INTRODUCTION TO EVOLUTION STRATEGY ALGORITHMS

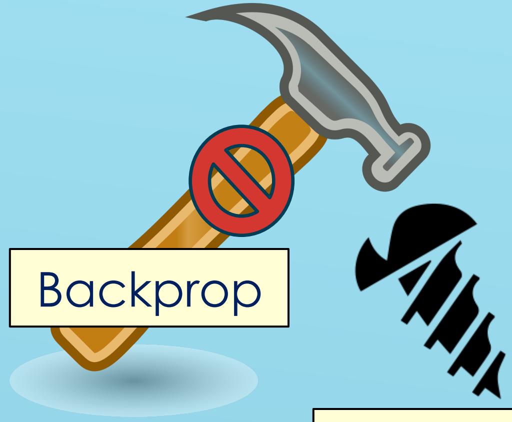
James Gleeson

Eric Langlois

William Saunders

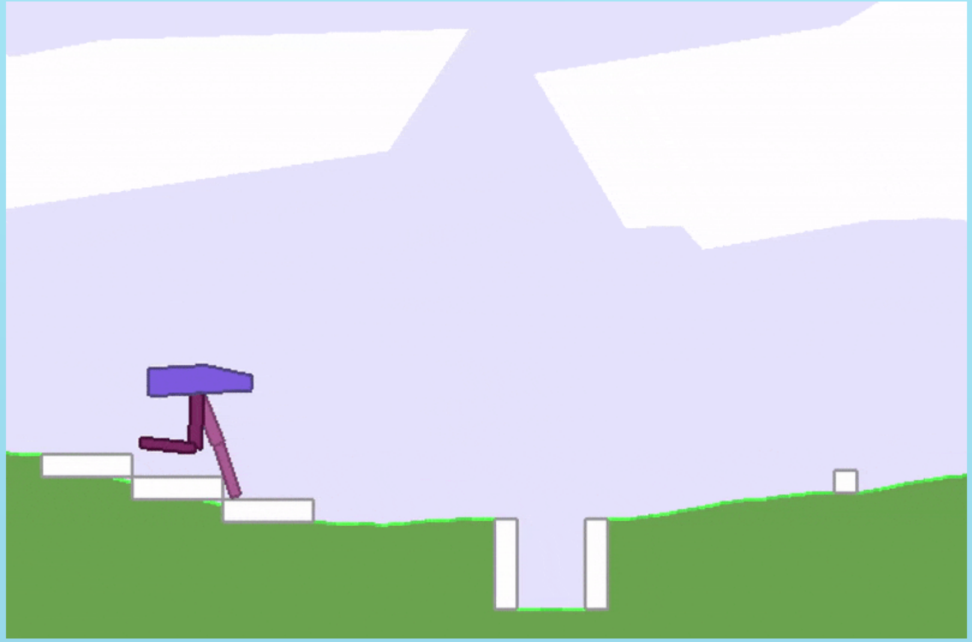
REINFORCEMENT LEARNING CHALLENGES

$f(\theta)$ is a **discrete function** of theta…
How do we get a gradient $\nabla_\theta f$?



Backprop

Discrete $f(\theta)$



$\nabla_\theta f$ **Local minima**

Credit assignment problem

Bob got a great bonus this year!

…what did Bob do to earn his bonus?

Time horizon: 1 year

[+] Met all his deadlines

[+] Took an ML course 3 years ago

Sparse reward signal

Evolution strategy

IDEA:
Lets just treat f like a black-box function when optimizing it.
"Try different θ", and see what works.
If we find good θ's, keep them, discard the bad ones.
Recombine θ_1 and θ_2 to form a new (possibly better) θ_3

EVOLUTION STRATEGY ALGORITHMS

- The template:

"Sample" new generation
Generate some parameter vectors for your neural networks.

MNIST ConvNet parameters

$\theta_1, \theta_2, \theta_3$

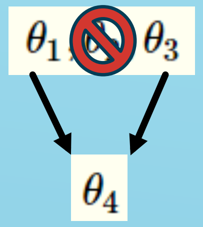
Fitness
Evaluate how well each neural network performs on a **training set**.

"Prepare" to sample the new generation:

