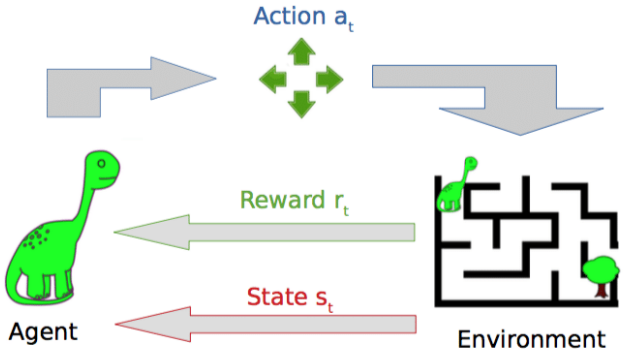
Asynchronous Methods for Deep Reinforcement Learning

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves,…



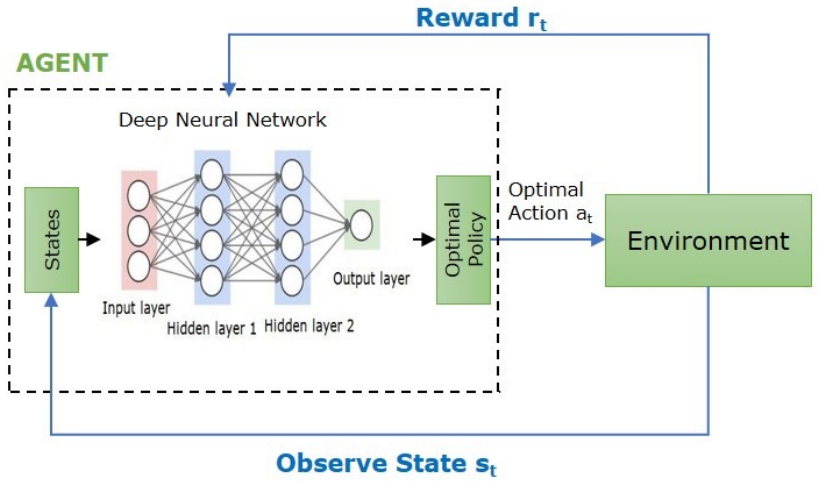
Presentation: Mobina Tavangarifard

Motivation

Why a New RL Method Was Needed?

➢ **DQN revolutionized RL but had limitations:**

- **Required huge replay memory**
- **Needed GPUs and long training**
- **Prone to instability**

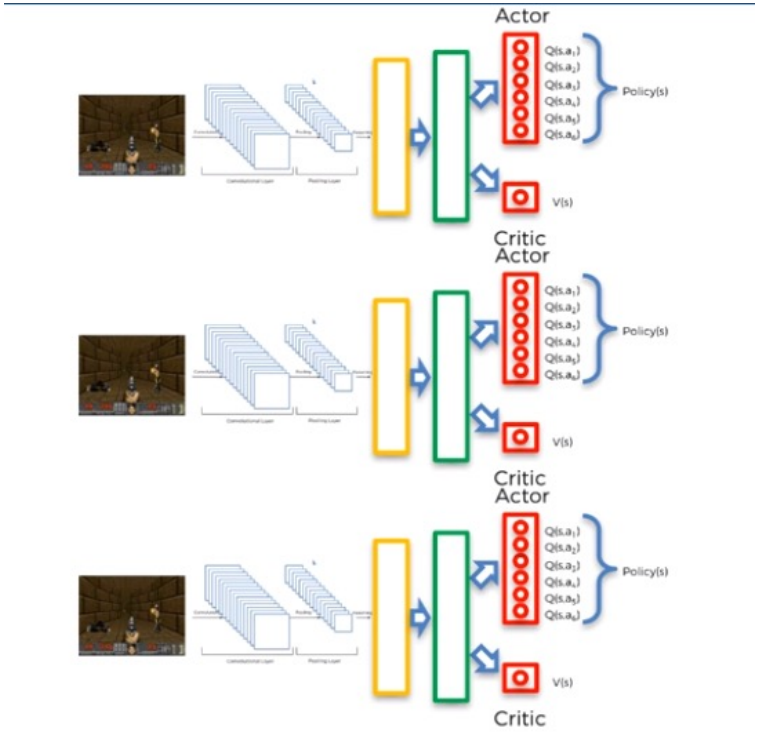


Goal: stable, faster learning using only CPUs

Key Idea

Core Idea of A3C

- Multiple agents run in parallel
- Each explores its own environment copy
- Updates global shared network asynchronously
- Naturally decorrelates experience, no replay buffer



