Agent57: Surpassing humans on Atari games

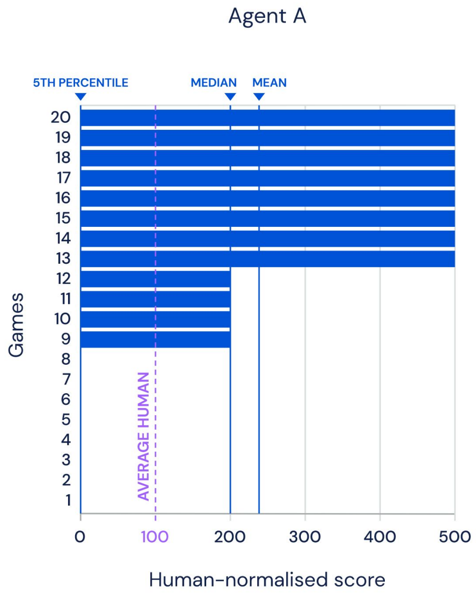
Northwestern | McCORMICK SCHOOL OF ENGINEERING

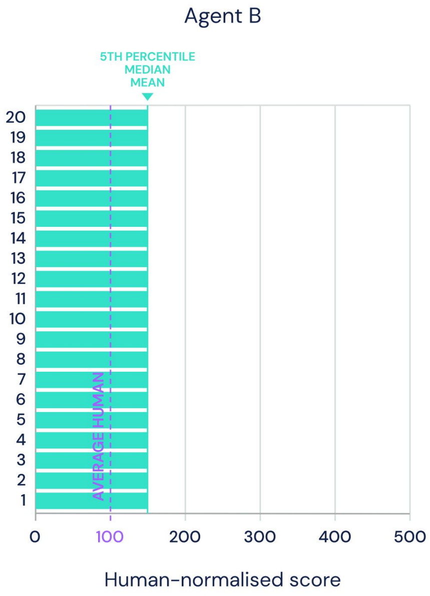
Omkar Ranadive

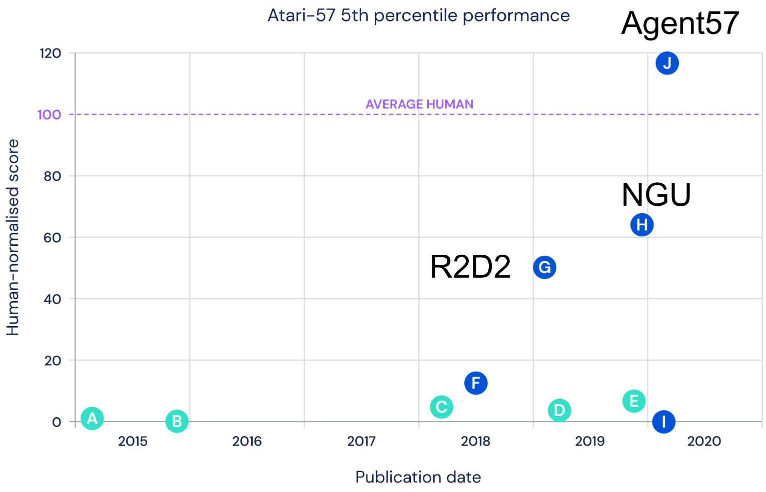
Arcade Learning Environment

- Suite of 57 different games of Atari 2600 console
- Each created by an independent party (no experimenter's bias)
- Each game has enough variation to claim generality

Are agents actually getting more intelligent?