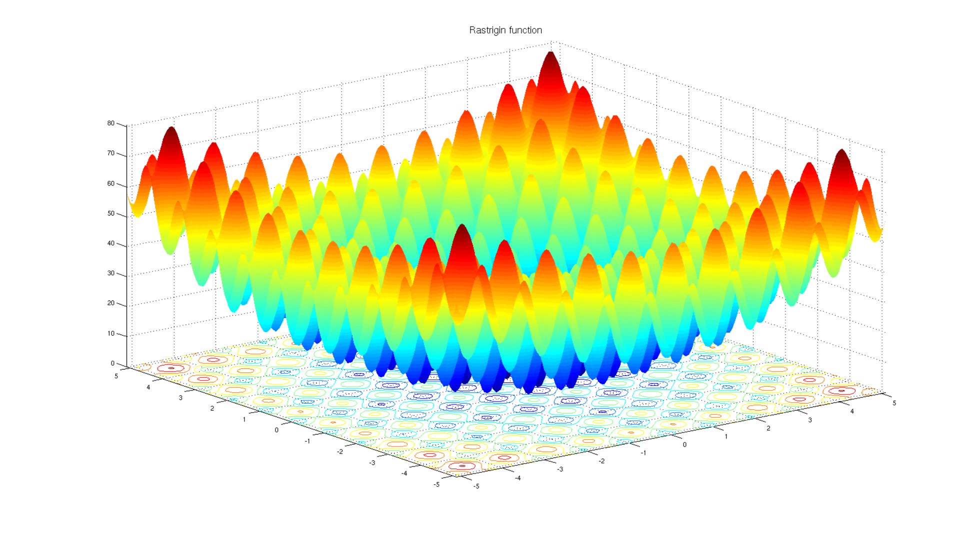
Given how well each "mutant" performed…

Natural selection! → Keep the good ones

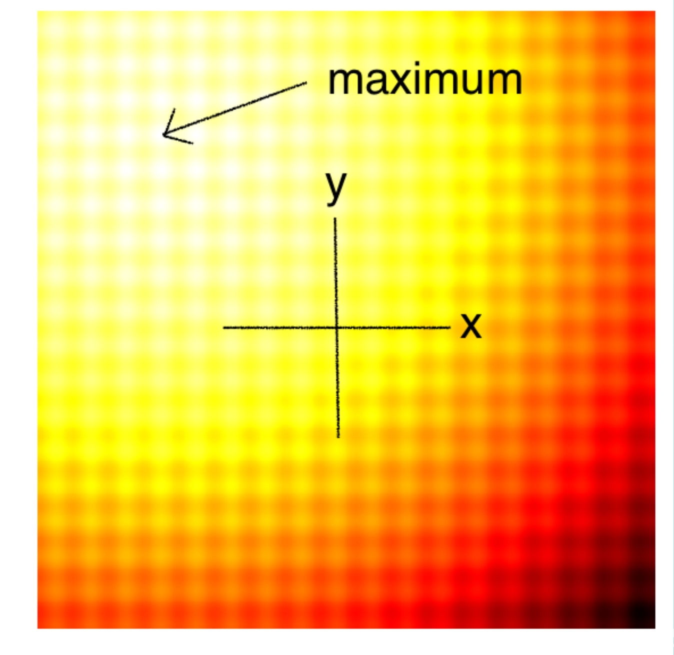
The ones that remain "recombine" to form the next generation.



SCARY "TEST FUNCTIONS" (1)



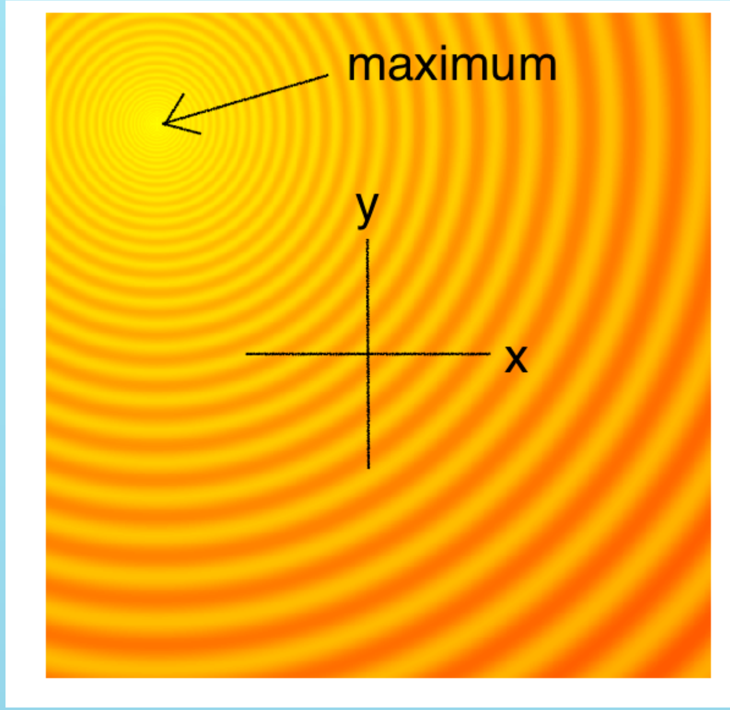
Rastrigin function
Test function



Rastrigin function (again)

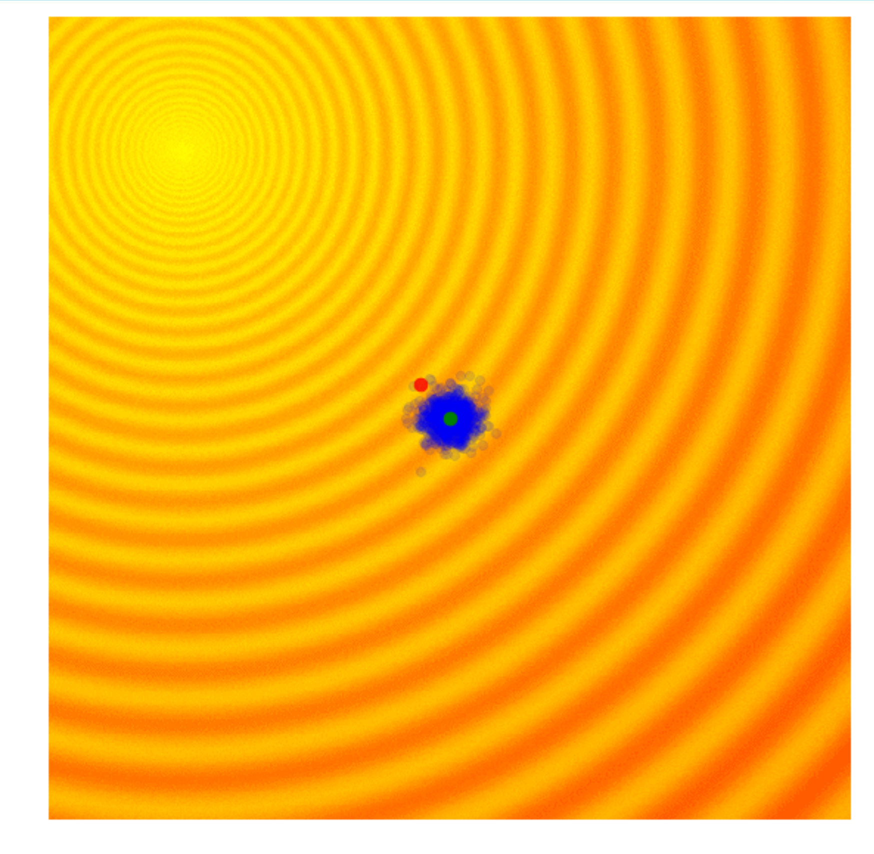
Lots of local optima; will be difficult to optimize with Backprop + SGD!

