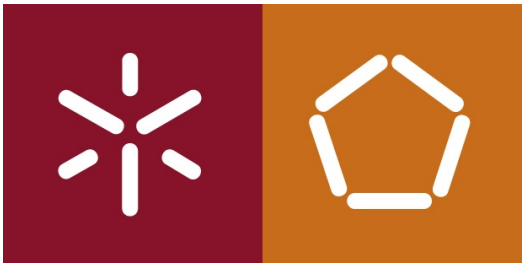

UNIVERSIDADE DO MINHO

MESTRADO EM ENGENHARIA INFORMÁTICA

PERFIL: SISTEMAS INTELIGENTES

COMPUTAÇÃO NATURAL

TP2

REINFORCEMENT LEARNING

Autores:
Diogo Rodrigues (PG37150
Luís Bouça (PG38933)
Henrique Ribeiro (PG38415)
Hugo Gonçalves (PG38928)

19 de Maio de 2019

Conteúdo

1 Introdução

A construção de um modelo de **Reinforcement Learning**, que tenha a capacidade para jogar uma versão simplificada do jogo pacman, tem como objetivo dar a perceber a forma de como estes modelos funcionam, e de como podem ser adaptados na tentativa de obter uma solução ótima.

A implementação do modelo, e a sua adaptação ao nosso problema, implica que seja necessário ter em conta algumas estratégias que nos permitam tirar o melhor proveito possível.
Neste caso, a escolha de parâmetros como a taxa de aprendizagem, probabilidade de exploração ou taxa de desconto, vão influenciar a forma de como de como o treino é feito, sendo assim necessário realizar algum tipo de otimização que possibilite obter os parâmetros que melhor se adaptem.

A otimização do modelo foi feita de forma recursiva, com recurso a **Algoritmos Genéticos** e a **Enxame de partículas**. Existe assim uma possibilidade de poder comparar estes estes dois métodos, que apesar de possuírem um principio base semelhante, a sua forma de convergir é diferente.
Por fim serão apresentados alguns resultados e conclusões provenientes da otimização por parte de ambos os modelos, e a forma de como os parâmetros obtidos influenciam o ambiente em que o agente está inserido.

2 Ambiente de Jogo

Figura 1: Jogo Pacman

A implementação de um modelo de **Reinformecent Learning**, com a capacidade de guiar as ações de um agente inserido num ambiente/mapa.

Neste caso, o agente, inserido no mapa smallGrid do jogo pacman, conforme apresentado na figura 1, deve ter a capacidade de fugir sem ser apanhado, e chegar ao estado final com o maior score possível, obtido através de recompensas que foi obtendo após as ações que foi tomando.

Um ambiente de pequenas dimensões, como é o caso do smallGrid, significa que existe um conjunto menor de ações a serem tomadas pelo agente, ou seja, existe a possibilidade de convergir mais rapidamente para uma solução ótima, e de não estar tão dependente de uma estratégia que se baseie na exploração do ambiente.

3 Reinforcement Learning

O modelo de **Reinforcement Learning** tem como base a matriz Q-table, que faz a correspondência entre o conjunto de ações possíveis, e os estados existentes no ambiente em que o agente está inserido. Esta matriz é inicializada a 0, indicando assim que o agente está num estado inicial, e ainda não possui qualquer conhecimento do ambiente.

Os valores da matriz ou Q-values, vão sendo atualizados à medida que o agente toma ações a partir do seu estado atual.

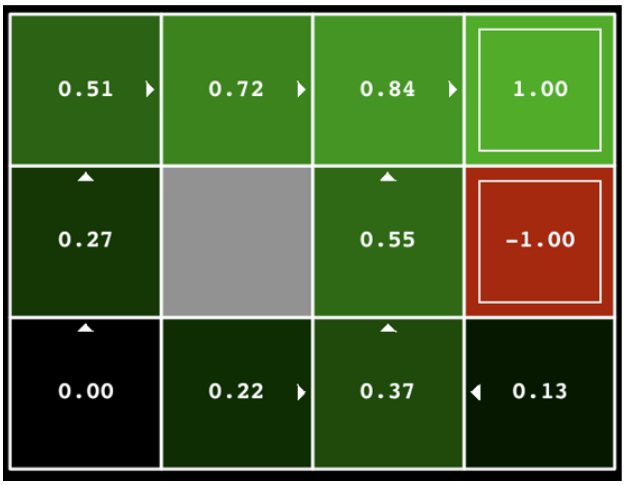

Figura 2: Q-table

A figura 2, é um exemplo dos qvalues atualizados após um determinado conjunto de ações.

