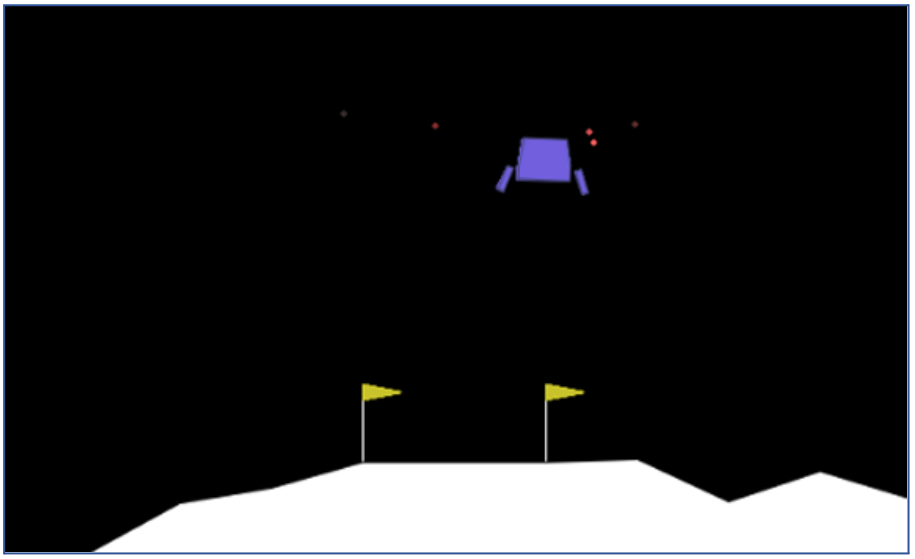
1 2 9 0 FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE D COIMBRA

Fundamentos de Inteligência Artificial

2024/2025 • 2º Semestre

Meta 2 : **Lunar Lander**



Elementos do Grupo (PL4)

Daniel Pereira • 2021237092 • danielpereira@student.dei.uc.pt

Eduardo Marques • 2022231584 • eduardomarques@student.dei.uc.pt

Índice

Introdução

O presente relatório descreve o desenvolvimento e implementação de um agente reativo no ambiente **Lunar Lander**, no âmbito da disciplina de Fundamentos de Inteligência Artificial. O principal objetivo foi desenvolver uma abordagem de neuroevolução com o intuito de aterrar a nave em segurança.

Descrição do Problema

O objetivo deste trabalho é desenvolver um agente autônomo, controlado por uma rede neuronal, capaz de realizar aterragens seguras no ambiente simulado **Lunar Lander**. Este ambiente representa uma nave espacial que deve pousar numa plataforma localizada no centro do cenário, enfrentando forças como a gravidade e, eventualmente, o vento. A tarefa requer que o agente aprenda a controlar com precisão os motores da nave a partir de observações contínuas do estado do ambiente. Para isso, utilizamos um **Algoritmo Evolucionário** que promove a evolução dos parâmetros da rede neuronal, com base numa função objetivo que avalia a qualidade das aterragens.

Implementação

O agente baseado em neuroevolução foi implementado utilizando **Python 3.12** e a **framework Gymnasium**. O código base fornecido foi adaptado para suportar a evolução dos agentes autônomos através de **Algoritmos Evolucionários**. Para isso, foram implementadas diversas funcionalidades, incluindo um mecanismo de seleção de progenitores, os operadores de crossover e de mutação e uma função objetivo personalizada.

Arquitetura

A solução proposta para o controlo do agente no ambiente Lunar Lander baseia-se numa **rede neuronal**, usada como controlador para determinar as ações a realizar em cada instante com base nas observações do ambiente.

A arquitetura da rede é definida por um vetor `SHAPE`, que indica o número de neurónios por camada. No nosso caso, foi utilizada a topologia `(8, 12, 2)`, correspondendo a:

- 8 neurónios de entrada (relativos às 8 variáveis de observação do ambiente),
- 1 camada oculta com 12 neurónios,
- 2 neurónios de saída (valores de controlo dos motores).