SCARY "TEST FUNCTIONS" (2)

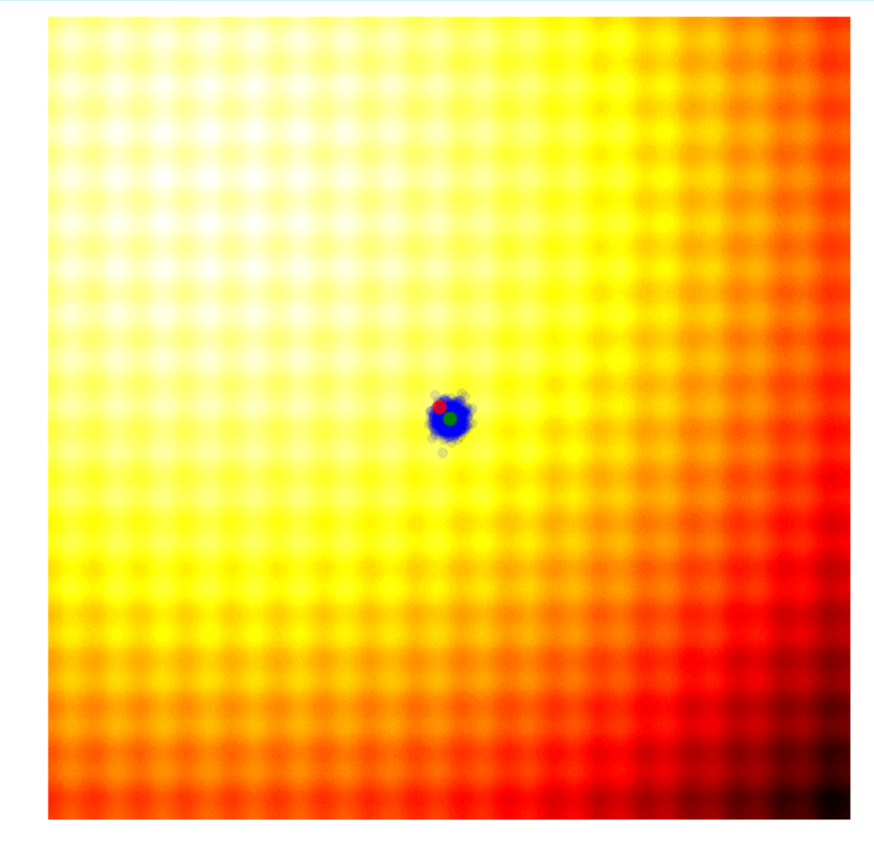


Schaffer function

WHAT WE WANT TO DO; "TRY DIFFERENT θ "



