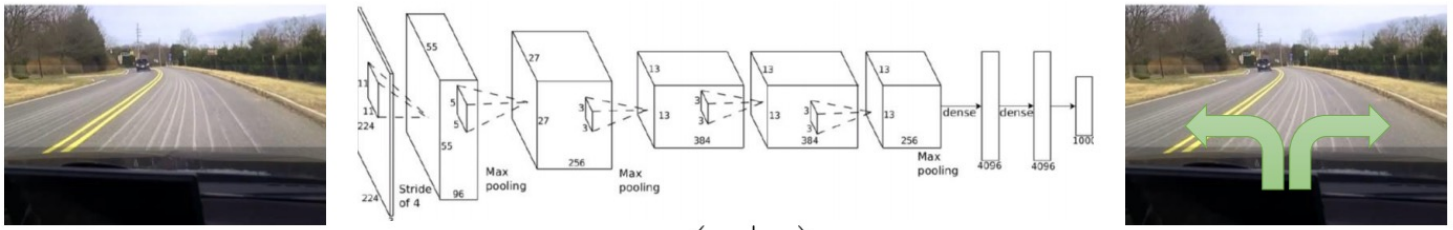
Policy Gradient Methods



Sources: Stanford CS 231n, Berkeley Deep RL course, David Silver's RL course

Image source

Outline

- Stochastic policy representation
- Finding the policy gradient
- REINFORCE algorithm
- Actor-critic algorithms
- Applications

Policy gradient methods: Motivation

- Instead of indirectly representing the policy using Q-values, it can be more efficient to parameterize and learn it directly
  - Especially in large or continuous action spaces

Image source: OpenAI Gym

Stochastic policy representation

- Learn a function giving the probability distribution over actions from the current state:

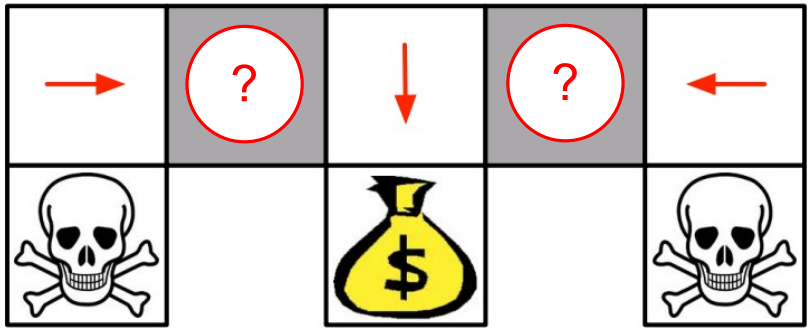
$$\pi_\theta(a|s) \approx P(a|s)$$

Stochastic policy representation

- Learn a function giving the probability distribution over actions from the current state:

$$\pi_\theta(a|s) \approx P(a|s)$$

- Why stochastic policies?
  - There are examples even of grid world scenarios where only a stochastic policy can reach optimality



Source:
D. Silver

The agent can't tell the difference between the gray cells

Stochastic policy representation

- Learn a function giving the probability distribution over actions from the current state:

$$\pi_\theta(a|s) \approx P(a|s)$$

- Why stochastic policies?
  - It's mathematically convenient!
    - Softmax policy:

$$\pi_\theta(a|s) = \frac{\exp(f_\theta(s,a))}{\sum_{a'} \exp(f_\theta(s,a'))}$$

    - Gaussian policy (for continuous action spaces):

$$\pi_\theta(a|s) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(a - f_\theta(s))^2}{2\sigma^2}\right)$$

Expected value of a policy

$$J(\theta) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t \, r_t \mid \pi_\theta\right]$$

$$= \mathbb{E}_\tau[r(\tau)]$$

Expectation of return over *trajectories* $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$

$$= \int_\tau r(\tau) \underbrace{p(\tau; \theta)} d\tau$$

Probability of trajectory τ under policy with parameters θ

Finding the policy gradient

$$J(\theta) = \int_{\tau} r(\tau) p(\tau;\theta) d\tau$$

$$\nabla_\theta J(\theta) = \int_{\tau} r(\tau) \nabla_\theta p(\tau;\theta) d\tau$$

$$= \int_{\tau} r(\tau) p(\tau;\theta) \frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} d\tau$$

$$= \int_{\tau} r(\tau) p(\tau;\theta) \nabla_\theta \log p(\tau;\theta)\, d\tau$$

$$= \mathbb{E}_\tau [r(\tau) \nabla_\theta \log p(\tau;\theta)]$$

Finding the policy gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau[r(\tau)\nabla_\theta \log p(\tau;\theta)]$$

Probability of trajectory

$\tau = (s_0, a_0, s_1, a_1, \dots)$

$$p(\tau;\theta) = P(s_0)\prod_{t\geq 0}\pi_\theta(a_t|s_t)P(s_{t+1}|s_t,a_t)$$

$$\log p(\tau;\theta) = \log P(s_0) + \sum_{t\geq 0}[\log \pi_\theta(a_t|s_t) + \log P(s_{t+1}|s_t,a_t)]$$

$$\nabla_\theta \log p(\tau;\theta) = \sum_{t\geq 0}\nabla_\theta \log \pi_\theta(a_t|s_t)$$

The *score function*

Score function $\nabla_\theta \log \pi_\theta(a|s)$

- For softmax policy:

$$\pi_\theta(a|s) = \frac{\exp(f_\theta(s,a))}{\sum_{a'} \exp(f_\theta(s,a'))}$$

$$\nabla_\theta \log \pi_\theta(a_t|s_t) = \nabla_\theta f_\theta(s,a) - \sum_{a'} \pi_\theta(a'|s\,)\, \nabla_\theta f_\theta(s,a')$$

- For Gaussian policy:

$$\pi_\theta(a|s) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(a - f_\theta(s))^2}{2\sigma^2}\right)$$

$$\nabla_\theta \log \pi_\theta(a_t|s_t) = \frac{\big(a - f_\theta(s)\big)}{\sigma^2} \nabla_\theta f_\theta(s) - \text{const.}$$

Finding the policy gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau[r(\tau)\nabla_\theta \log p(\tau;\theta)]$$

$$\nabla_\theta \log p(\tau;\theta) = \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \left[\underbrace{\left(\sum_{t\geq 0} \gamma^t r_t \right)}_{\text{Return of trajectory } \tau} \underbrace{\left(\sum_{t\geq 0} \nabla_\theta \log \pi_\theta(a_t|s_t) \right)}_{\text{Gradient of log-likelihood of actions under current policy}} \right]$$

- How do we estimate the gradient in practice?