Reinforcement Learning Basics

State → Action → Reward → Next State

- Goal: **Maximize** expected discounted reward

$$R_t = \sum \gamma^k r_{t+k}$$

- Policy gradient updates

$$\nabla_\theta J = \mathbb{E}[\nabla_\theta \log \pi(a|s)(R - b)]$$

b is a **baseline** (a learned function of the state) subtracted to reduce variance; in practice it's usually the value function

Related Work

- **Prior Approaches**

  - **DQN**: Replay buffer, GPU-heavy
  - **Gorila DQN**: Distributed, 100 machines
  - **DistBelief**: Async SGD for supervised nets

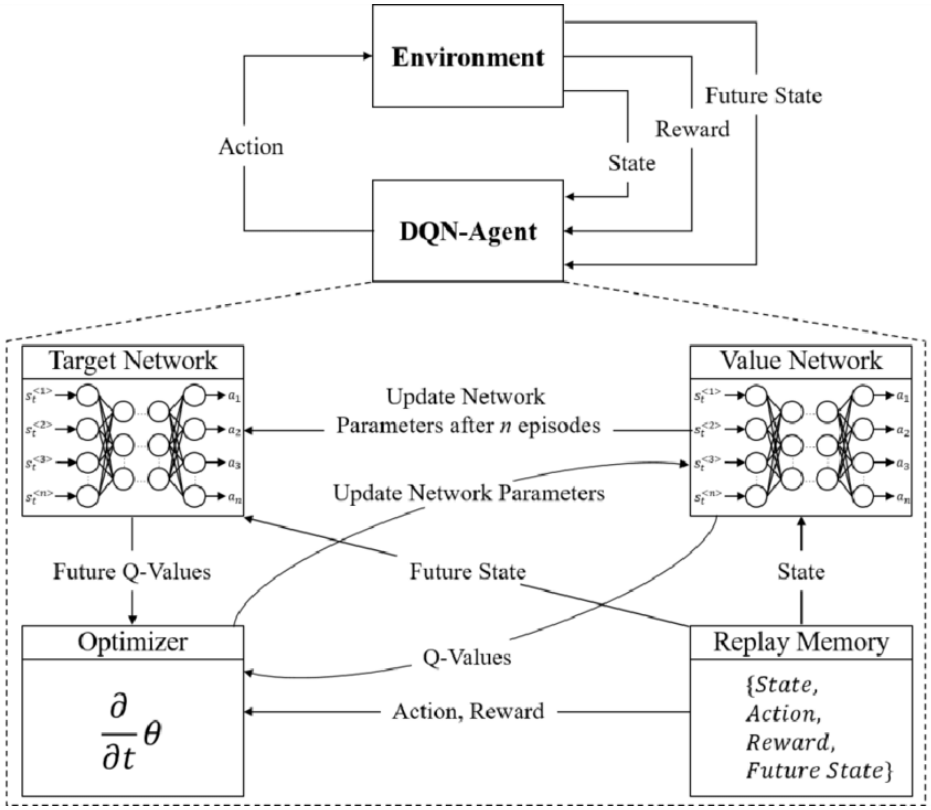
Method	Replay Buffer	Hardware	Stability
DQN	Yes	GPU	Medium
Gorila	Yes	Cluster	High cost
A3C	No	CPU	High

A3C combines distributed learning with RL

Limitations of DQN

Why DQN Needed Help?