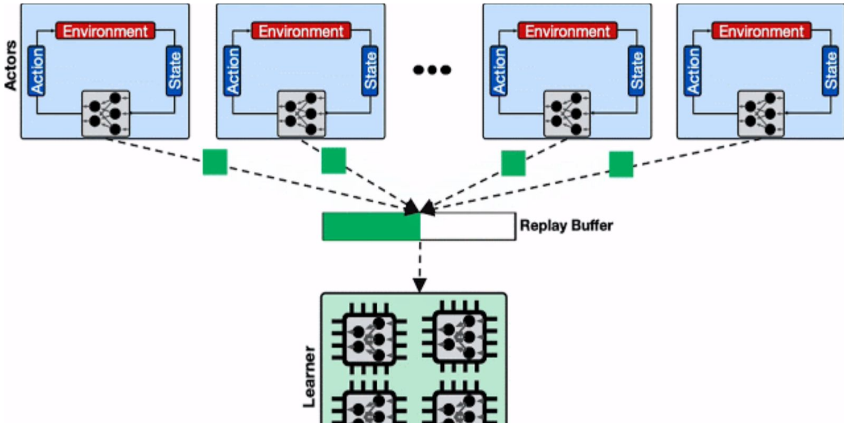




Dark Blue ones = Distributed agents
Light blue ones = single actor agents

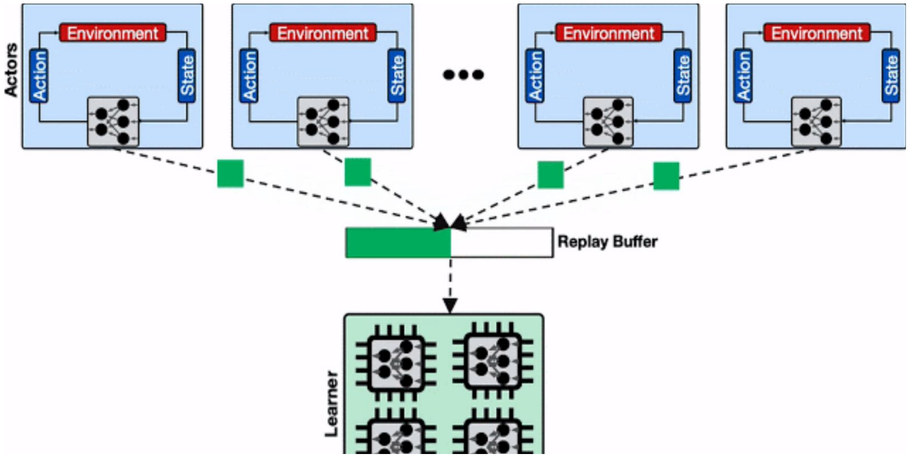
Distributed RL

- **Key idea:** Separate the acting from the learning
- Create multiple copies of the environment
- Each copy of the environment has its own actor
- Actors explore the environment and put experiences in central buffer
- A learner learns from the central buffer



Distributed RL

- Actor updates are asynchronous
- Leads to greater data collection and better exploration



Recurrent Replay Distributed DQN (R2D2)

- Memory is required to handle long-term dependencies better
- R2D2 uses RNN (LSTM) to handle this
- This is done in the distributed RL setting

Curiosity: Problem of exploration

- Desire to learn something, seek new experiences
- Learn about things which we don’t know much about
- In RL, this is useful for exploration
- So, make an agent explore the environment better by making it “curious”

Why does curiosity help?

- In RL, extrinsic rewards are usually sparse
- So, positive reinforcement happens only when we somehow encounter these rewards - difficult task
- Humans still explore the environment using motivation/curiosity
- Similarly, curiosity as an intrinsic reward would help the agent

Curiosity: Formal definition

- Curiosity is the error in predicting the consequence of its own actions
- Agent predicts the next state based on present state and action