Finding the policy gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau[r(\tau)\nabla_\theta \log p(\tau;\theta)]$$

$$\nabla_\theta \log p(\tau;\theta) = \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \left[\left(\sum_{t\geq 0} \gamma^t r_t \right) \left(\sum_{t\geq 0} \nabla_\theta \log \pi_\theta(a_t|s_t) \right) \right]$$

- Stochastic approximation: sample N trajectories $\tau_1, \ldots, \tau_N$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=0}^{T_i} \gamma^t r_{i,t} \right) \left(\sum_{t=0}^{T_i} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \right)$$

REINFORCE algorithm

1. Sample N trajectories τ_i using current policy π_θ

2. Estimate the policy gradient:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} r(\tau_i)\left(\sum_{t=0}^{T_i} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\right)$$

3. Update parameters by gradient ascent:

$$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$$

Williams et al. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning, 8(3):229-256, 1992

REINFORCE: Single-step version

1. In state s, sample action a using current policy π_θ, observe reward r

2. Estimate the policy gradient:

$$\nabla_\theta J(\theta) \approx r \, \nabla_\theta \log \pi_\theta(a|s)$$

3. Update parameters by gradient ascent:

$$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$$

- What effect does this update have?
  - Push up the probability of good actions, push down probability of bad actions

Outline

- Stochastic policy representation
- Finding the policy gradient
- REINFORCE algorithm
- Actor-critic algorithms

Reducing variance

- Gradient estimate (for a single trajectory):

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

- **General problem**: rewards of sampled trajectories are too noisy and lead to unreliable policy gradients

Reducing variance

- Gradient estimate (for a single trajectory):

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

- First observation: it seems bad to weight each action in a trajectory by the return of the entire trajectory. In particular, rewards obtained *before* an action was taken should not be used to weight that action
  - Instead, for each action, consider only the cumulative *future* reward:

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Reducing variance

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

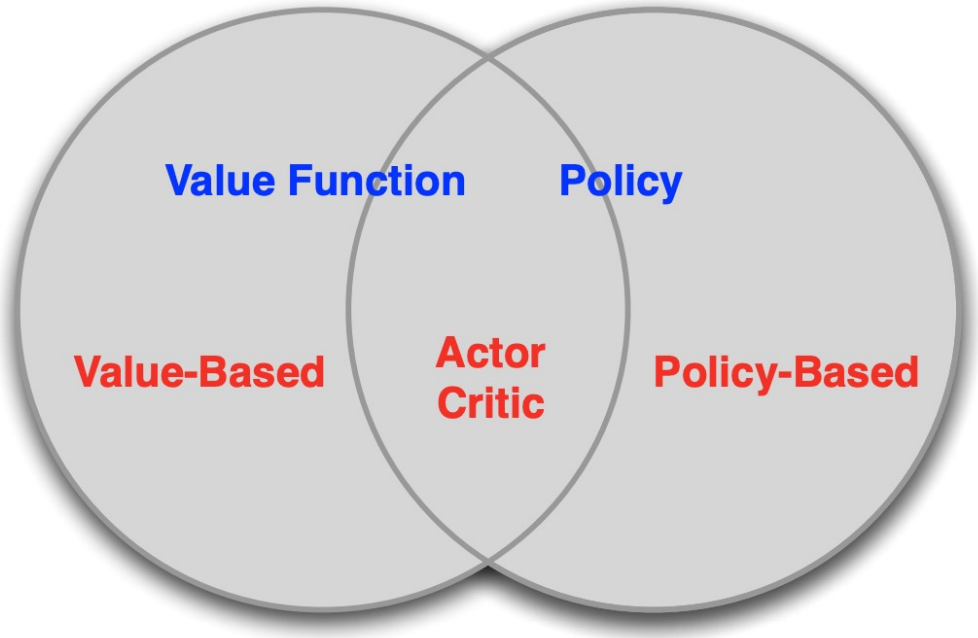
Observed cumulative reward after taking action a_t in state s_t

- But then, why not use *expected* cumulative reward?

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} Q^{\pi_\theta}(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

Actor-Critic algorithms

- Combine policy gradients and Q-learning by simultaneously training an *actor* (the policy) and a *critic* (the Q-function)



Source: D. Silver

Reducing variance

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} Q^{\pi_\theta}(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

- Next observation: the raw Q-values are not the most useful. If all Q-values are good, we will try to push up the probabilities of all the actions
- Instead, compare Q-values of actions to some *baseline function* of the state:

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \underbrace{\left(Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)\right)}_{\text{Advantage function } A^{\pi_\theta}(s_t, a_t)} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Estimating the advantage function

- Advantage function:

$$A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s)$$
$$= \mathbb{E}_{s'}[r + \gamma\, V^{\pi_\theta}(s')|s,a] - V^{\pi_\theta}(s)$$
$$\approx r + \gamma\, V^{\pi_\theta}(s') - V^{\pi_\theta}(s)$$

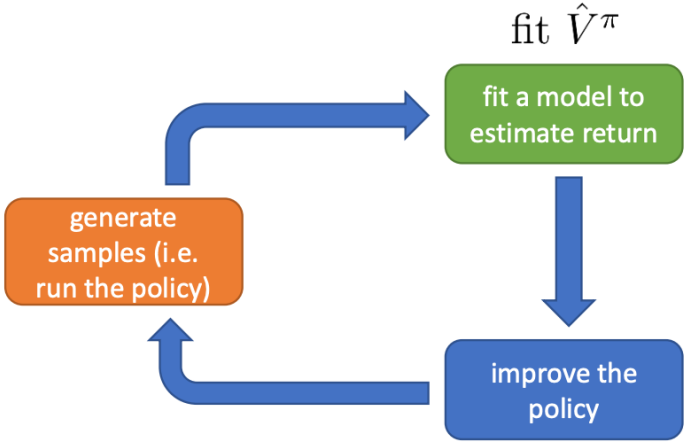
(from a single transition)