A rede aplica a função de ativação **`tanh`** entre camadas, promovendo valores contínuos entre -1 e 1, o que se adequa bem ao espaço de ações contínuo do ambiente.

Cada indivíduo da população representa uma rede com pesos codificados diretamente num vetor de valores reais — o **genótipo**. Estes pesos são os únicos parâmetros que sofrem evoluções pelo algoritmo. A avaliação de cada indivíduo consiste em executar uma simulação e calcular a sua qualidade através de uma função objetivo que considera fatores como posição, velocidade, orientação e sucesso da aterragem.

Algoritmo Evolucionário

O algoritmo utilizado é um **Algoritmo Evolucionário** que tem como objetivo otimizar os pesos da rede neuronal responsável por controlar a nave. A população inicial é composta por indivíduos com genótipos aleatórios (valores entre -1 e 1), e a evolução ocorre ao longo de várias gerações.

Em cada geração, os seguintes passos são realizados:

1. **Seleção de progenitores**: Um subconjunto de indivíduos é escolhido aleatoriamente e o melhor entre eles é selecionado para reprodução.
2. **Crossover**: Dependendo da qualidade dos progenitores, são utilizados diferentes tipos de recombinação, incluindo combinação linear e recombinação uniforme para gerar um novo indivíduo.
3. **Mutação**: A mutação é adaptativa, variando conforme o fitness do indivíduo. Indivíduos com alto desempenho são ligeiramente ajustados através de uma distribuição Gaussiana, enquanto os restantes sofrem mutações mais amplas e aleatórias.
4. **Avaliação**: O novo indivíduo é testado numa simulação e avaliado pela função objetivo, que penaliza desvios em relação a uma aterragem estável e recompensa aterragens bem-sucedidas.

5. **Seleção de sobreviventes**: É utilizado o elitismo, garantindo que os melhores indivíduos da geração anterior sejam preservados, combinando-os com os melhores descendentes para formar a nova população.

Este ciclo repete-se ao longo de um número fixo de gerações. A estrutura modular do código permite ajustar facilmente os operadores genéticos e os parâmetros experimentais, o que foi essencial para realizar as comparações entre diferentes configurações.

Funções Implementadas

`objective_function(observation)`

Esta função avalia a qualidade de cada indivíduo (ou seja, cada rede neuronal) com base no desempenho da nave durante a simulação. Utiliza uma função heurística ponderada que penaliza desvios com diferentes pesos nas seguintes variáveis:

- **x**: posição horizontal da nave;
- **y**: altura da nave;
- **vx**: velocidade horizontal;
- **vy**: velocidade vertical;
- **theta**: inclinação da nave;
- **vtheta**: velocidade angular.

A penalização é subtraída a um valor base (100), resultando no **fitness**. Se a aterragem for bem-sucedida, é atribuído um **bónus adicional de 50 pontos**. Esta abordagem recompensa controladores que garantem estabilidade, precisão e sucesso na aterragem.

`parent_selection(population)`

Este método realiza a **seleção do progenitor**. São escolhidos aleatoriamente sete indivíduos da população atual, e o que tiver maior fitness é selecionado como progenitor. Esta abordagem favorece os indivíduos mais aptos e ao mesmo tempo, mantém a diversidade genética.

`crossover(p1, p2)`

A função de crossover gera um novo indivíduo (filho) a partir de dois progenitores **p1** e **p2**, combinando os seus genótipos com diferentes estratégias conforme o fitness dos mesmos:

- **Ambos têm fitness alto**: é realizada uma combinação linear equilibrada, com 50% dos genes de cada pai.
- **Só um tem fitness alto**: a contribuição genética desse progenitor é aumentada para 85%.
- **Ambos com fitness baixo**: os genes são herdados aleatoriamente entre os pais.

