Non-Differentiable Optimization
RL for the Travelling Salesman Problem (TSP)
Definition (TSP):

In TSP, we want to find a Hamiltonian cycle in a complete graph with the shortest length.

I.e. We want to visit all nodes in the shortest amount of time.
• State $s$ is a graph with $n$ nodes. ($n$ is a fixed constant.)
• Action $a = (a_1, a_2, ..., a_n)$ is a permutation of the nodes (i.e. a Hamiltonian cycle).
• Policy with weights $\theta$ is given by:

$$p_\theta(a|s) = \prod_{t=1}^{n} p_\theta(a_t|s, a_{1:t-1})$$

where $p_\theta(a_t|s, a_{1:t-1})$ is implemented by a modified transformer network.
TSP as a RL-Problem: Attention, Learn To Solve Routing Problems! [Kool et al., 2019]

- Loss is defined by the weight of the loop:
  \[ L(a) = \sum_{i=1}^{n} w(a_i, a_{i+1}) \]

- Update the parameters \( \theta \) using REINFORCE:
  \[ \nabla L(\theta | s) = \mathbb{E}_{p_{\theta}(a|s)} L(a) \nabla \log p_{\theta}(a|s) \]
<table>
<thead>
<tr>
<th>Method</th>
<th>$n = 20$ Obj.</th>
<th>Gap</th>
<th>Time</th>
<th>$n = 50$ Obj.</th>
<th>Gap</th>
<th>Time</th>
<th>$n = 100$ Obj.</th>
<th>Gap</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concorde</td>
<td>3.84</td>
<td>0.00%</td>
<td>1m</td>
<td>5.70</td>
<td>0.00%</td>
<td>2m</td>
<td>7.76</td>
<td>0.00%</td>
<td>3m</td>
</tr>
<tr>
<td>LKH3</td>
<td>3.84</td>
<td>0.00%</td>
<td>18s</td>
<td>5.70</td>
<td>0.00%</td>
<td>5m</td>
<td>7.76</td>
<td>0.00%</td>
<td>21m</td>
</tr>
<tr>
<td>Gurobi</td>
<td>3.84</td>
<td>0.00%</td>
<td>7s</td>
<td>5.70</td>
<td>0.00%</td>
<td>2m</td>
<td>7.76</td>
<td>0.00%</td>
<td>17m</td>
</tr>
<tr>
<td>Gurobi (1s)</td>
<td>3.84</td>
<td>0.00%</td>
<td>8s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nearest Insertion</td>
<td>4.33</td>
<td>12.91%</td>
<td>1s</td>
<td>6.78</td>
<td>19.03%</td>
<td>2s</td>
<td>9.46</td>
<td>21.82%</td>
<td>6s</td>
</tr>
<tr>
<td>Random Insertion</td>
<td>4.00</td>
<td>4.36%</td>
<td>0s</td>
<td>6.13</td>
<td>7.65%</td>
<td>1s</td>
<td>8.52</td>
<td>9.69%</td>
<td>3s</td>
</tr>
<tr>
<td>Farthest Insertion</td>
<td>3.93</td>
<td>2.36%</td>
<td>1s</td>
<td>6.01</td>
<td>5.53%</td>
<td>2s</td>
<td>8.35</td>
<td>7.59%</td>
<td>7s</td>
</tr>
<tr>
<td>Nearest Neighbor</td>
<td>4.50</td>
<td>17.23%</td>
<td>0s</td>
<td>7.00</td>
<td>22.94%</td>
<td>0s</td>
<td>9.68</td>
<td>24.73%</td>
<td>0s</td>
</tr>
<tr>
<td>Vinyals et al. (gr.)</td>
<td>3.88</td>
<td>1.15%</td>
<td></td>
<td>7.66</td>
<td>34.48%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bello et al. (gr.)</td>
<td>3.89</td>
<td>1.42%</td>
<td></td>
<td>5.95</td>
<td>4.46%</td>
<td></td>
<td>8.30</td>
<td>6.90%</td>
<td></td>
</tr>
<tr>
<td>Dai et al.</td>
<td>3.89</td>
<td>1.42%</td>
<td></td>
<td>5.99</td>
<td>5.16%</td>
<td></td>
<td>8.31</td>
<td>7.03%</td>
<td></td>
</tr>
<tr>
<td>Nowak et al.</td>
<td>3.93</td>
<td>2.46%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EAN (greedy)</td>
<td>3.86</td>
<td>0.66%</td>
<td>2m</td>
<td>5.92</td>
<td>3.98%</td>
<td>5m</td>
<td>8.42</td>
<td>8.41%</td>
<td>8m</td>
</tr>
<tr>
<td>AM (greedy)</td>
<td><strong>3.85</strong></td>
<td><strong>0.34%</strong></td>
<td><strong>0s</strong></td>
<td><strong>5.80</strong></td>
<td><strong>1.76%</strong></td>
<td><strong>2s</strong></td>
<td><strong>8.12</strong></td>
<td><strong>4.53%</strong></td>
<td><strong>6s</strong></td>
</tr>
</tbody>
</table>
Evolutionary Strategies for Optimizing RL Problems
Black-Box Optimization

\[ F(\theta) \]