- Therefore, it is sufficient to learn the value function:

$$V^{\pi_\theta}(s) \approx V_w(s)$$

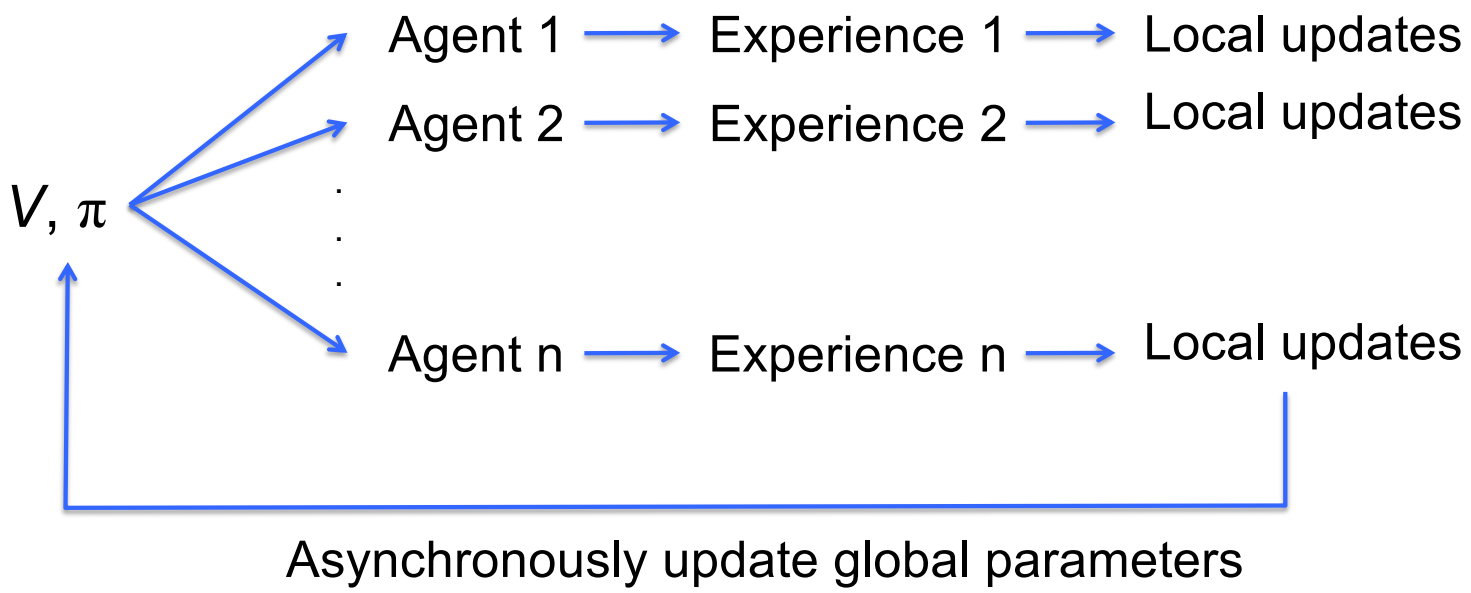
Online actor-critic algorithm

1. Sample action a using current policy, observe reward r, successor state s'
2. Update $V_w(s)$ towards target $r + \gamma V_w(s')$
3. Estimate $A^{\pi_\theta}(s,a) = r + \gamma V_w(s') - V_w(s)$
4. Estimate $\nabla_\theta J(\theta) = A^{\pi_\theta}(s,a) \nabla_\theta \log \pi_\theta(a|s)$
5. Update policy parameters: $\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$



Source: Berkeley RL course

Asynchronous advantage actor-critic (A3C)



Mnih et al. Asynchronous Methods for Deep Reinforcement Learning. ICML 2016

Asynchronous advantage actor-critic (A3C)

Method	Training Time	Mean	Median
DQN	8 days on GPU	121.9%	47.5%
Gorila	4 days, 100 machines	215.2%	71.3%
D-DQN	8 days on GPU	332.9%	110.9%
Dueling D-DQN	8 days on GPU	343.8%	117.1%
Prioritized DQN	8 days on GPU	463.6%	127.6%
A3C, FF	1 day on CPU	344.1%	68.2%
A3C, FF	4 days on CPU	496.8%	116.6%
A3C, LSTM	4 days on CPU	623.0%	112.6%

Mean and median human-normalized scores over 57 Atari games

Mnih et al. Asynchronous Methods for Deep Reinforcement Learning. ICML 2016

Proximal policy optimization (PPO)

- Standard actor-critic algorithms are *on-policy*, cannot perform multiple update steps using the same set of samples
- PPO enables multiple updates using the same minibatch of samples by using a complicated "surrogate objective" that carefully restricts how much the policy can change each time

Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
  for actor=$1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

J. Schulman et al. Proximal Policy Optimization Algorithms. arXiv 2017

Outline

- Stochastic policy representation
- Finding the policy gradient
- REINFORCE algorithm
- Actor-critic algorithms
- Applications