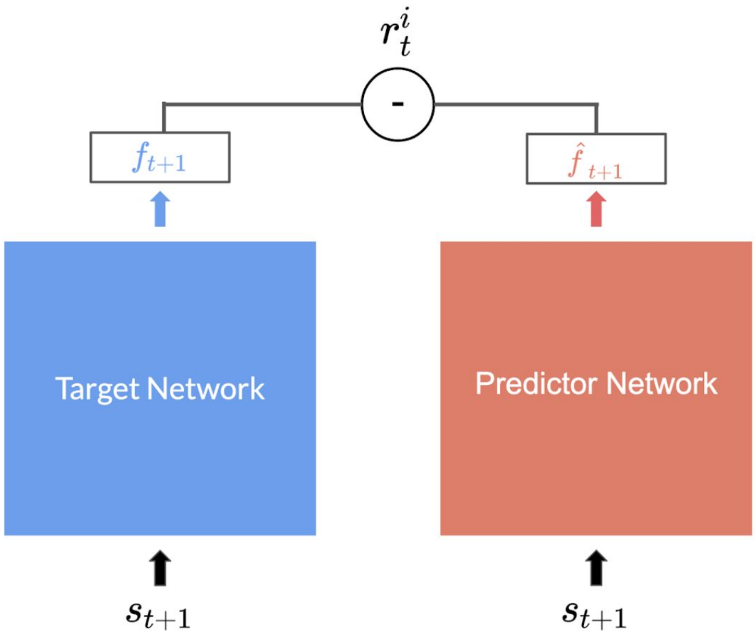
$$p(\phi(x_{t+1})|x_t, a_t)$$

- The intrinsic reward is then: $r_t = -\log p(\phi(x_{t+1})|x_t, a_t)$
- Lower the probability higher the reward. So the agent gets rewards if it predicts hard to predict states

The problem with curiosity

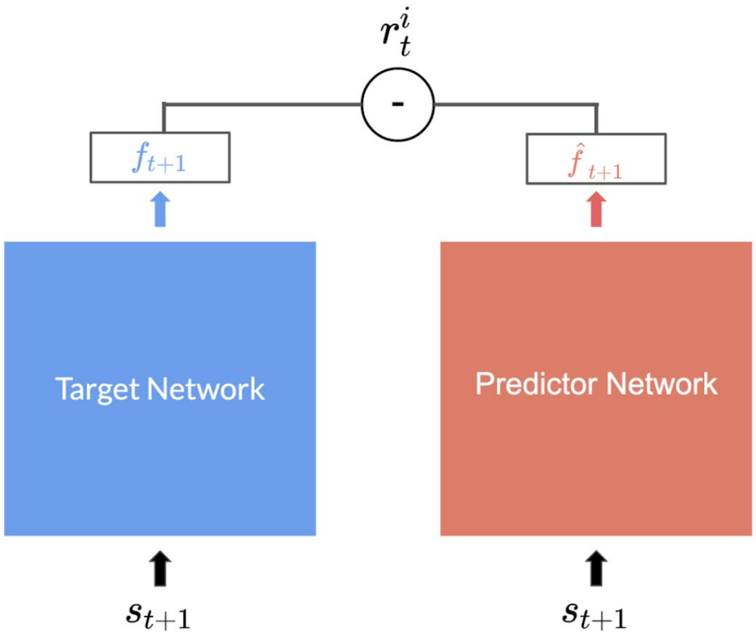
Random Network Distillation

- **Key idea:** Neural networks can predict those things better which it has already seen
- Ex - Samples from training set are always easier to predict
- Two networks: Randomly initialized and a predictor network
- The predictor network tries to predict the output of the random network



Random Network Distillation

- Error: $\|\hat{f}(x;\theta) - f(x)\|^2$
- Error will be low if the state is already seen before
- Error will be high for novel states
- High error is indicative of a novel state!



Agents eventually lose curiosity

- As agent explores more, novelty of the state reduces
- Eventually, the agent will lose curiosity and only exploit
- Agent becomes purely driven by extrinsic rewards
- Agent loses the opportunity to learn more from these novel states
- How to keep the agent motivated in a **directed** manner?

Never Give Up (NGU)

- Agent should keep exploration (never give up)
- Divide novelty into two parts -> Episodic novelty and lifelong novelty
- Episodic novelty: Discourage agents from revisiting the same states **within an episode**
- Lifelong novelty: Modulate how much the agent explores **over all episodes**

Never Give Up: Episodic Novelty

- Keep episodic memory M
- Fill it with states as the agent explores the agent
- For each state, compare it with the states present in the memory M to see how novel the new state is
- This is done using K-nearest neighbors