- Sequential data → **correlated** samples
- **Replay memory** = huge storage
- Off-policy → **outdated** data
- **Unstable** updates without careful tuning



The Asynchronous Framework

Multiple actor-learners

- Local copies of network parameters
- Gradients pushed to global model **asynchronously**
- Lock-free "**Hogwild**" updates

Overview of Algorithms

Four Algorithms in the Framework

1. One-step Q-learning
2. One-step SARSA
3. n-step Q-learning
4. **A3C (Actor–Critic)** – the best performer

Algorithm Flow

A3C Training Loop

1. Actor runs for t_{max} steps
2. Collects (s, a, r) trajectory
3. Computes multi-step return R
4. Updates shared parameters asynchronously
5. Syncs local copy

Example: Asynchronous RL Framework

Algorithm 1 Asynchronous one-step Q-learning - pseudocode for each actor-learner thread.

// Assume global shared θ, θ^-, and counter $T = 0$.
Initialize thread step counter $t \leftarrow 0$
Initialize target network weights $\theta^- \leftarrow \theta$
Initialize network gradients $d\theta \leftarrow 0$
Get initial state s
repeat
  Take action a with ϵ-greedy policy based on $Q(s, a; \theta)$
  Receive new state s' and reward r
  $$y = \begin{cases} r & \text{for terminal } s' \\ r + \gamma \max_{a'} Q(s', a'; \theta^-) & \text{for non-terminal } s' \end{cases}$$
  Accumulate gradients wrt θ: $d\theta \leftarrow d\theta + \frac{\partial (y - Q(s,a;\theta))^2}{\partial \theta}$
  $s = s'$
  $T \leftarrow T + 1$ and $t \leftarrow t + 1$
  if $T \mod I_{target} == 0$ **then**
    Update the target network $\theta^- \leftarrow \theta$
  end if
  if $t \mod I_{AsyncUpdate} == 0$ or s is terminal **then**
    Perform asynchronous update of θ using $d\theta$.
    Clear gradients $d\theta \leftarrow 0$.
  end if
until $T > T_{max}$

Actor–Critic Architecture

Policy + Value Networks

Shared convolutional layers

- Two output heads:
  - **Actor:** π(a|s)
  - **Critic:** V(s)

- Advantage:

$$A(s, a) = R - V(s)$$

Loss Function

Combined Objective Function

$$L = L_{policy} + c_v L_{value} + c_e L_{entropy}$$

- **Policy term**: encourages good actions
- **Value term**: improves critic accuracy
- **Entropy term**: keeps exploration alive

where

$$L_{\text{policy}} = -\log \pi(a_t|s_t)(R_t - V(s_t)),$$
$$L_{\text{value}} = (R_t - V(s_t))^2,$$
$$L_{\text{entropy}} = -\beta \sum_a \pi(a|s_t) \log \pi(a|s_t).$$