Application: Adaptive policies for robot locomotion

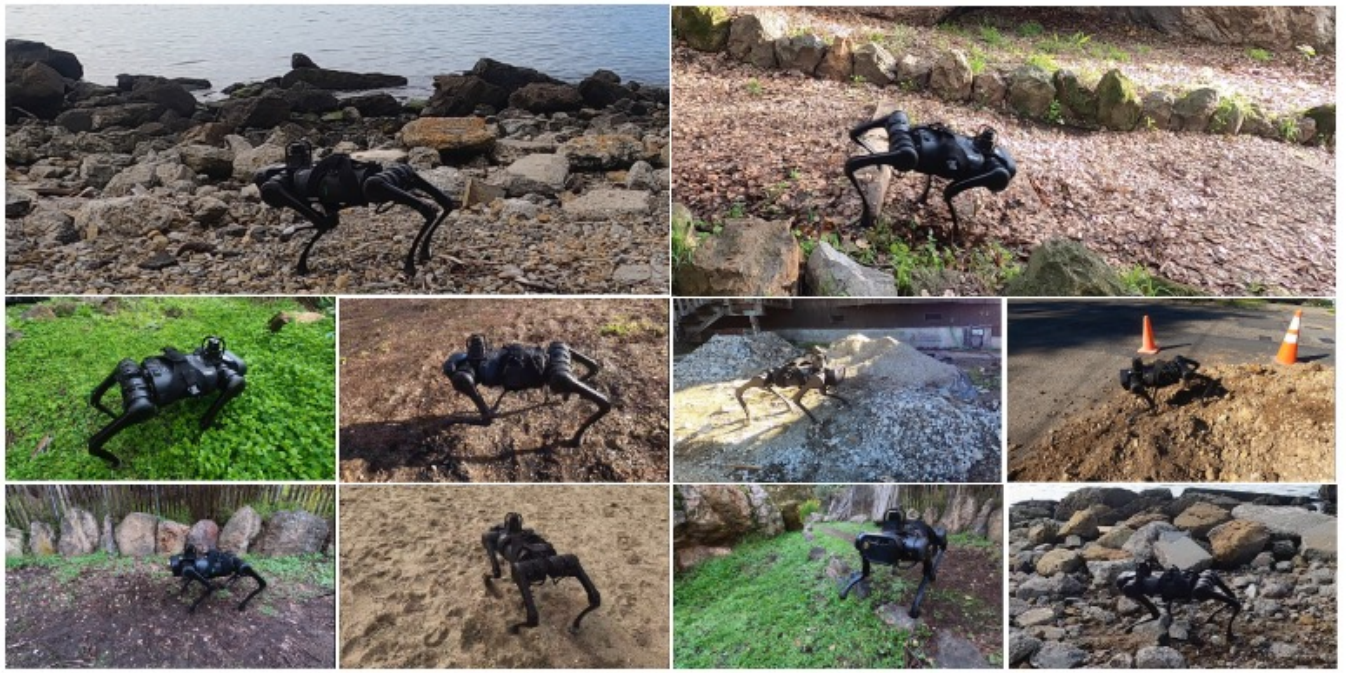
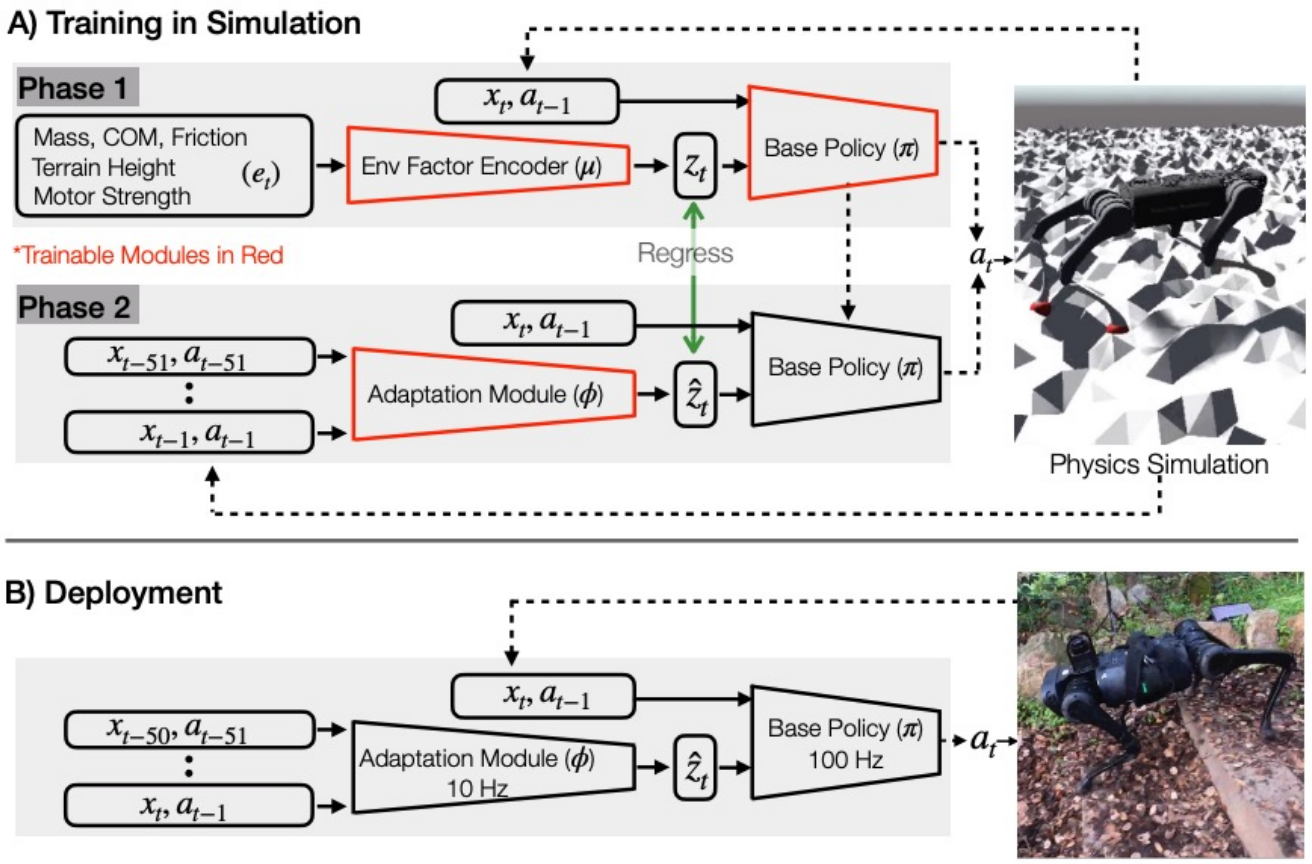


Fig. 1: We demonstrate the performance of RMA on several challenging environments. The robot is successfully able to walk on sand, mud, hiking trails, tall grass and dirt pile without a single failure in all our trials. The robot was successful in 70% of the trials when walking down stairs along a hiking trail, and succeeded in 80% of the trials when walking across a cement pile and a pile of pebbles. The robot achieves this high success rate despite never having seen unstable or sinking ground, obstructive vegetation or stairs during training. All deployment results are with the same policy without any simulation calibration, or real-world fine-tuning. Videos at https://ashish-kmr.github.io/rma-legged-robots/