As ações tomadas pelo agente a partir de um qualquer estado são avaliadas por recompensas, que podem ser ou não benéficas para o agente.

$$Q[s, a] = Q[s, a] + alpha*(R + gamma*Max[Q(s', A)] - Q[s, a])$$

Figura 3: Q-value

A figura 3, mostra a formula para a atualização do Q-value, que tem em conta o valor atual do Q-value, assim como o valor de um determinado Q-value, com base na transação do próximo estado e ação, e alguns parâmetros que vão influenciar o valor final do Q-value, como o gamma e a recompensa, e a taxa de aprendizagem.

O valor de gamma ou fator de desconto, é um parâmetro que varia entre [0-1], e irá determinar a importância que o agente dá às recompensas futuras. No caso de gamma = 0, significa que o agente apenas tem em conta as recompensas imediatas.

O alpha, é referente à taxa de aprendizagem, e influencia diretamente o produto da multiplicação com todos os outros fatores acima descritos.

A escolha das ações, são feitas com base na estratégia de aprendizagem que foi definida para o nosso modelo. É neste processo que o parâmetro epsilon, vai influenciar se o modelo vai seguir uma estratégia que premia a exploração do ambiente (**exploration**), em que o objetivo é obter o maior conhecimento e informação possível do ambiente, ou se segue

uma estratégia que tira proveito do ambiente, obtendo assim recompensas rápidas(**exploitation**).Independentemente do ambiente, o agente deve seguir ambas estratégias ainda que de forma parcial, permitindo assim que o mesmo explore o ambiente e que se adapte a alterações sofridas nele.

3.1 Função getAction

```
def getAction(self, state):
    """
      Compute the action to take in the current state.  With
      probability self.epsilon, we should take a random action and
      take the best policy action otherwise.  Note that if there are
      no legal actions, which is the case at the terminal state, you
      should choose None as the action.

      HINT: You might want to use util.flipCoin(prob)
      HINT: To pick randomly from a list, use random.choice(list)
    """
    # Pick Action
    legalActions = self.getLegalActions(state)
    action = None
    "*** YOUR CODE HERE ***"
    #util.raiseNotDefined()
    if len(legalActions) != 0: #if there are no legal actions, returns action none
      if util.flipCoin(self.epsilon) == True: #getting probability of exploration.
        action = random.choice(legalActions) #getting randomly actions
      else:
        action = self.getPolicy(state)
    return action
```

Figura 4: Função getAction

A figura 4, mostra o excerto de código implementado, referente à função **getAction** do ficheiro **qlearningAgents.py**.
Numa fase inicial é obtido o conjunto de ações possíveis de escolher no estado atual.
A escolha da ação vai ser feita mediante o parâmetro epsilon, pois a sua probabilidade irá determinar se a ação vai ser escolhida de forma aleatória de entre todo o conjunto de ações, com o objetivo de explorar o ambiente, ou se vai ser escolhida a ação que maior Q-value tem face ao estado atual, seguindo assim uma política ótima.

3.2 Função getPolicy

```
def computeActionFromQValues(self, state):
    """
      Compute the best action to take in a state.  Note that if there
      are no legal actions, which is the case at the terminal state,
      you should return None.
    """
    "*** YOUR CODE HERE ***"
    #util.raiseNotDefined()
    action = None
    legalActions = self.getLegalActions(state)
    if len(legalActions) > 0:
      action, value = self.max_dict(state, legalActions)
    #util.raiseNotDefined()
    return action
```

Figura 5: Função getPolicy

A função **computeActionFromQValues**, apresentada na figura 5, é chamada pela função **getPolicy** acima descrita. Esta função é responsável por obter a melhor ação possível a realizar mediante o estado atual, sendo uma sub-chamada à função **max_dict**, que foi adicionada a este ficheiro **qlearningAgents.py**.

3.3 Função max_dict

```
def max_dict(self,state, legalActions):
    max_key = None
    max_val = float('-inf')
    for action in legalActions:
      qvalue = self.getQValue(state, action) #Gets de q-value pair from this state action.
      if qvalue > max_val:
        max_val = qvalue
        max_key = action

    return max_key, max_val
```

Figura 6: Função max_dict

A figura 6, apresenta a função **max_dict**, responsável por obter os maiores valores possíveis de Q-value, existentes na Q-table, realizando para isso consultas do tipo par <Chave, Valor>, ou seja, a Q-table irá funcionar como um dicionário.

As consultas vão ser feitas recorrendo à função **getQValue**, que vai ser descrita posteriormente neste documento.