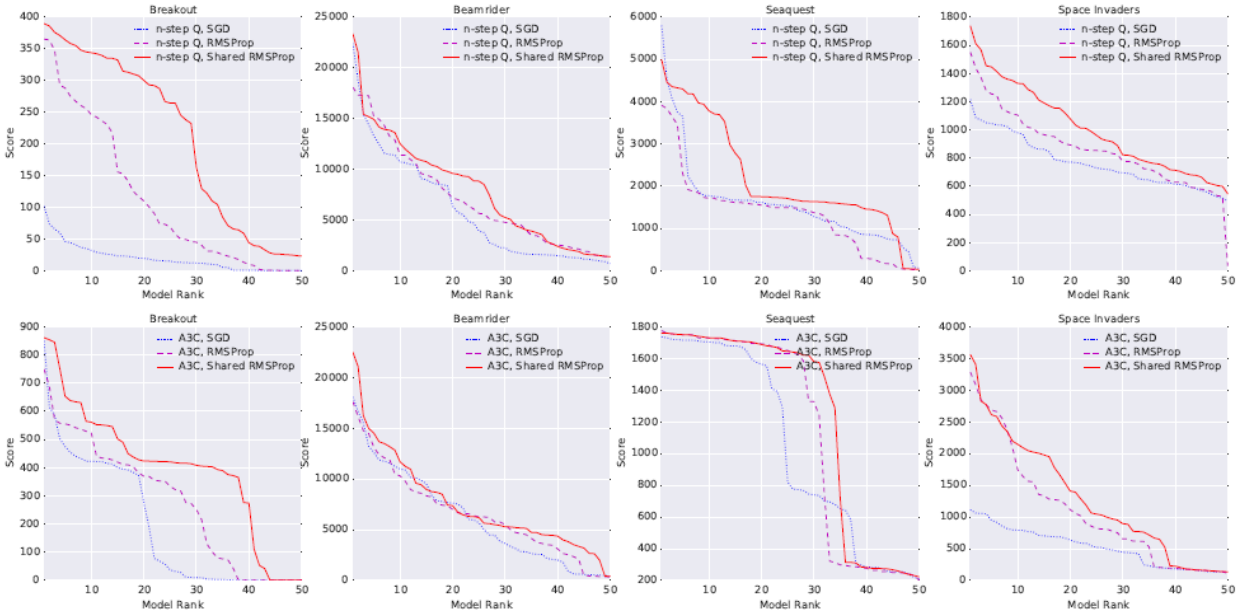
β → **0.01**

Optimization Setup

Shared RMSProp and Hogwild!

$$g \leftarrow \alpha g + (1-\alpha)(\nabla_\theta L)^2, \quad \theta \leftarrow \theta - \eta \frac{\nabla_\theta L}{\sqrt{g+\epsilon}},$$

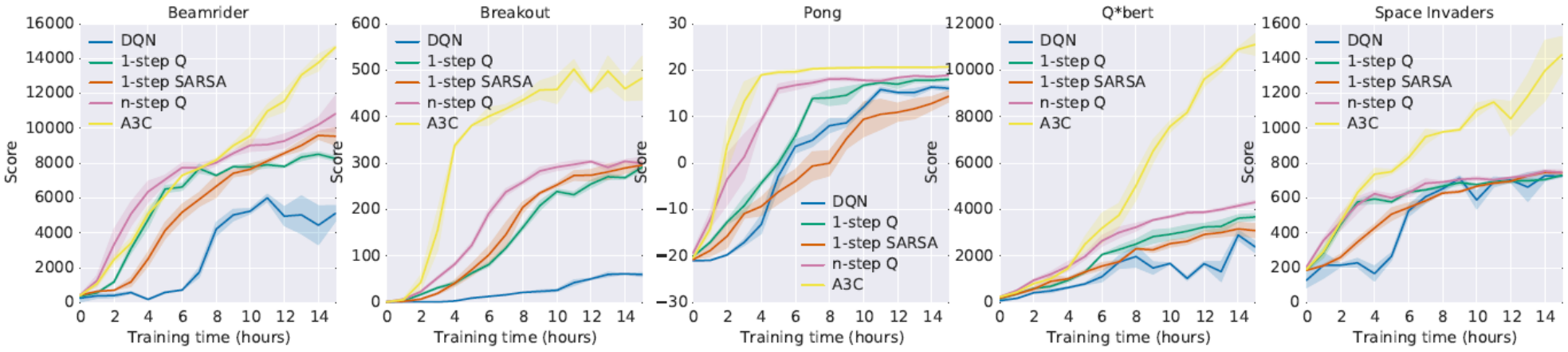
- RMSProp decay = 0.99, ε = 1e–8
- Shared statistics across threads
- Gradient clipping = 40
- t_{max} = 5 steps per rollout
- 16 CPU threads



Environments

Tested Environments

- Atari 2600 games
- MuJoCo physics simulator
- TORCS car racing
- Labyrinth 3D navigation