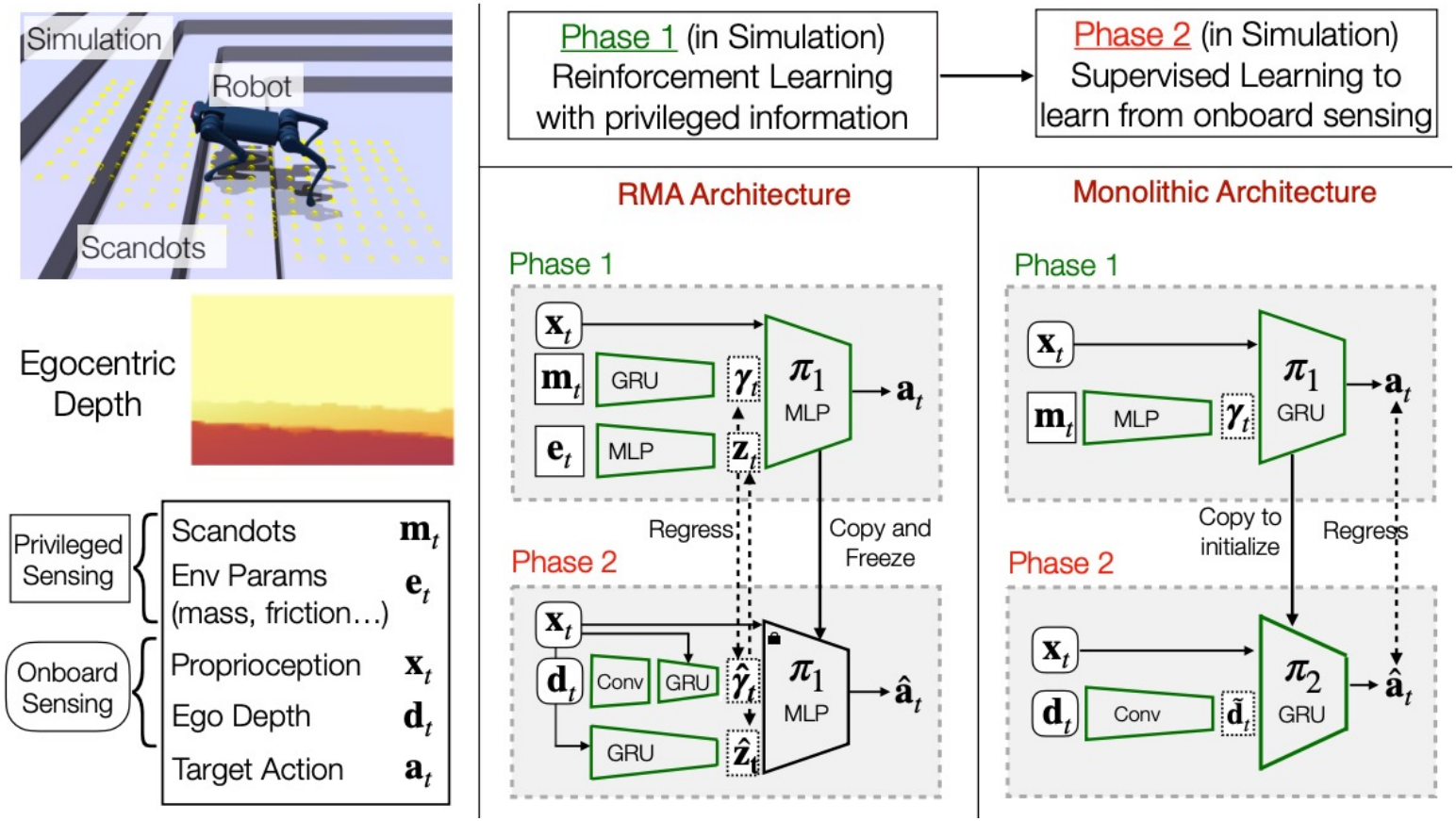
A. Kumar et al. RMA: Rapid Motor Adaptation for Legged Robots. RSS 2021
Method video