3.4 Função getQValue

```
def getQValue(self, state, action):
    """
      Returns Q(state,action)
      Should return 0.0 if we have never seen a state
      or the Q node value otherwise
    """
    "*** YOUR CODE HERE ***"
    #util.raiseNotDefined()
    return self.Q[(state, action)]
```

Figura 7: Função getQValue

A função getQValue, representa uma consulta de dicionário utilizando chave valor, que neste caso representam o estado ação, retornando assim o valor desse mesmo Q-value.

3.5 Função getValue

```
def computeValueFromQValues(self, state):
    """
      Returns max_action Q(state,action)
      where the max is over legal actions.  Note that if
      there are no legal actions, which is the case at the
      terminal state, you should return a value of 0.0.
    """
    "*** YOUR CODE HERE ***"

    #util.raiseNotDefined()
    value = 0.0 #Setting up default value
    legalActions = self.getLegalActions(state)
    if len(legalActions) > 0:
      action, value = self.max_dict(state, legalActions)
    #util.raiseNotDefined()
    return value #Return value or 0.0 if there are no legal actions.
```

Figura 8: Função getValue

Semelhante à lógica implementada na função **getPolicy**, mas neste caso pretendemos retornar o maior Q-value associado a uma determinada ação, num dado estado.

3.6 Função Update

```
def update(self, state, action, nextState, reward):
    """
      The parent class calls this to observe a
      state = action => nextState and reward transition.
      You should do your Q-Value update here

      NOTE: You should never call this function,
      it will be called on your behalf
    """
    "*** YOUR CODE HERE ***"
    #util.raiseNotDefined()
    qvalue = self.getQValue(state, action) #old q-value

    max_value = self.getValue(nextState)

    qvalue = qvalue + self.alpha * (reward + self.discount * max_value - qvalue) #Updating qvalue

    self.Q[(state, action)] = qvalue

    return qvalue
```

Figura 9: Função update

Conforme referido anteriormente, o update dos Q-values é a base de algoritmo de **Reinforcemente Learning**. Neste caso é feita uma atualização ao valor de Q-value atual, usando para isso a formula representada na figura 3.

Inicialmente é feita uma chamada ao método **getQValue**, como forma de obter o Q-Value atual, e que vai ser utilizado na formula descrita.

O maior valor do Q-value obtido no próximo estado, é obtido com a função **getValue** descrito anteriormente.

4 Enxame de Partículas

O uso do algoritmo de enxame de partículas num contexto de otimização, tem como objetivo a escolha dos parâmetros que melhor se adaptam ao problema que estamos a tentar modelar com recurso à aprendizagem por reforço.

Os parâmetros em questão, são referentes ao alpha, gamma, epsilon, que são parâmetros essenciais em todo o processo de treino, conforme foi referido anteriormente.

O objetivo deste algoritmo, é de ter um conjunto de elementos suficientemente capazes, de no seu todo chegar a uma solução ótima. Para isso

cada um dos elementos deve fazer uma busca independente e relacionar-se com os restantes, de modo a que todos sigam uma política comum.

De maneira semelhante aos algoritmos genéticos, estes tentam simular o mundo animal, como o movimento de peixes ou pássaros, e a forma de como essas espécies se relacionam entre si.

FOR each particle i
  FOR each dimension d
    Initialize position x_{id} randomly within permissible range
    Initialize velocity v_{id} randomly within permissible range
  End FOR
END FOR
Iteration k=1
DO
  FOR each particle i
    Calculate fitness value
    IF the fitness value is better than p_best_{id} in history
      Set current fitness value as the p_best_{id}
    END IF
  END FOR
  Choose the particle having the best fitness value as the g_best_d
  FOR each particle i
    FOR each dimension d
      Calculate velocity according to the equation
      $v_{id}(k+1) = w\ v_{id}(k) + c_1\ rand_1(p_{id}-x_{id}) + c_2\ rand_2\ (p_{gd}-x_{id})$
      Update particle position according to the equation
      $x_{id}(k+1) = x_{id}(k) + v_{id}(k+1)$
    END FOR
  END FOR
k=k+1
WHILE maximum iterations or minimum error criteria are not attained

Figura 10: Algoritmo Particle Swarm

A figura 10, apresenta o algoritmo contendo a ideia base por trás de um algoritmo de enxame de partículas.

Inicialmente é necessário ter em conta a quantidade de partículas que compõem o nosso exame, pois dependendo do problema, podemos convergir ou não mais rapidamente para uma solução ótima, no entanto o tempo despendido na busca dessa solução pode ou não ser mais demoroso.