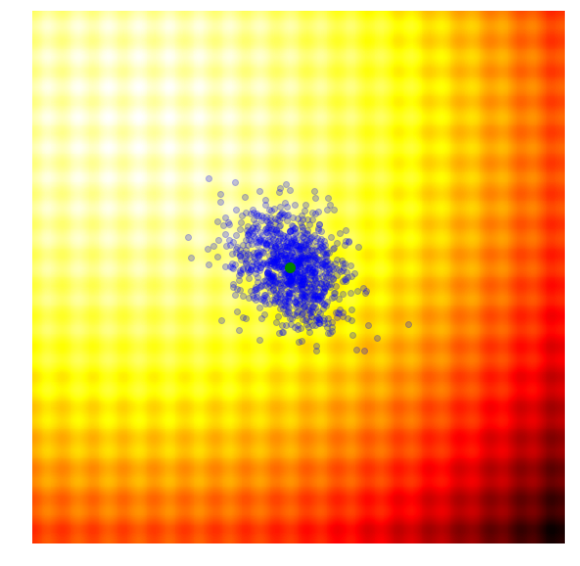
Schaffer

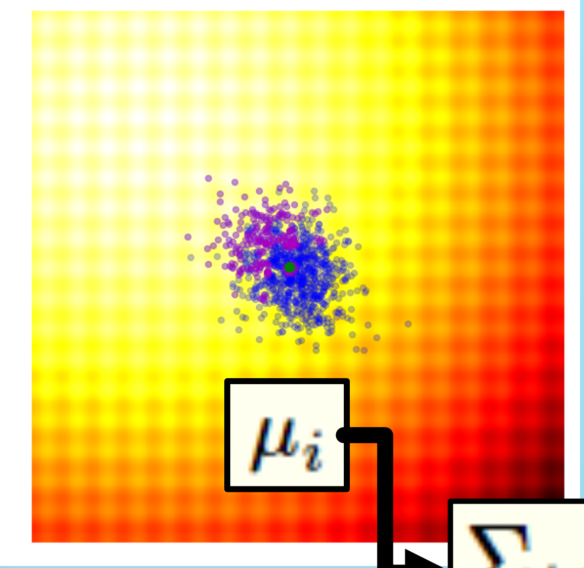


Rastrigin

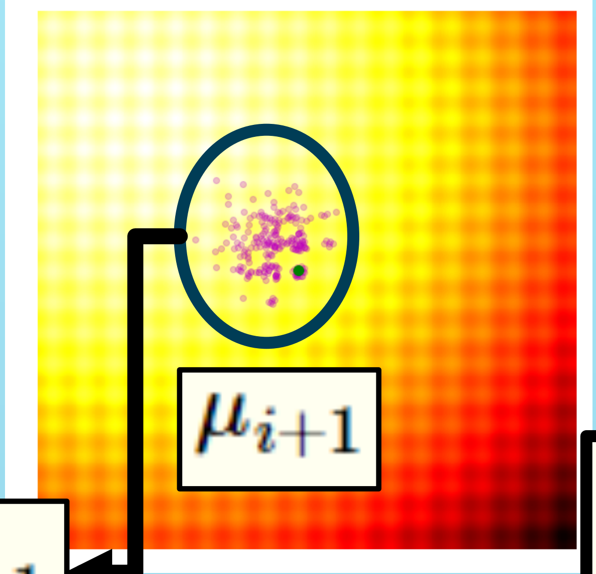
Algorithm: CMA-ES

CMA-ES; HIGH-LEVEL OVERVIEW



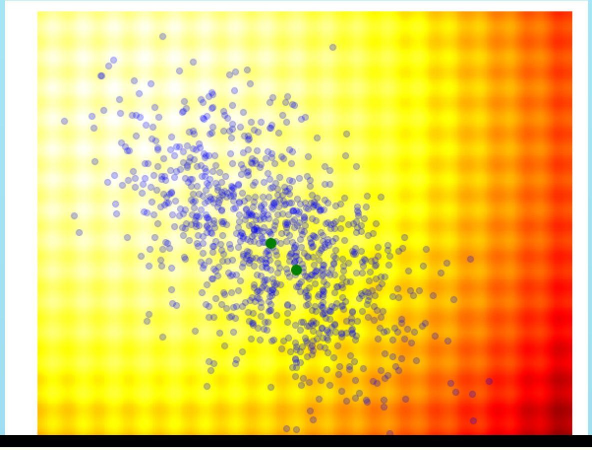


μ_i



μ_{i+1}

Σ_{i+1}



Generate mutants: $g(i) + \epsilon$

$\epsilon \sim N(\mu_{i+1}, \Sigma_{i+1})$

Step 1:
Calculate fitness of current generation $g(1)$

Step 2:
Natural selection!

Keep the top 25%.
(purple dots)

Step 3:
Recombine to form the new generation:

Discrepancy between mean of previous generation and top 25% will cast a wider net!

ES: LESS COMPUTATIONALLY EXPENSIVE

IDEA:

Sample neural-network parameters from
a multi-variate gaussian w/ **diagonal covariance** matrix.
Update $N(\theta{=}[\mu, \Sigma])$ parameters using REINFORCE gradient estimate.

$$\boxed{z} \sim N(\boxed{\theta = [\mu, \Sigma]})$$

Neural-network parameters.

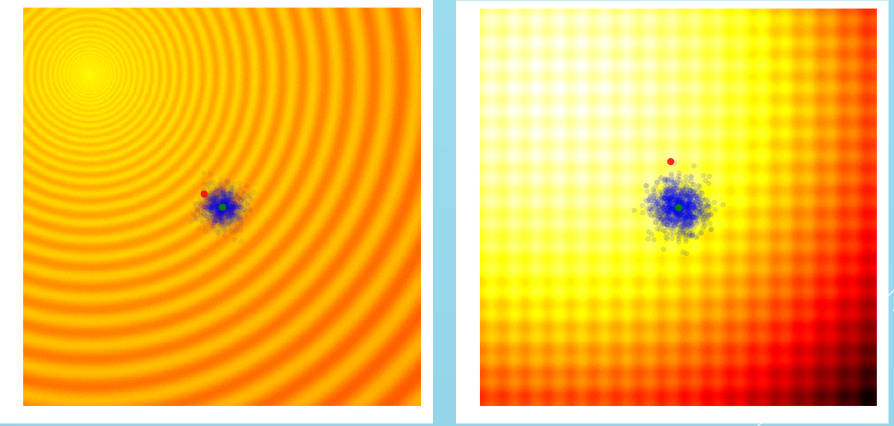
Parameters for sampling neural-network parameters.

$$J(\theta) = E_\theta[F(z)] = \int F(z)p(z|\theta)dz$$

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta E_\theta[F(\theta)] \\ &= E_\theta[F(z)\nabla_\theta \log p(z|\theta)] \\ &\approx \sum_{i=1}^{N} F(z_i)\nabla_\theta \log p(z_i|\theta)\end{aligned}$$