Application: Adaptive policies for robot locomotion



A. Kumar et al. RMA: Rapid Motor Adaptation for Legged Robots. RSS 2021
Method video

Locomotion with vision in the loop



A. Agarwal et al. Legged Locomotion in Challenging Terrains using Egocentric Vision. CoRL 2022

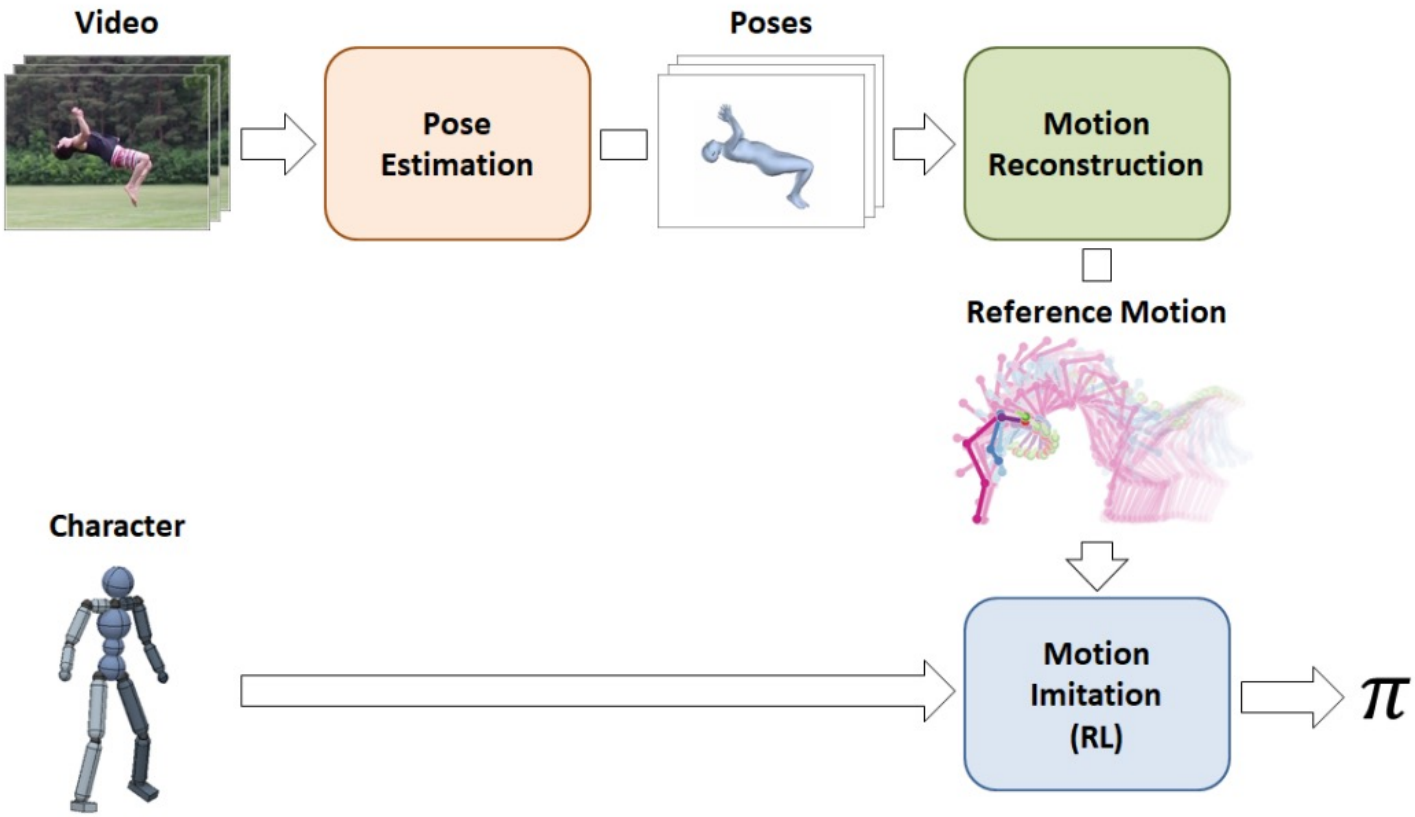
Application: Learning skills from video

Fig. 1. Simulated characters performing highly dynamic skills learned by imitating video clips of human demonstrations. **Left:** Humanoid performing cartwheel B on irregular terrain. **Right:** Backflip A retargeted to a simulated Atlas robot.