Após definirmos a quantidade de partículas, é necessário definir uma posição e velocidade inicial.

A inicialização dos vetores numa fase inicial, vai permitir que na primeira iteração do algoritmo, o fitness de cada uma das partículas seja calculado, e o melhor local e global seja atualizado.

Finalmente é necessário atualizar a posição e velocidade em cada uma das partículas, valores que são influenciados por alguns parâmetros que relevam critérios de sociabilidade para com as restantes partículas.

$$v_{k+1}^i = w_k v_k^i + c_1 r_1 \left(p_k^i - x_k^i\right) + c_2 r_2 \left(p_k^g - x_k^i\right)$$

Figura 11: Atualização da velocidade

A figura 11, apresenta a formula para a atualização da velocidade de uma partícula no vetor de velocidades. A atualização da nova velocidade, implica que sejam definidos previamente os seguintes parâmetros:

- **w**: Parâmetro que influencia a propagação do movimento dado pelo último valor da velocidade;
- **c1**: Importância dada ao valor de fitness local;
- **c2**: Importância dada ao valor de fitness global;
- **r1**: Parâmetro que controla a influencia individual relativa a c1;
- **r2**: Parâmetro que controla a influencia social relativa a c2;

A nova velocidade é assim calculada com base na velocidade e posição atual, assim como fitness local e global.

$$x_{k+1}^i = x_k^i + v_{k+1}^i$$

Figura 12: Atualização da posição

A atualização da nova posição vai ser a posição atual juntamnete com a nova velocidade calculada anteriormente, conforme mostra a figura 12.

O processo de otimização irá terminar no caso de todas as iterações terem sido percorridas, no entanto isto não vai garantir que uma solução ótima tenha sido encontrada. Opcionalmente é possível adicionar um segundo critério de paragem, baseado no erro, ou seja, defini-se um threshold a partir do qual o processo iterativo irá terminar no caso do erro ser relativamente baixo. No entanto, é preciso ter em conta que isto pode levar a casos de o processo iterativo ser parado de forma precoce. Uma paragem precoce pode significar que a solução encontrada não é a mais próxima da solução ideal para o problema.

4.1 Inicialização dos vetores

```
#initial particle positions
particle_position_vector = np.random.random((n_particles,3)).round(decimals=2)

pbest_position = particle_position_vector   #initial pbest positions of each particles
pbest_fitness_value = np.zeros((n_particles,), dtype=float)

gbest_fitness_value = 0.0  #initial global best fitness within all particles
gbest_position = np.zeros((3,), dtype=float)

velocity_vector = np.zeros((n_particles, 3), dtype=float)
```

Figura 13: Inicializar posição/velocidade

A figura 13, mostra o excerto de código referente à inicialização dos vetores de posição e de velocidades, assim como os valores do melhor local e global.

O vetor de posições é composto por 3 posições referentes aos 3 hyperparâmetros que pretendemos encontrar alpha, gamma, epsilon, referentes ao modelo de aprendizagem por reforço.

O vetor de velocidades tem tamanho igual ao número de partículas existentes, e vai ser inicializado com os valores a 0, sendo posteriormente atualizados ao longo do processo iterativo.

4.2 Função de aptidão

```
def fitness(position):

    parameters = "-a epsilon="+str(position[0])+",alpha="+str(position[1])+",gamma="+str(position[2])
    byteOutput = subprocess.check_output("python pacman.py -q -p PacmanQAgent -x 2000 -n 2010 -l smallGrid "+parameters, shell=True
    strOutput = byteOutput.decode('UTF-8').rstrip()
    match = re.search(r"\(([0-9]\.[0-9][0-9])\)", strOutput)  #Regular expression to extract the accuracy from command output
    accuracy = match.group(1)
    return float(accuracy)
```

Figura 14: Fitness

A figura 14, apresenta o código referente à função de aptidão, que vai ser chamada em cada iteração, por cada uma das partículas existentes.

O objetivo desta função é injetar os parâmetros recebidos de cada partícula no modelo de aprendizagem por reforço, e retornar a accuracy associada.

4.3 Atualização do pbest e gbest

```
for i in range(n_particles):
    particle_fitness = fitness(particle_position_vector[i])
    print(particle_fitness, ' ', particle_position_vector[i])

    if(pbest_fitness_value[i] < particle_fitness):
        pbest_fitness_value[i] = particle_fitness
        pbest_position[i] = particle_position_vector[i]

    if(gbest_fitness_value < particle_fitness):
        gbest_fitness_value = particle_fitness
        gbest_position = particle_position_vector[i]
```

Figura 15: Melhor local/global

A atualização do pbest e gbest é feita após cada chamada à função fitness, em cada umas das particulas.