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

$O(\theta)$



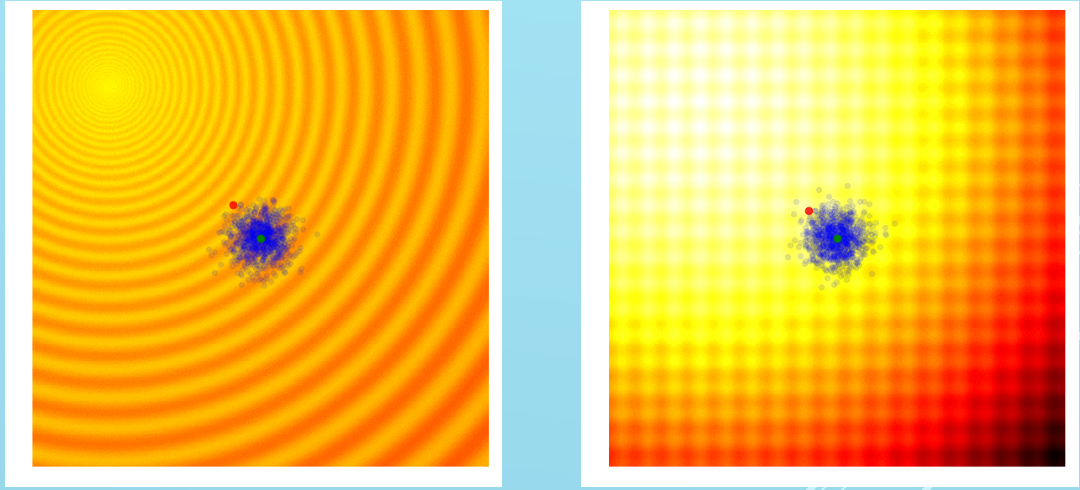
Adaptive σ and μ

ES: __EVEN_LESS__ COMPUTATIONALLY EXPENSIVE

- **IDEA:**

  Just use the same σ and μ for each parameter.
  ➔ Sample neural-network parameters from "**isotropic gaussian**"
  $= N(\mu, \sigma^2 I)$

- Seems suspiciously simple…but it can compete!
  - OpenAI ES paper:
    - σ is a hyperparameter
    - 1 set of hyperparameters for Atari
    - 1 set of hyperparameters for Mujoco
    - Competes with A3C and TRPO performance



Constant σ and μ

EVOLUTION STRATEGIES AS A SCALABLE ALTERNATIVE TO REINFORCEMENT LEARNING

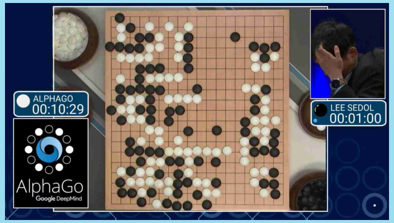
James Gleeson

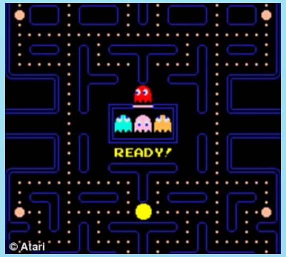
Eric Langlois

William Saunders

TODAY'S RL LANDSCAPE AND RECENT SUCCESS

- **Discrete action tasks:**
  - Learning to play Atari from raw pixels
  - Expert-level go player





- **Continuous action tasks:**
  - "Hopping" locomotion

Q-learning:

Learn the action-value function: $Q(s, a)$

Policy gradient; e.g. TRPO:

Learn the policy directly $\pi(a|s, \theta)$

Approximate the function using a neural-network,
train it using gradients computed via **backpropagation**
(i.e. the chain rule)

MOTIVATION: PROBLEMS WITH BACKPROPAGATION

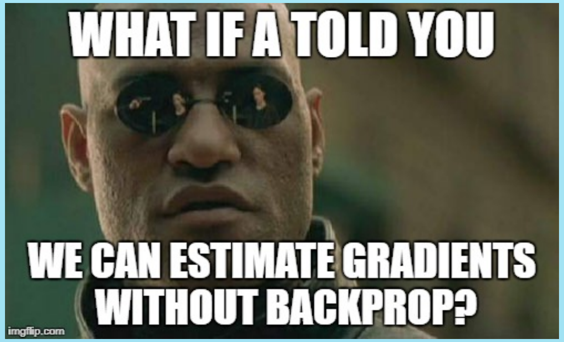
- Backpropagation isn't perfect:
  - GPU memory requirements
  - Difficult to parallelize
  - Cannot apply directly to non-differentiable functions
    - e.g. discrete functions $F(\theta)$ (the topic of this course)
  - Exploding gradient (e.g. for RNN's)
- You have a datacenter, and cycles to spend

RL problem

AN ALTERNATIVE TO BACKPROPAGATION: EVOLUTION STRATEGY (ES)

Claim:

$$\frac{\partial}{\partial\theta}F(\theta) \approx \frac{1}{\sigma^2}E_{\epsilon\sim N(0,\sigma^2)}[\epsilon F(\theta+\epsilon)]$$



And have it be ***embarrassingly parallel***?