Video

X. B. Peng et al. SFV: Reinforcement Learning of Physical Skills from Videos. SIGGRAPH Asia 2018

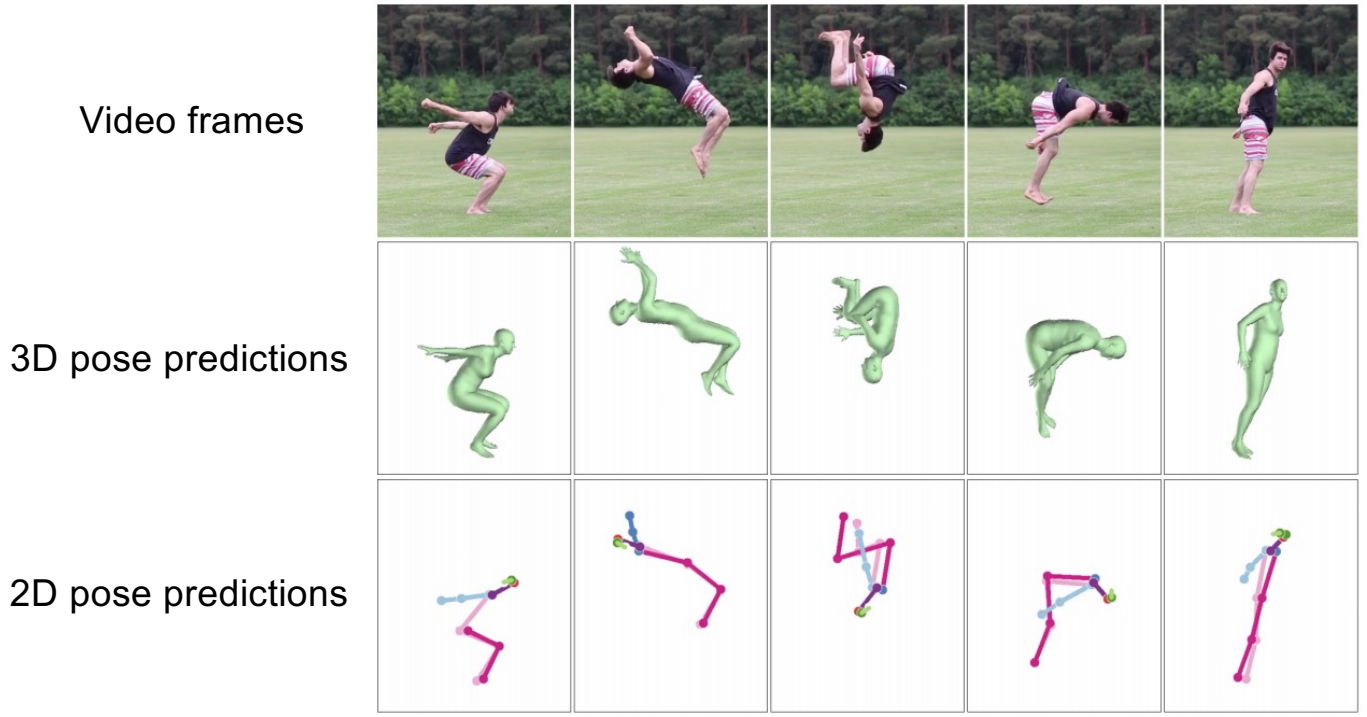
Application: Learning skills from video



X. B. Peng et al. SFV: Reinforcement Learning of Physical Skills from Videos. SIGGRAPH Asia 2018

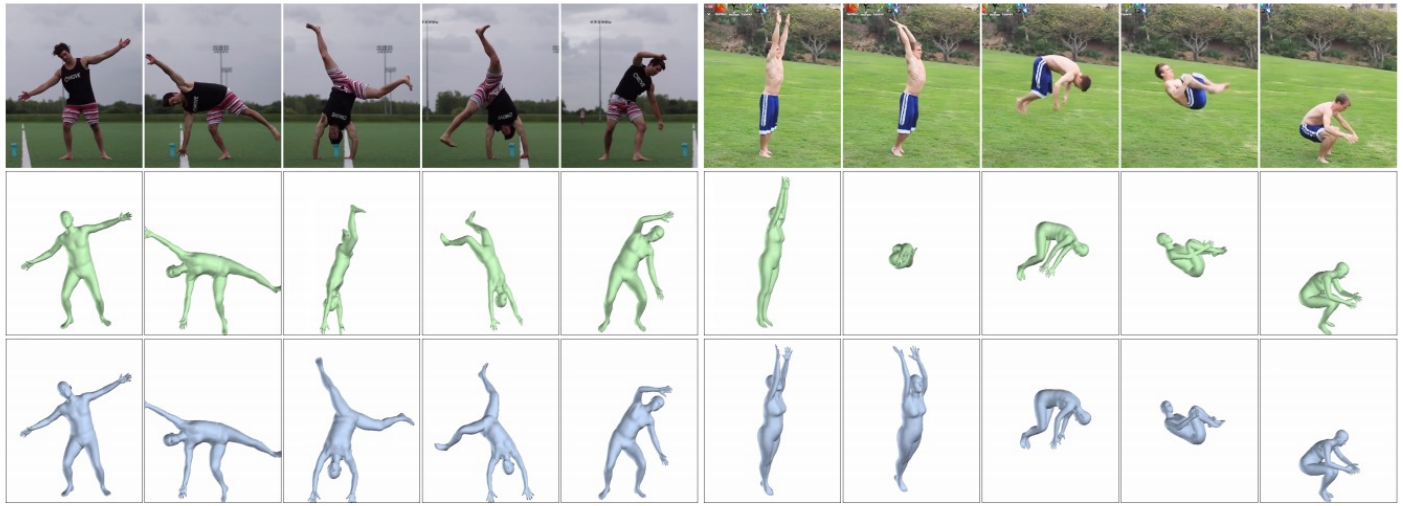
Skills from video: Motion estimation details

- In each video frame, use both OpenPose 2D pose estimator and Human Mesh Recovery (HMR) 3D pose estimator



Skills from video: Motion estimation details

- In each video frame, use both OpenPose 2D pose estimator and Human Mesh Recovery (HMR) 3D pose estimator
- Motion reconstruction: optimize trajectory consistent with per-frame 2D and 3D predictions while maintaining smoothness

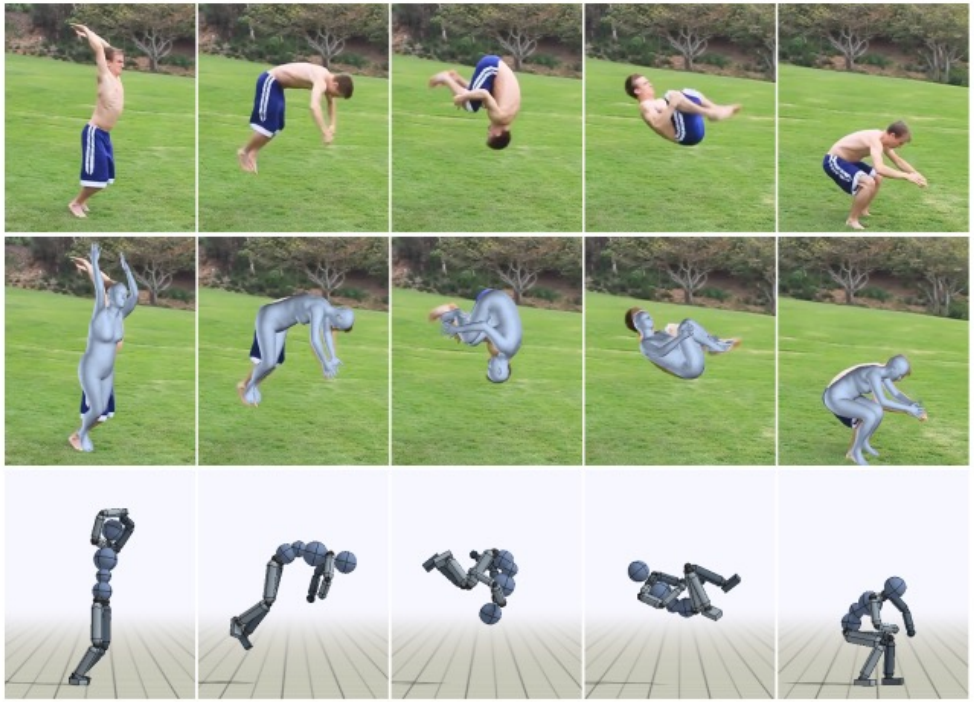


(a) Cartwheel A (b) Frontflip

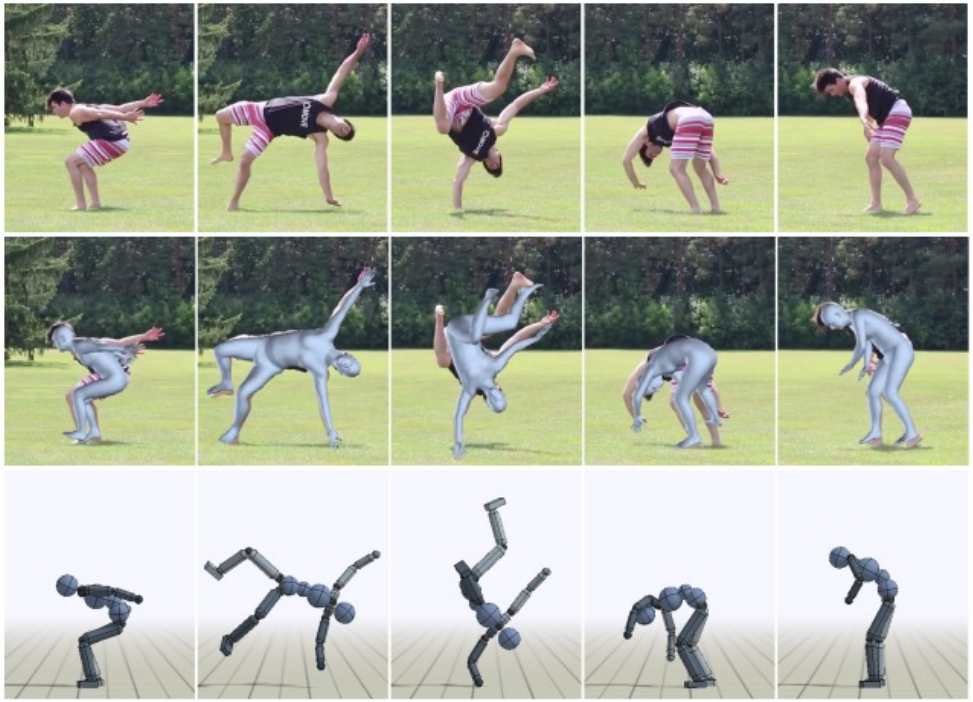
Fig. 11. 3D pose predictions before and after motion reconstruction. **Top:** Video. **Middle:** Raw predictions from the 3D pose estimator before motion reconstruction. **Bottom:** After motion reconstruction. The motion reconstruction process is able to fix erroneous predictions from the 3D pose estimator by taking advantage of the information from the 2D pose estimator and temporal consistency between adjacent frames.