Black Box F

θ

optimization algorithm

F(θ)
Evolutionary Strategies (ES)

initialize $\psi$

repeat:
- Generate a set of samples $D = \{(\theta_1, F(\theta_1)), \ldots, (\theta_n, F(\theta_n))\}$, where $\theta_i$ is drawn from the distribution $p(\theta)$
- Evaluate the fitness of samples in $D$
- Use the fitness to update the parameter $\psi$
Example: Simple Gaussian ES

Define $\psi = (\mu, \sigma)$ and $\theta_i \sim N(\mu, \sigma^2 I)$,

Initialize the parameters $\psi^{(0)} = (\mu^{(0)}, \sigma^{(0)})$, size of the elite set $m \in \{1, ..., n\}$

repeat for $t \in \{1, ..., T\}$:

- Generate a set of samples $D = \{(\theta_1, F(\theta_1)), ..., (\theta_n, F(\theta_n))\}$, where $\theta_i \sim N(\mu^{(0)}, \sigma^{(0)^2 I})$
- Let $L$ be the set of $\theta_i$ with the largest values of $F(\theta_i)$ with $|L| = m$
- Update $\psi^{(t+1)} = (\mu^{(t+1)}, \sigma^{(t+1)}) = (\text{mean}(L), \text{std}(L))$
Example: Simple Gaussian ES

Want to find the minimum of $F(\theta) = \text{distance of } \theta \text{ to } +$

$x = \text{current mean}$

$o = \text{generated points with mean } x \text{ and std } \sigma$

$o = \text{best performing points}$
In policy gradient RL, we are optimizing $v_\theta(s) = \mathbb{E}_{\pi_\theta} [G | s] = \mathbb{E}_{\pi_\theta} [\sum_{t=0}^{T} \gamma^t r_t | s]$

We use the log-likelihood trick in REINFORCE to get:

$\nabla_\theta v_\theta(s) \propto \mathbb{E}_{\pi_\theta} [G_t \nabla_\theta \log(\pi_\theta(a, s))]$
In contrast to REINFORCE, which optimizes $v_\theta(s) = \mathbb{E}_{\pi_\theta}[G|s]$, we will optimize the following expectation with respect to the parameter $\psi$:

$$\mathbb{E}_{\theta \sim p_\psi}[F(\theta)]$$

where $F(\theta) = G_\theta$. Here $G_\theta$ is the cumulative reward of one episode following policy $\pi_\theta$ and an unbiased estimate of the value function $v_\theta(s_0) = \mathbb{E}_{\pi_\theta}[G|s_0]$.
Say, we want to model $\theta$ by a Gaussian with mean $\psi$ and fixed covariance $\sigma I$. I.e. $\theta \sim p_\psi = N(\psi, \sigma^2 I)$

Then we can write the gradient as follows:

$$
\nabla_\psi \mathbb{E}_{\theta \sim p_\psi} [F(\theta)] = \mathbb{E}_{\theta \sim p_\psi} [F(\theta) \nabla_\psi \log p_\psi(\theta)] \\
\propto \mathbb{E}_{\theta \sim p_\psi} [F(\theta) \nabla_\psi (-\frac{1}{2} \frac{(\theta - \psi)^2}{\sigma^2})] \\
= \mathbb{E}_{\theta \sim p_\psi} [F(\theta) \frac{(\theta - \psi)}{\sigma^2}] \\
= \mathbb{E}_{\epsilon \sim N(0,I)} [F(\psi + \epsilon \sigma) \frac{(\epsilon \sigma)}{\sigma^2}] \\
= \mathbb{E}_{\epsilon \sim N(0,I)} [F(\psi + \epsilon \sigma) \frac{\epsilon}{\sigma}] \\
= \frac{1}{\sigma} \mathbb{E}_{\epsilon \sim N(0,I)} [\epsilon F(\psi + \epsilon \sigma)]
$$
Recalling the definition of directional derivatives in higher dimensions,

\[ D_v f(x) = \lim_{h \to 0} \frac{f(x+hv) - f(x)}{h} \]

we see that our gradient can be interpreted as randomized finite differences:

\[ \frac{1}{\sigma} \mathbb{E}_{\epsilon \sim N(0,1)} [\epsilon F(\psi + \epsilon \sigma)] \]

\[ = \mathbb{E}_{\epsilon \sim N(0,1)} [\epsilon \frac{F(\psi + \epsilon \sigma) - F(\psi)}{\sigma}] \]
Algorithm 2 Parallelized Evolution Strategies