Never Give Up: Episodic Novelty

$$r_t^{\text{episodic}} = \frac{1}{\sqrt{n(f(x_t))}} \approx \frac{1}{\sqrt{\sum_{f_i \in N_k} K(f(x_t), f_i)} + c}$$

$$K(x, y) = \frac{\epsilon}{\frac{d^2(x,y)}{d_m^2} + \epsilon}$$

K(x, y) = similarity between two states; d = Euclidean distance

Never Give Up: Lifelong curiosity

$$r_t^i = r_t^{\text{episodic}} \cdot \min\{\max\{\alpha_t, 1\}, L\}$$

$$\alpha_t = 1 + \frac{err(x_t) - \mu_e}{\sigma_e}$$

L = 5, alpha = modulating factor, err(xt) = Random distillation error

Never Give Up: Architecture

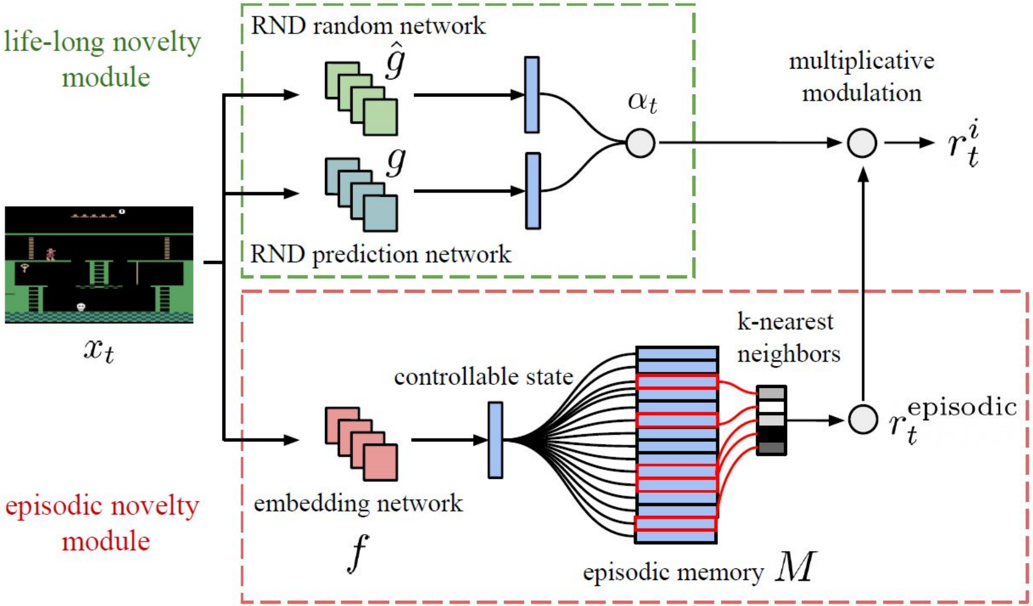


Image courtesy of Adria et al

Never Give Up: Scaling to distributed architecture

- Combined reward: $r_t^{\beta_i} = r_t^e + \beta_i r_t^i.$
- Instead of learning Q(x, a) learn Q(x, a, Bi)
- That is, we can learn different “goals”
- In this case, a goal is the degree of exploration
- Bi = 0, leads to no exploration and Bi = 1 leads to full exploration

Never Give Up: Scaling to distributed architecture

- Agents in different copies of environment will be given different value of Bi
- So each agent explores the environment differently
- The learner learns from every agents experience

NGU: Problems

- Equal weightage given to all policies
- Long term credit assignment is still difficult
- To deal with long term credit, adjust the discount factor dynamically
- Recap of discount factor:

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Building blocks of Agent57

- Split state-value function: $Q(x, a, j; \theta) = Q(x, a, j; \theta^e) + \beta_j Q(x, a, j; \theta^i)$
- So use two neural nets -> One for extrinsic rewards, one for intrinsic
- Easier to handle the variance in two rewards
- Adaptive exploration: Instead of Q(X, a, Bi) we learn Q(X, a, Bi, Gi) where Bi = term to control intrinsic exploration, Gi = discount factor to control extrinsic exploration
- Learn this using multi-arm bandits

Agent57: Putting everything together

- Agent57 is basically R2D2 + NGU + Metacontroller + State Value function decomposition
- Manages to beat human performance on every single game of the Atari2600 suite