Proof:

$$F(\theta+\epsilon) \approx f(\theta) + f'(\theta) + f''(\theta)\epsilon^2/2$$

2nd order Taylor series approximation

$$E_\epsilon[\epsilon F(\theta+\epsilon)] \approx E_\epsilon[\epsilon F(\theta) + \epsilon^2 F'(\theta) + \epsilon^3 F''(\theta)/2]$$

$$= \underbrace{E_\epsilon[\epsilon]}_{\substack{=\mu \\ =0}}F(\theta) + \underbrace{E_\epsilon[\epsilon^2]}_{\substack{=E_\epsilon[(\epsilon-\mu)^2] \\ =\sigma^2}}F'(\theta) + \underbrace{E_\epsilon[\epsilon^3]}_{\substack{E_\epsilon[(\epsilon-\mu)^i]=0 \\ \text{for } N(\mu,\sigma^2) \text{ and odd i}}}F''(\theta)$$

$F(\theta)$ independent of ϵ

$$\epsilon \sim N(0,\sigma^2)$$

$$E_\epsilon[\epsilon F(\theta+\epsilon)] = \sigma^2 F'(\theta)$$

$$\underbrace{\frac{\partial}{\partial\theta}F(\theta)}_{\text{Gradient of objective } F(\theta)} = \underbrace{\frac{1}{\sigma^2}E_\epsilon[\epsilon F(\theta+\epsilon)]}_{\text{No derivates of } F(\theta)}$$

Gradient of objective $F(\theta)$

No derivates of $F(\theta)$

No chain rule / backprop required!

Relevant to our course:

$F(\theta)$ could be a discrete function of θ

THE MAIN CONTRIBUTION OF THIS PAPER

- Criticisms:
  - Evolution strategy aren't new!
  - **Common sense:**
    The variance/bias of this **gradient estimator** will be too high, making the algorithm unstable on today's problems!

$$\frac{\partial}{\partial\theta}F(\theta+\epsilon) \approx \frac{1}{\sigma^2}E_{\epsilon\sim N(0,\sigma^2)}[\epsilon F(\theta+\epsilon)]$$

- This paper aims to **refute your common sense**:
  - Comparison against state-of-the-art RL algorithms:
    - **Atari:**
      Half the games do better than a ***recent algorithm (A3C)***, half the games do worse
    - **Mujoco:**
      Can match state-of-the-art ***policy gradients*** on continuous action tasks.

Linear speedups with more compute nodes: 1 day with A3C ➔ 1 hour with ES

FIRST ATTEMPT AT ES: THE SEQUENTIAL ALGORITHM

Gradient estimator needed for updating θ:

$$\frac{\partial}{\partial\theta}\boxed{F}(\theta+\epsilon) \approx \frac{1}{\sigma^2} E_{\epsilon\sim N(0,\sigma^2)}[\epsilon F(\theta+\epsilon)]$$

Sample: $\frac{1}{\sigma^2}\epsilon F(\theta+\epsilon)$

In RL, the fitness $F(\theta)$ is defined as:

$F(\theta) = E_\tau[R_\tau]$

Where:

τ = An episode of state (s) action (a) pairs

R_τ = Sum of rewards received over episode τ

<u>Embarassingly parallel!</u>

for each $Worker \downarrow i \quad i=1..n$:

$Worker \downarrow i$:computes $F \downarrow i$ in parallel

Algorithm 1 Evolution Strategies

1: **Input:** Learning rate α, noise standard deviation σ, initial policy parameters θ_0

2: **for** $t = 0, 1, 2, \ldots$ **do**

3: Sample $\epsilon_1, \ldots \epsilon_n \sim \mathcal{N}(0, I)$ — Generate n random perturbations of θ

4: Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$ for $i = 1, \ldots, n$ — ***<u>Sequentially</u>*** run each mutant

5: Set $\theta_{t+1} \leftarrow \theta_t + \alpha\frac{1}{n\sigma}\sum_{i=1}^{n} F_i\epsilon_i$ — Compute gradient estimate

6: **end for**

SECOND ATTEMPT: THE PARALLEL ALGORITHM

Algorithm 2 Parallelized Evolution Strategies

Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
Initialize: n workers with known random seeds, and initial parameters θ_0
for $t = 0, 1, 2, \ldots$ **do**
  for each worker $i = 1, \ldots, n$ **do**
    Sample $\epsilon_i \sim \mathcal{N}(0, I)$
    Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$
  end for
  Send all scalar returns F_i from each worker to every other worker
  for each worker $i = 1, \ldots, n$ **do**
    Reconstruct all perturbations ϵ_j for $j = 1, \ldots, n$ using known random seeds
    Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{j=1}^{n} F_j \epsilon_j$
  end for
end for

Embarassingly parallel!

With F_j and ϵ_j known by everyone, each worker compute the ***same gradient estimate***

Tradeoff:
redundant computation over $|\theta|$ message size

- **KEY IDEA:** Minimize communication cost avoid sending $\mathrm{len}(\epsilon) = |\theta|$, send $\mathrm{len}(F_i) = 1$ instead.

  How? *Each worker* ***reconstructs*** *random perturbation vector ϵ*

  …How? *Make initial random seed of $Worker_i$ globally known.*

EXPERIMENT: HOW WELL DOES IT SCALE?