`mutation(p)`

A mutação aplica alterações ao genótipo de um indivíduo, ajustando os seus valores conforme o fitness:

1. Indivíduos com Fitness Elevado (`>= FITNESS_HIGH_THRESHOLD`)

Distribuição Gaussiana (normal) com média 0 e desvio padrão pequeno uma vez que, indivíduos com fitness elevado já representam boas soluções com **características vantajosas**, promovendo apenas alterações **suaves.**

2. Indivíduos com Fitness Médio (`>= FITNESS_LOW_THRESHOLD e < HIGH`)

Distribuição uniforme entre valores moderados (por exemplo `[-2, 2]`). Estes indivíduos demonstram ter potencial, mas ainda apresentam limitações importantes. Ao aplicar uma **mutação uniforme moderada**, procuramos explorar novas regiões do espaço genético que possam revelar melhorias significativas (sem distorções extremas).

3. Indivíduos com Fitness Baixo (`< FITNESS_LOW_THRESHOLD`)

Distribuição uniforme ampla (por exemplo `[-3, 3]`). Quando um indivíduo tem um desempenho muito fraco, é sinal de que está longe de uma solução viável. Neste caso, é mais vantajoso permitir **grandes alterações genéticas**, com o

objetivo de posicionar o indivíduo em outro ponto do espaço de soluções, em busca de regiões de melhor desempenho.

O objetivo é possibilitar **pequenas melhorias nas soluções de alto desempenho** e assegurar **elevada diversidade genética em soluções de baixo desempenho**, promovendo assim a convergência eficaz do algoritmo.

Experiências e Resultados

<table>
<tr><th></th><th>Mutação</th><th>Crossover</th><th>Elitismo</th><th>População</th><th>Gerações</th><th>Média Fitness</th><th>Sucesso (%)</th></tr>
<tr><td>1</td><td>0.008</td><td rowspan="2">0.5</td><td rowspan="4">0</td><td rowspan="8">100</td><td rowspan="8">100</td><td>143,454</td><td>88,0</td></tr>
<tr><td>2</td><td>0.05</td><td>145,084</td><td>91,2</td></tr>
<tr><td>3</td><td>0.008</td><td rowspan="2">0.9</td><td>144,828</td><td>90,1</td></tr>
<tr><td>4</td><td>0.05</td><td>143,801</td><td>88,4</td></tr>
<tr><td>5</td><td>0.008</td><td rowspan="2">0.5</td><td rowspan="4">1</td><td>147,208</td><td>95,7</td></tr>
<tr><td>6</td><td>0.05</td><td>148,730</td><td>97,6</td></tr>
<tr><td>7</td><td>0.008</td><td rowspan="2">0.9</td><td>147,792</td><td>96,0</td></tr>
<tr><td>8</td><td>0.05</td><td>144,427</td><td>90,0</td></tr>
</table>

Notas para preenchimento:

- **Média Fitness**: média dos valores do fitness ao longo de 5 execuções.
- **Sucesso (%)**: média das taxas de sucesso em % dos episódios com aterragem bem-sucedida ao longo de 5 execuções.

Com a introdução do vento, observa-se uma degradação nos resultados. Tanto a **Média Fitness** como o **Sucesso (%)** tendem a diminuir, refletindo a maior dificuldade nas condições de aterragem ao longo das 5 execuções.

Conclusão

Neste trabalho aplicamos um algoritmo evolutivo para treinar controladores no ambiente Lunar Lander, avaliando-os em condições com e sem vento. A estrutura modular do código facilitou a análise dos compromissos, permitindo ajustar facilmente operadores genéticos e parâmetros de rede para maximizar o seu desempenho. Além disso, a função objetivo desenvolvida mostrou-se eficaz na penalização de estados instáveis, enquanto o elitismo garantiu a preservação das melhores soluções ao longo das gerações.