Atari Results

Atari 2600 Performance

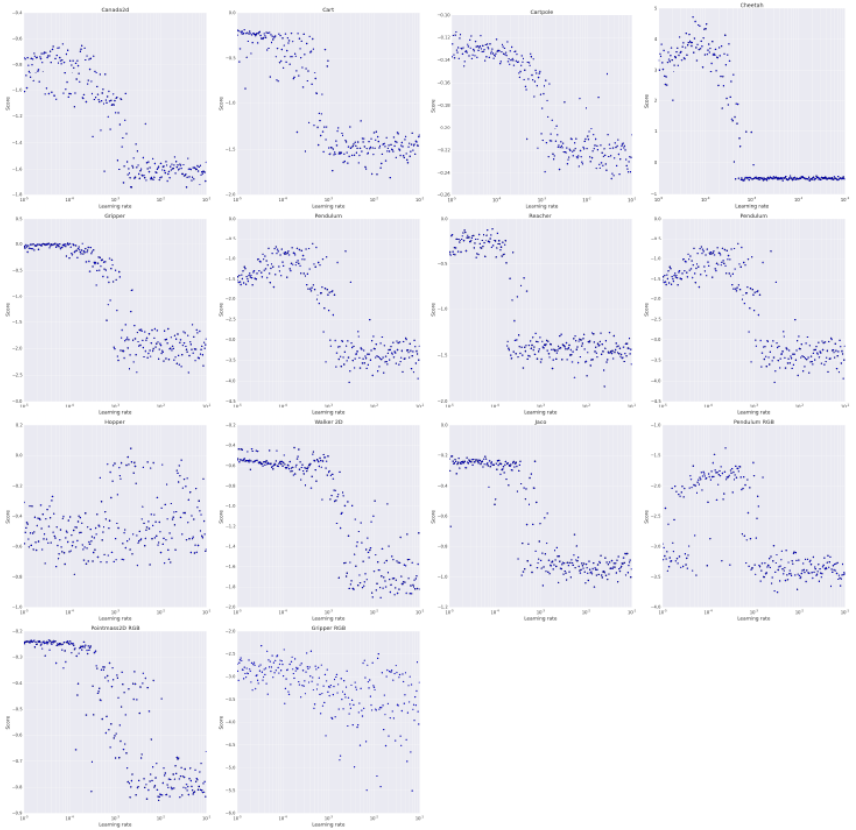
- **10× faster** than DQN on CPU
- LSTM variant doubled performance
- Mean score **623** (vs DQN 121)
- Stable across hyperparameters

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%

MuJoCo Results

MuJoCo Performance

- Gaussian policy output
- Normalized returns per thread
- Smooth torque control in HalfCheetah, Walker2d
- Comparable to DDPG, but **CPU-only**



3D Navigation

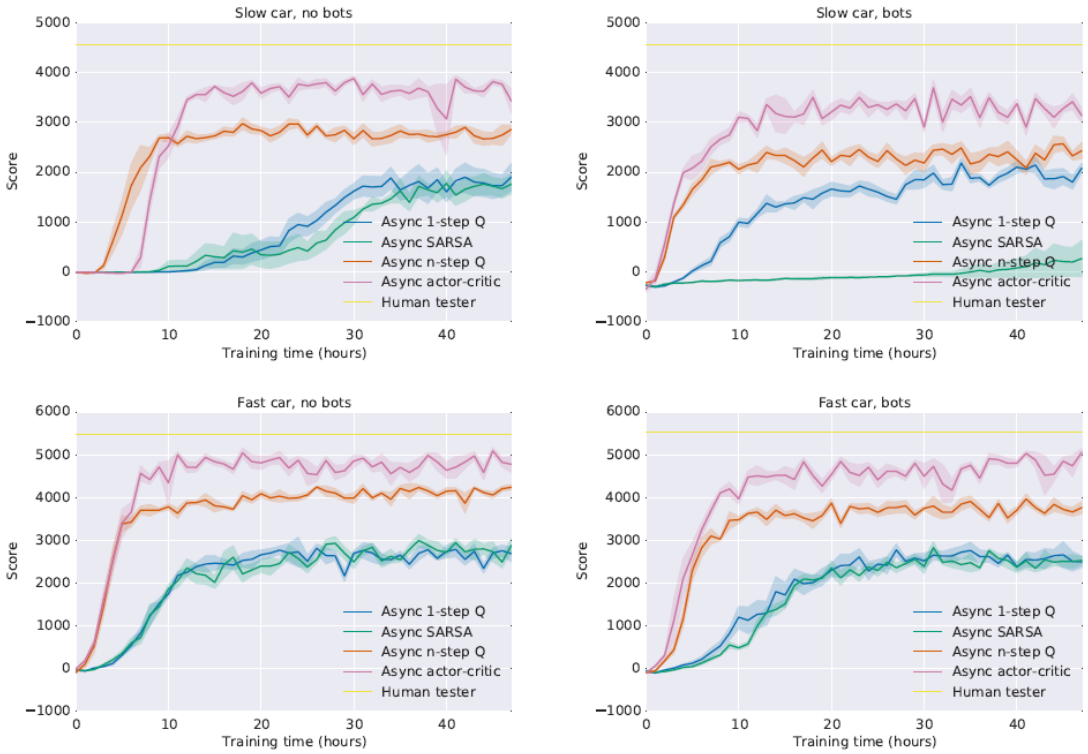
Labyrinth 3D Tasks

- Partial observability
- LSTM version solves mazes
- Sparse rewards handled by entropy regularization