Actual speedup

Ideal speedup (perfectly linear)

$\frac{657min}{60min} \approx 11.0\times$

$\frac{200cores}{18cores} \approx 11.1\times$

$\frac{657min}{10min} \approx 65.7\times$

$\frac{1440cores}{18cores} \approx 80.0\times$

Criticism:

Are diminishing returns due to:

- increased communication cost from more workers
- less reduction in variance of the gradient estimate from more workers

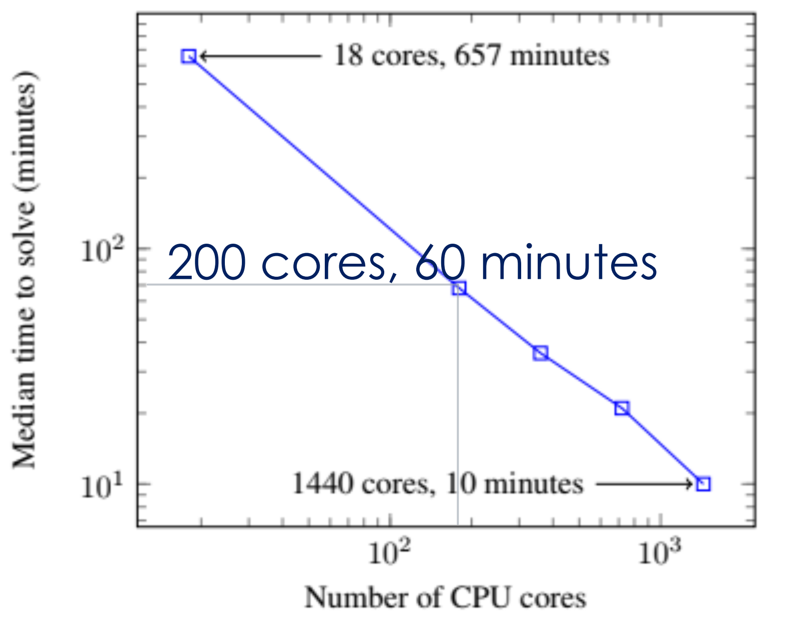


Figure 1: Time to reach a score of 6000 on 3D Humanoid with different number of CPU cores. Experiments are repeated 7 times and median time is reported.

- Linearly!

With diminishing returns; often inevitable.

INTRINSIC DIMENSIONALITY OF THE PROBLEM

$$\frac{\partial}{\partial\theta}F(\theta) \approx \frac{1}{\sigma^2}E_{\epsilon\sim N(0,\sigma^2)}[\epsilon F(\theta+\epsilon)]$$

$$E_\epsilon[\frac{F(\theta)\epsilon}{\sigma^2}] = 0, \quad \text{since } \ E_\epsilon[\epsilon] = 0$$

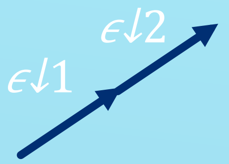
$$\frac{\partial}{\partial\theta}F(\theta) \approx E_{\epsilon\sim N(0,\sigma^2)}[\frac{F(\theta+\epsilon)-F(\theta)}{\sigma^2}\epsilon]$$

$\approx$ finite differences in some random direction ϵ

➔ # update steps scales with $|\theta|$?

Justification:

E.g. Simple linear regression:
Double $|\theta| \to |\theta'|$



$\boxed{y = x\theta}$

$\frac{\partial}{\partial\theta}F(\theta)$

$x' = (x, x)$
$y' = x'\theta'$
$\boxed{\approx 2x\theta}$

$\epsilon_1 \sim \epsilon_2 \sim N(\mu, \sigma^2)$

$\approx 4\frac{\partial}{\partial\theta}F(\theta)$
$\eta' = \frac{\eta}{2}, \sigma' = \frac{\sigma}{2}$

After adjusting η and σ,
Update step has the same effect.
➔ Same # of update steps.

Argument:
# of update steps in ES scales with the ***intrinsic dimensionality*** of θ needed for the problem, ***not with the length*** of θ.

WHEN IS ES A BETTER CHOICE THAN POLICY GRADIENTS?

How do we compute gradients?

Policy gradients:
Policy network outputs a softmax of probabilities for different discrete actions, and we **sample an action randomly**.

$$\nabla_\theta F_{PG}(\theta) = \mathbb{E}_\epsilon \left\{ R(\mathbf{a}(\epsilon,\theta)) \nabla_\theta \log p(\mathbf{a}(\epsilon,\theta);\theta) \right\}$$

Evolution strategy (ES):
We **randomly perturb our parameters:** $\theta \rightarrow \tilde{\theta}$
then select actions according to $\tilde{\theta}$

$$\nabla_\theta F_{ES}(\theta) = \mathbb{E}_\xi \left\{ R(\mathbf{a}(\xi,\theta)) \nabla_\theta \log p(\tilde{\theta}(\xi,\theta);\theta) \right\}$$

ASIDE: In case you forget; for independent X & Y:

$$\begin{aligned} Var[XY] &= E[X^2Y^2] - E[XY]^2 \\ &= E[X^2]E[Y^2] - E[X]^2E[Y]^2 \\ &\# \ Since\ E[X^2] = Var[X] + E[X]^2 \\ &= (Var[X] + E[X]^2)(Var[Y] + E[Y]^2) - E[X]^2E[Y]^2 \\ &= Var[X]Var[Y] + Var[X]E[Y]^2 + Var[Y]E[X]^2 \\ &\approx Var[X]Var[Y] \end{aligned}$$

$$\begin{aligned} \mathrm{Var}[\nabla_\theta F_{PG}(\theta)] &\approx \mathrm{Var}[R(\mathbf{a})]\, \boxed{\mathrm{Var}[\nabla_\theta \log p(\mathbf{a};\theta)]}, \\ \mathrm{Var}[\nabla_\theta F_{ES}(\theta)] &\approx \mathrm{Var}[R(\mathbf{a})]\, \boxed{\mathrm{Var}[\nabla_\theta \log p(\tilde{\theta};\theta)]}. \end{aligned}$$

Independent of episode length.

$$\sum_{t=1}^{T} \nabla_\theta \log p(a_t;\theta)$$

Variance of gradient estimate grows linearly with the length of the episode.
γ only fixes this for short-term returns!

