FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D COIMBRA

Fundamentos de Inteligência Artificial

Licenciatura em Engenharia Informática

2025

Relatório | Projeto

The evolution of Lunar Lander

Projeto realizado por:

André Miguel Correia Cardoso | 2022222265

João Filipe Valente Cardoso | 2022222301

Sérgio Miguel Lopes Marques | 2022222096

Introdução

O presente relatório documenta o desenvolvimento e implementação de uma abordagem baseada em neuroevolução para o controlo de aterragem de uma nave espacial no ambiente de simulação Lunar Lander, no âmbito da unidade curricular de Fundamentos de Inteligência Artificial. A neuroevolução representa uma abordagem que combina redes neuronais com algoritmos evolutivos, permitindo a otimização dos parâmetros da rede através de mecanismos inspirados na evolução natural. Neste projeto, o objetivo principal consistiu no desenvolvimento de um agente capaz de controlar os motores de uma nave lunar, de forma a garantir uma aterragem segura e precisa na plataforma designada. Para isso, o agente terá que aprender a controlar os motores da nave, a partir das observações contínuas do ambiente. Esta aprendizagem foi desenvolvida nas funções implementadas: função objetivo, parent selection, crossover e mutation.

Ao longo deste relatório, são apresentadas as decisões de implementação tomadas, os resultados experimentais obtidos e uma análise crítica do desempenho dos agentes evoluídos, bem como as conclusões e aprendizagens resultantes do projeto.

Arquitetura da Rede Neuronal

A rede neuronal utilizada é composta por uma camada de entrada, uma camada escondida e uma camada de saída.

- **Camada de entrada**: 8 neurónios, correspondentes aos parâmetros de observação do ambiente.
- **Camada escondida**: 12 neurónios, com função de ativação tangente hiperbólica.
- **Camada de saída**: 2 neurónios, correspondentes às ações de controlo dos motores.

Esta estrutura é definida através do vetor SHAPE = (nInputs, 12, nOutputs), onde nInputs = 8 e nOutputs = 2.

Representação dos Indivíduos

No algoritmo evolucionário desenvolvido, cada indivíduo é representado por um dicionário com duas componentes:

- **genotype**: lista de números reais que representa os pesos sinápticos da rede neuronal.

- **fitness:** valor real que representa a qualidade da solução, calculado pela função objetivo.

Algoritmo Evolucionário

O algoritmo utilizado tem como objetivo otimizar os pesos da rede neuronal. A população inicial é composta por indivíduos com genótipos entre -1 e 1 e a evolução ocorre ao longo de várias gerações.

Em cada geração, é realizado o seguinte:

1. **Seleção de progenitores:** É feita uma seleção do indivíduo com melhor fitness de um subconjunto da população.

2. **Crossover:** Cruzamento de dois indivíduos, gerando, assim, um novo, com características de ambos. A técnica utilizada varia consoante o nível de fitness de cada pai.

3. **Mutação:** Utilizada mutação gaussiana para indivíduos com um fitness muito bom e mutação uniforme para indivíduos com valores inferiores.

4. **Avaliação:** O indivíduo recentemente criado é testado numa simulação e é realizada uma avaliação ao mesmo pela função objetivo.

5. **Seleção de sobreviventes:** Garante-se que os melhores indivíduos da geração anterior sejam mantidos, de modo a combiná-los com os melhores descendentes para formar uma nova população.

Funções do Algoritmo Evolucionário Implementadas

objective_function(observation):

A função objective_function() avalia a qualidade do indivíduo (o seu fitness) com base nas observações da nave, como por exemplo: a posição da nave na simulação, a velocidade vertical e a velocidade horizontal, o ângulo da nave.

Para isso, atribuímos um peso a cada uma das observações que considerámos importantes analisar, com o intuito de penalizar os indivíduos que se afastem da solução pretendida. A escolha de cada um destes pesos foi inspirada na solução desenvolvida durante o primeiro trabalho prático. À posição da nave no eixo x, atribuímos o maior peso, uma vez que pensamos que esta é a medição com maior importância, já que é pretendido que a nave aterre dentro dos limites definidos pelas bandeiras. As velocidades (horizontal e vertical) e o ângulo de inclinação da nave, também foram considerados como sendo mais importantes na aterragem controlada da nave, com um peso ligeiramente maior.