1: **Input:** Learning rate $\alpha$, noise standard deviation $\sigma$, initial policy parameters $\hat{\psi}_0$
2: **Initialize:** $n$ workers with known random seeds, and initial parameters $\hat{\psi}_0$
3: for $t = 0, 1, 2, \ldots$ do
4:   for each worker $i = 1, \ldots, n$ do
5:     Sample $\epsilon_i \sim \mathcal{N}(0, I)$
6:     Compute returns $F_i = F(\hat{\psi}_t + \sigma \epsilon_i)$
7:   end for
8:   Send all scalar returns $F_i$ from each worker to every other worker
9:   for each worker $i = 1, \ldots, n$ do
10:      Reconstruct all perturbations $\epsilon_j$ for $j = 1, \ldots, n$ using known random seeds
11:      Set $\hat{\psi}_{t+1} \leftarrow \hat{\psi}_t + \alpha \frac{1}{n\sigma} \sum_{j=1}^{n} F_j \epsilon_j$
12:   end for
13: end for
Advantages of ES

● Efficient parallelization
  ○ by synchronizing random seeds of workers before training, each worker knows what perturbation the others used
  ○ then we only need to communicate the episode return between workers
● computation and memory efficient since no backpropagation is needed
● robust: little hyperparameter tuning needed, no frameskip needed
● easy to implement

But

● 3-10 times less data efficient
Experiments on MuJoCo:

Ratio of ES timesteps to TRPO timesteps needed to reach various percentages of TRPO’s learning progress at 5 million timesteps.

<table>
<thead>
<tr>
<th>Environment</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>HalfCheetah</td>
<td>0.15</td>
<td>0.49</td>
<td>0.42</td>
<td>0.58</td>
</tr>
<tr>
<td>Hopper</td>
<td>0.53</td>
<td>3.64</td>
<td>6.05</td>
<td>6.94</td>
</tr>
<tr>
<td>InvertedDoublePendulum</td>
<td>0.46</td>
<td>0.48</td>
<td>0.49</td>
<td>1.23</td>
</tr>
<tr>
<td>InvertedPendulum</td>
<td>0.28</td>
<td>0.52</td>
<td>0.78</td>
<td>0.88</td>
</tr>
<tr>
<td>Swimmer</td>
<td>0.56</td>
<td>0.47</td>
<td>0.53</td>
<td>0.30</td>
</tr>
<tr>
<td>Walker2d</td>
<td>0.41</td>
<td>5.69</td>
<td>8.02</td>
<td>7.88</td>
</tr>
</tbody>
</table>
While exploration in RL is done in the action space, exploration in ES is done in the parameter space.

I.e. In ES, we explore by changing the whole parameter $\theta$ (not just by making actions more random), potentially leading to a new behaviour.
Genetic Algorithms for Optimizing RL Problems
Definition (Genetic Algorithms)

initialize parameter vectors $\theta_1^{(0)}, ..., \theta_n^{(0)}$

repeat for $t$ in $\{1, ..., T\}$:

- Generate a set of samples $D = \{(\theta_1^{(t)}, F(\theta_1^{(t)})), ..., (\theta_n^{(t)}, F(\theta_n^{(t)}))\}$
- Evaluate the fitness of samples in $D$
- Select the $B$ top performing parameter vectors; they become the parents
- Mutate and/or breed the parents in some way to get new parameters $\theta_1^{(t+1)}, ..., \theta_n^{(t+1)}$
We want to find the minimum of 
\[ F(\theta) = \text{distance of } \theta \text{ to } + \]