TORCS Racing

Driving from Vision

- RGB camera input
- Steering, throttle, brake actions
- 75–90% of human performance
- Input noise → better generalization

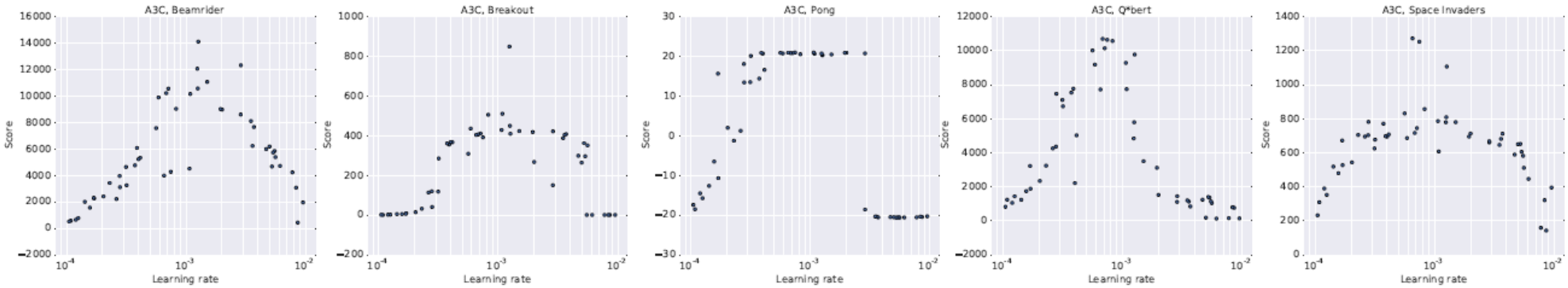


Ablation Studies

What Really Matters?

- Shared RMSProp → **smoother learning**
- β = 0.005–0.02 best for exploration
- t_{max} = 5 optimal
- **16 threads** ≈ best speed/stability tradeoff

	Number of threads				
Method	1	2	4	8	16
1-step Q	1.0	**3.0**	**6.3**	**13.3**	**24.1**
1-step SARSA	1.0	**2.8**	**5.9**	**13.1**	**22.1**
n-step Q	1.0	**2.7**	**5.9**	**10.7**	**17.2**
A3C	1.0	2.1	3.7	6.9	12.5



Scalability

Linear Speedup from Threads

- Speed scales almost linearly up to 16 threads
- Gradient noise across threads cancels out
- Implicit variance reduction

Practical Setup

- C++/Lua + OpenMP threads
- No GPU used
- Shared memory for gradients
- Linear LR decay after 10M steps
- LSTM hidden = 256

Limitations

Weaknesses of A3C

- No replay → **less sample efficient**
- Slightly non-reproducible (async noise)
- Still needs tuning (β, LR)
- **Diminishing** returns after 32 threads

Evolution

Algorithms Inspired by A3C

- **A2C:** synchronous version
- **PPO:** clipped objective for stability
- **IMPALA:** scalable distributed RL
- **AlphaStar / AlphaZero:** inherited async policy–value design

Key Takeaways

What We Learned?

1. Asynchronous parallelism = stability + speed
2. Replay memory can be replaced by decorrelated agents
3. A3C handles all RL regimes
4. Foundation for modern RL (PPO, IMPALA, etc.)

Broader Impact

Why A3C Mattered?

- First **scalable CPU-based** deep RL system
- **Unified continuous** and discrete control
- Inspired all **subsequent distributed RL** frameworks

Final Thoughts

- Simple yet revolutionary idea
- Merged learning stability with hardware scalability
- Remains a cornerstone of reinforcement learning research

Thank you