Para todos os indivíduos, definimos como fitness base uma pontuação de 50 pontos, aos quais vamos retirar as penalizações calculadas anteriormente. Posteriormente, vamos verificar se a aterragem da nave foi bem sucedida. Caso isso aconteça, o indivíduo recebe uma pontuação bónus de 25 pontos, podendo chegar a um máximo teórico de 75 pontos.

```python
def objective_function(observation):

    x = observation[0]
    y = observation[1]
    vx = observation[2]
    vy = observation[3]
    theta = observation[4]
    vtheta = observation[5]

    weight_x = 3.0
    weight_y = 1.0
    weight_vx = 2.0
    weight_vy = 2.0
    weight_theta = 2.0
    weight_vtheta = 1.0

    penalty = 0
    penalty += weight_x * abs(x)
    penalty += weight_y * abs(y)
    penalty += weight_vx * abs(vx)
    penalty += weight_vy * abs(vy)
    penalty += weight_theta * abs(theta)
    penalty += weight_vtheta * abs(vtheta)

    fitness = 50 - penalty

    # Verificar se aterrou com sucesso
    landing_success = check_successful_landing(observation)

    # Bónus para aterragem bem-sucedida
    reward = 0
    reward += 25 if landing_success else 0

    fitness += reward

    return fitness, landing_success
```

parent_selection(population):

Esta função tem como objetivo escolher dois indivíduos da população (de preferência aqueles com melhor fitness) para, posteriormente, criar um indivíduo novo que adote comportamentos de cada um dos pais, que pretendemos replicar. Ou seja, se dois indivíduos conseguirem ter uma aterragem bem sucedida, o nosso objetivo é criar um indivíduo que conjugue os “bons” comportamentos de ambos os pais.

```python
def parent_selection(population):

    tournament_size = 5

    candidates = random.sample(population, tournament_size)

    valid_population = []
    for ind in candidates:
        if ind['fitness'] is not None:
            valid_population.append(ind)

    if len(valid_population) == 0:
        return copy.deepcopy(random.choice(candidates))

    sum_fitness = 0
    for ind in valid_population:
        sum_fitness += ind['fitness']

    if sum_fitness == 0:
        return copy.deepcopy(random.choice(candidates))

    valid_population.sort(key=lambda ind: ind['fitness'], reverse=True)

    return copy.deepcopy(valid_population[0])
```

Na nossa parent_selection(), decidimos escolher para seleção 6 indivíduos da população, uma vez que números inferiores, no fim da evolução, podiam gerar resultados indesejados, já que os selecionados podiam ser uns dos poucos indivíduos com um fitness baixo. Escolhendo um número superior, poderia acontecer o contrário, no início da evolução, uma vez que a probabilidade de a maior parte ter um fitness baixo é grande.

Depois de escolhidos os indivíduos, verificamos se não há nenhuma irregularidade no fitness de cada um e, de seguida, ordenamos os indivíduos por valor de fitness, escolhendo aquele com um valor de fitness maior.

crossover(p1, p2):

```python
def crossover(p1, p2):

    child = {'genotype': [], 'fitness': None}

    if p1['fitness'] >= FITNESS_THRESHOLD and p2['fitness'] >= FITNESS_THRESHOLD:
        for gene1, gene2 in zip(p1['genotype'], p2['genotype']):
            gene = gene1 * 0.5 + gene2 * 0.5
            child['genotype'].append(gene)

    elif p1['fitness'] >= FITNESS_THRESHOLD:
        for gene1, gene2 in zip(p1['genotype'], p2['genotype']):
            gene = gene1 * 0.9 + gene2 * 0.1
            child['genotype'].append(gene)

    elif p2['fitness'] >= FITNESS_THRESHOLD:
        for gene1, gene2 in zip(p1['genotype'], p2['genotype']):
            gene = gene1 * 0.1 + gene2 * 0.9
            child['genotype'].append(gene)

    else:
        for gene1, gene2 in zip(p1['genotype'], p2['genotype']):
            if random.random() > 0.5:
                child['genotype'].append(gene1)
            else:
                child['genotype'].append(gene2)

    return child
```

Nesta função, é realizado o cruzamento entre os dois indivíduos previamente selecionados, tendo em conta o fitness de cada um:

Se ambos os indivíduos tiverem um valor de fitness alto, ou seja, maior do que FITNESS_THRESHOLD, cada gene do indivíduo filho terá metade da informação do gene de um pai e a outra metade virá do gene do outro pai.