Credit assignment problem
ES makes fewer (potentially incorrect) assumptions

EXPERIMENT: ES ISN'T SENSITIVE TO LENGTH OF EPISODE τ

- ***Frame-skip F:***
  - *Agent can select an action every F frames of input pixels*
  - *E.g. F = 4*
    frame 1: agent selects an action
    frame 1-3: agent is forced to take Noop action
    IDEA:
    artificially inflate the length of an episode τ

Playing pong with frameskip

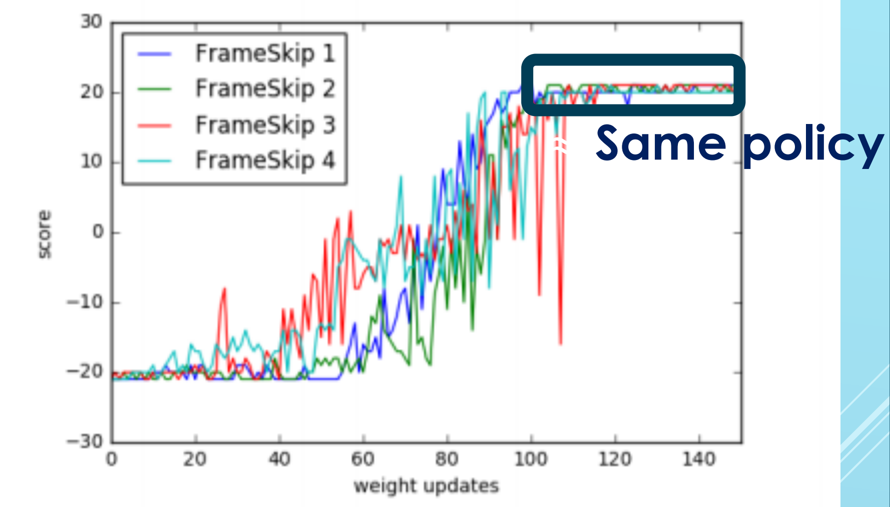


Figure 2: Learning curves for Pong using varying frame-skip parameters. Although performance is stochastic, each setting leads to about equally fast learning, with each run converging in around 100 weight updates.

Argument:
Since the ES algorithm doesn't make ***any assumption*** about time horizon γ (decaying reward), it is less sensitive to long episodes τ (i.e. the ***credit assignment problem***)

EXPERIMENT: LEARNED PERFORMANCE

- The authors looked at:
  - discrete action tasks -- Atari
  - continuous action tasks -- Mujoco

EXPERIMENT: DISCRETE ACTION TASKS -- ATARI

- Paper's claim:
  "Given the same amount of compute time as other algorithms, ***compared to A3C***, ES does better on 21 games, worse on 29 "

Slightly misleading claim if you aren't reading carefully:

A3C still does better on most games across all algorithms
➔ES is still beaten by other algorithms when it beats A3C

50 games in total

Best score:	4 8%	19 38%	11 22%	7 14%	9 18%
Game	**DQN**	**A3C FF, 1 day**	**HyperNEAT**	**ES FF, 1 hour**	**A2C FF**
Montezuma's Revenge	50.0	**53.0**	0.0	0.0	0.0
Breakout	303.9	**551.6**	2.8	9.5	368.5
Pong	16.2	11.4	17.4	**21.0**	20.8
Skiing		13700.0	7983.6	**15442.5**	15245.8

EXPERIMENT: CONTINUOUS ACTION TASKS -- MUJOCO

Sampling complexity:
How many steps in the environment were needed to reach X% of policy gradient performance?

$\frac{\#\ ES\ Timesteps}{\#\ TRPO\ Timesteps}$

< 1 ➔ Better sampling complexity
> 1 ➔ Worse sampling complexity

Table 1: MuJoCo tasks: Ratio of ES timesteps to TRPO timesteps needed to reach various percentages of TRPO's learning progress at 5 million timesteps.

Environment	25%	50%	75%	100%
HalfCheetah	0.15	0.49	0.42	0.58
Hopper	0.53	3.64	6.05	6.94
InvertedDoublePendulum	0.46	0.48	0.49	1.23
InvertedPendulum	0.28	0.52	0.78	0.88
Swimmer	0.56	0.47	0.53	0.30
Walker2d	0.41	5.69	8.02	7.88

Harder tasks: at most 10x more samples required

Simpler tasks: as few as 0.33x samples required

SUMMARY: EVOLUTION STRATEGY

- ES are a viable alternative to **current RL algorithms**:

Q-learning:

Learn the action-value function: $Q(s, a)$

Policy gradient; e.g. TRPO:

Learn the policy directly $\pi(a|s, \theta)$

- **ES:**

Treat the problem like a black-box, perturb θ and evaluate fitness $F(\theta)$:

$$F(\theta) = E_\tau[R_\tau]$$

Where:

$$\begin{aligned} \tau &= \text{An episode of state } (s) \text{ action } (a) \text{ pairs} \\ R_\tau &= \text{Sum of rewards received over episode } \tau \end{aligned}$$

- No potentially incorrect assumptions about ***credit assignment problem*** (e.g. time horizon γ)
- No backprop required
  - Embarrassingly parallel
  - Lower GPU memory requirements