Skills from video: Motion imitation details

- Find policy to encourage simulated motion to reproduce reference motion



(a) Frontflip



(b) Handspring A

Skills from video: Motion imitation details

- **States** (body configurations): position of each body segment relative to the root, rotations, linear and angular velocities, *phase* during motion (0 at start, 1 at end)
- **Actions:** target rotations for each joint (36 dimensions)
- **Transition model** or **dynamics**: derived from physics-based simulation
- **Imitation reward function**: sum of pose, velocity, end-effector, and center-of-mass error terms
- **Finding the policy**: proximal policy optimization
  - Requires policy network (two FC layers, outputs represent parameters of Gaussian distributions over actions), value network
  - Adaptive state initialization: learn initial state distribution as another policy

Skills from video: Results

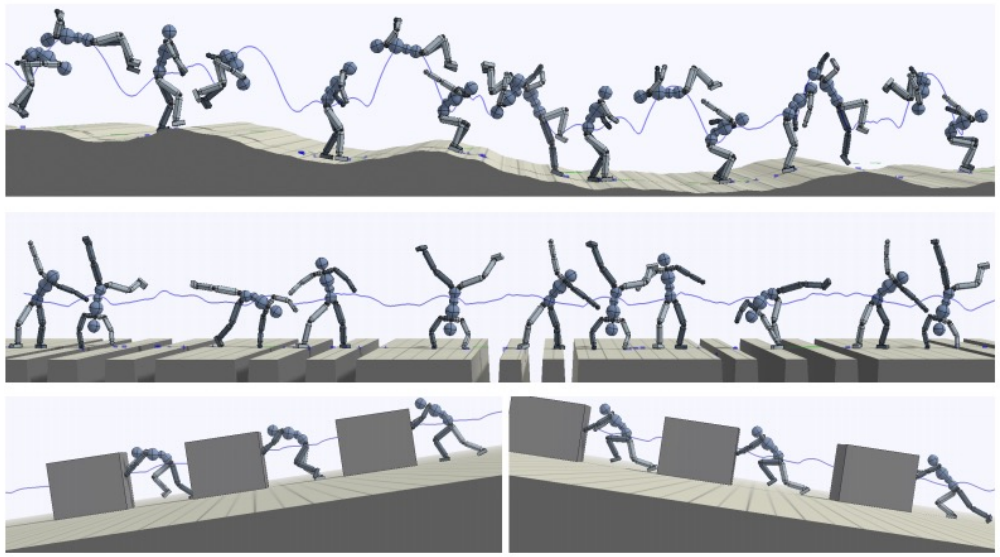


Fig. 8. Skills retargeted to different environments. **Top-to-Bottom:** Backflip A across slopes, cartwheel B across gaps, pushing a box downhill and uphill.

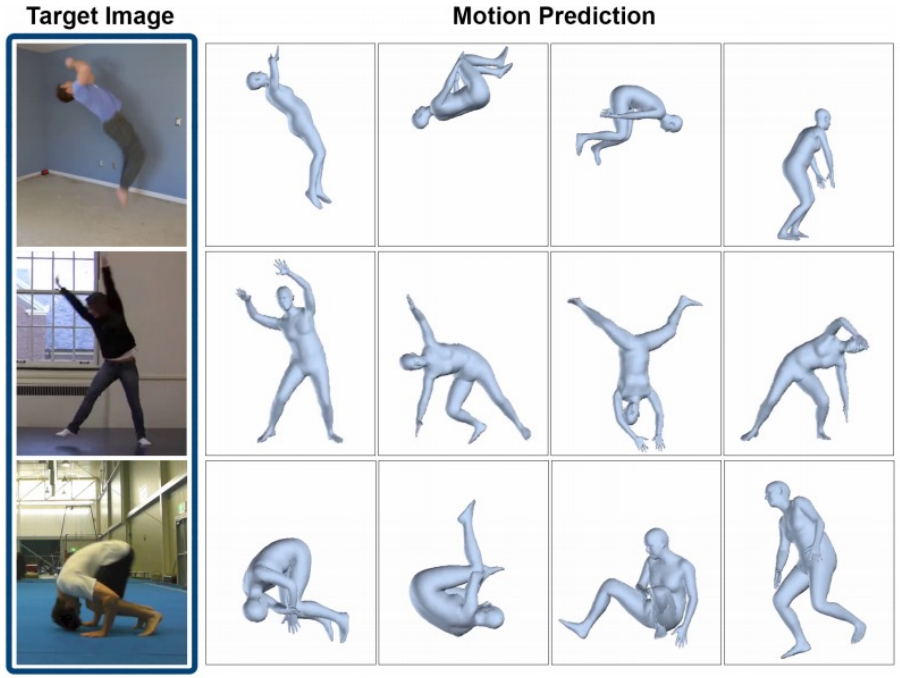


Fig. 14. Given a target image, our motion completion technique predicts plausible motions for the actor in the image.

Video