Caso apenas o primeiro pai ou apenas o segundo pai tenha um valor de fitness alto, ou seja, maior que FITNESS_THRESHOLD, os genes do filho irão ter 90% da informação vinda do pai “bom” e 10% do pai “mau”, com o intuito de preservar e replicar os comportamentos do pai com o valor de fitness alto.

Se nenhum dos pais for “bom”, e o valor do fitness de cada um for menor que FITNESS_THRESHOLD, a origem dos genes dos filhos será escolhida aleatoriamente.

mutation(p):

Na função de mutação do nosso algoritmo evolucionário, aplicamos uma mutação diferente a cada indivíduo, consoante o seu nível de fitness:

```python
def mutation(p):

    mutated = copy.deepcopy(p)

    current_fitness = p.get('fitness', None)

    if current_fitness is not None and current_fitness >= FITNESS_THRESHOLD + 4:
        for i in range(len(mutated['genotype'])):
            if random.random() < PROB_MUTATION:
                mutated['genotype'][i] += random.gauss(0, STD_DEV)
                mutated['genotype'][i] = max(-1.0, min(1.0, mutated['genotype'][i]))

    elif current_fitness is not None and current_fitness >= FITNESS_THRESHOLD and current_fitness < FITNESS_THRESHOLD + 4:
        for i in range(len(mutated['genotype'])):
            if random.random() < PROB_MUTATION:
                mutated['genotype'][i] = random.uniform(-2.0, 2.0)
    else:
        for i in range(len(mutated['genotype'])):
            if random.random() < PROB_MUTATION:
                mutated['genotype'][i] = random.uniform(-3.0, 3.0)

    mutated['fitness'] = None

    return mutated
```

Se o valor de fitness do indivíduo for muito bom, ou seja, maior do que FITNESS_MUITO_BOM, será aplicada uma mutação gaussiana suave, que irá preservar as características boas do indivíduo.

Se o valor de fitness for bom, ou seja, se estiver entre FITNESS_THRESHOLD e FITNESS_MUITO_BOM, é utilizada uma mutação uniforme com valores de genótipo normais (-2.0 - 2.0), dentro dos valores esperados.

Caso o fitness do indivíduo seja muito baixo, ou seja, menor que FITNESS_THRESHOLD, será aplicada uma mutação uniforme com valores mais afastados daqueles que são expectáveis (-3.0 - 3.0).

Experiências e Análise dos Resultados

Depois de serem testadas várias soluções e de serem efetuados ajustes às funções implementadas, obtivemos uma solução com uma taxa de sucesso para todos os testes bastante positiva.

	Mutação	Crossover	Elitismo	**Média de Fitness**	**Média % de Sucesso**
Experiência 1	0.008	0.5	0	72,611614	91,54
Experiência 2	0.05			71,760144	89,56
Experiência 3	0.008	0.9		72,085406	90,42
Experiência 4	0.05			72,013264	89,66
Experiência 5	0.008	0.5	1	73,835778	96,04
Experiência 6	0.05			73,260328	94,28
Experiência 7	0.008	0.9		73,547891	95,57
Experiência 8	0.05			73,387814	94,54

Conclusão

A implementação do algoritmo evolutivo, com componentes como seleção por torneio, cruzamento adaptativo e mutação dependente da aptidão, revelou-se eficaz na evolução de controladores capazes de realizar aterragens bem-sucedidas.

Os resultados experimentais demonstraram que a parametrização do algoritmo tem um impacto significativo no desempenho dos agentes evoluídos. Em particular, a probabilidade de mutação de 0.008 combinada com uma probabilidade de cruzamento de 0.5 e um elitismo de 1 produziu os melhores resultados nas condições sem vento. Nos testes com presença de vento reparámos que as taxas de sucesso diminuem significativamente.