\( o = \text{child points} \)
\( o = \text{best performing points / parents of the next generation} \)
initialize parameter vectors $\theta_1^{(0)}, \ldots, \theta_n^{(0)}$, the size of the elite set $m$, standard deviation $\sigma$

repeat for $t$ in $\{1, \ldots, T\}$:
- For each $\theta_i^{(t)}$ do 30 runs in the environment with the policy $\pi(\theta_i^{(t)})$ and store the mean of the cumulative rewards $G_n$.
- Let $L$ be the set of the $m$ parameters with the highest mean of rewards.
- for $i$ in $\{1, \ldots, n\}$:
  - choose $\theta$ in $L$ uniformly at random
  - Define $\theta_i^{(t+1)} = \theta + \sigma \epsilon$, where $\epsilon \sim N(0, 1)$
Advantages of ES and GA

Same advantages as ES:

- Efficient parallelization
- Computation and memory efficient since no backpropagation is needed
- Robust: little hyperparameter tuning needed, no frameskip needed
- Easy to implement
Experiments

GA, ES run significantly faster than DQN and A3C. GA outperformed the other methods on frostbite, skiing and venture.

We see: Each method outperforms all other methods on at least one game.

When RS performs well, so does GA.

<table>
<thead>
<tr>
<th>Game</th>
<th>Frames</th>
<th>Time</th>
<th>Forward Passes</th>
<th>Backward Passes</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>amidar</td>
<td>DQN 200M</td>
<td>~7-10d</td>
<td>450M</td>
<td>1.25B U</td>
<td></td>
</tr>
<tr>
<td>assault</td>
<td>ES 1B</td>
<td>~1h</td>
<td>250M</td>
<td>250M U</td>
<td></td>
</tr>
<tr>
<td>asterix</td>
<td>A3C 1B</td>
<td>~4d</td>
<td>250M</td>
<td>250M U</td>
<td></td>
</tr>
<tr>
<td>asteroids</td>
<td>RS 1B</td>
<td>~1h or 4h</td>
<td>250M</td>
<td>250M U</td>
<td></td>
</tr>
<tr>
<td>atlantis</td>
<td>GA 1B</td>
<td>~1h or 4h</td>
<td>250M</td>
<td>250M U</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Game</th>
<th>DQN</th>
<th>ES</th>
<th>A3C</th>
<th>RS</th>
<th>GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>amidar</td>
<td>978</td>
<td>112</td>
<td>264</td>
<td>143</td>
<td>263</td>
</tr>
<tr>
<td>assault</td>
<td>4,280</td>
<td>1,674</td>
<td>5,475</td>
<td>649</td>
<td>714</td>
</tr>
<tr>
<td>asterix</td>
<td>4,359</td>
<td>1,440</td>
<td>22,140</td>
<td>1,197</td>
<td>1,850</td>
</tr>
<tr>
<td>asteroids</td>
<td>1,365</td>
<td>1,562</td>
<td>4,475</td>
<td>1,307</td>
<td>1,661</td>
</tr>
<tr>
<td>atlantis</td>
<td>279,987</td>
<td>1,267,410</td>
<td>911,091</td>
<td>26,371</td>
<td>76,273</td>
</tr>
<tr>
<td>enduro</td>
<td>729</td>
<td>95</td>
<td>-82</td>
<td>36</td>
<td>60</td>
</tr>
<tr>
<td>frostbite</td>
<td>797</td>
<td>370</td>
<td>191</td>
<td>1,164</td>
<td>4,536</td>
</tr>
<tr>
<td>gravitar</td>
<td>473</td>
<td>805</td>
<td>304</td>
<td>431</td>
<td>476</td>
</tr>
<tr>
<td>kangaroo</td>
<td>7,259</td>
<td>11,200</td>
<td>94</td>
<td>1,099</td>
<td>3,790</td>
</tr>
<tr>
<td>seaquest</td>
<td>5,861</td>
<td>1,390</td>
<td>2,355</td>
<td>503</td>
<td>798</td>
</tr>
<tr>
<td>skiing</td>
<td>-13,062</td>
<td>-15,443</td>
<td>-10,911</td>
<td>-7,679</td>
<td>-6,502</td>
</tr>
<tr>
<td>venture</td>
<td>163</td>
<td>760</td>
<td>23</td>
<td>488</td>
<td>969</td>
</tr>
<tr>
<td>zaxxon</td>
<td>5,363</td>
<td>6,380</td>
<td>24,622</td>
<td>2,538</td>
<td>6,180</td>
</tr>
</tbody>
</table>
It could be that saddle points or noisy gradients prevent the gradient-based methods from learning effectively in some environments.

Since GA and RS do not use gradients, they are not affected by this.
Some optimization problems can be framed as RL-tasks and then solved with RL methods.

Black Box optimization methods like GA and ES can provide an alternative approach to solve RL problems.