O pbest é atualizado sempre que o valor de fitness atual, seja maior que o seu valor atual existente no array, enquanto que o gbest é atualizado sempre que um pbest for maior que o gbest existente.

4.4 Atualização da velocidade e posição

```
for i in range(n_particles):
    #rand1 and rand2 are random numbers between 0 and 1, and they control the influence of each value: Social and individual
    rand1 = random.random()
    rand2 = random.random()

    #calculating the new position and the new velocity
    new_velocity = (W*velocity_vector[i]) + (c1*rand1) * (pbest_position[i] - particle_position_vector[i]) + (c2*rand2) * (gbest_position-particle_position_vector[i])
    new_position = new_velocity + particle_position_vector[i]
```

Figura 16: Nova velocidade/posição

A nova velocidade/posição vai ser atualizada em cada uma das partículas, respeitando a formula descrita previamente. No entanto é necessário definir previamente parâmetros como **c1**, **c2** e **w**.

5 Algoritmos Genéticos

O uso de algoritmos genéticos, num contexto de otimização do modelo de aprendizagem por reforço, será num termo de comparação com o algoritmo de enxame de partículas.

Este algoritmo baseia-se na teoria da evolução de Charles Darwin, na tentativa de procurar uma solução otimizada para o problema que estamos a tentar solucionar.

De maneira semelhante ao exame de partículas, este algoritmo baseia-se na tentativa de procurar soluções recorrendo aos vários elementos da população que foi definida no algoritmo. No entanto a sua forma de convergir e os problemas que pode ocorrer neste tipo de algoritmos são distintos dos existentes no exame de partículas.

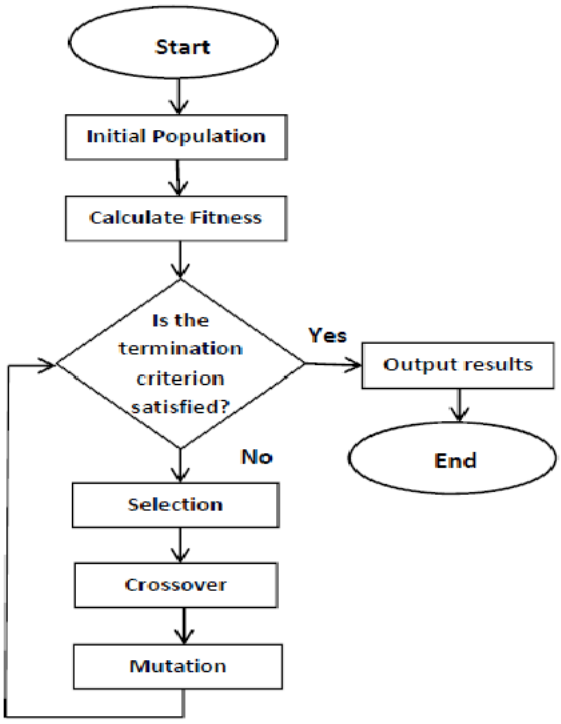

Figura 17: Diagrama de Fluxo

A figura 17, é um diagrama de fluxo que tenta representar o funcionamento dos algoritmos genéticos.

Conforme foi referido anteriormente, inicialmente é definida uma população inicial, em que cada um dos elementos é responsável por tentar procurar uma solução ótima para o problema. A solução de cada um dos elementos da população será posteriormente avaliada, determinando assim se vale a pena continuar o processo de otimização.

O processo de seleção visa a seleção dos melhores elementos da população com o objetivo de os cruzar e gerar novos descendentes.

Com o objetivo de manter uma diversidade sobre toda a população, é aplicado um processo de mutação sobre os descendentes, permitindo assim que a busca por uma solução não fique estagnada.

A nova população, contendo os pais mais aptos e seus descentes que sofreram a mutação, serão avaliados pela função de aptidão, iniciando-se assim um novo ciclo.

Tal como o exame de partículas, os algoritmos genéticos podem possuir dois critérios de paragem. Um que se baseie num número de gerações que devem ser percorridas e outro que se baseie num threshold a partir do

qual o erro é bastante significativo para o problema que estamos a tratar.

Um critério de paragem baseado apenas no threshold do erro, pode levar a uma paragem precoce do algoritmo, pelo que a sua utilização juntamente com um processo de backtracking, sobre o qual o resultado da geração atual era comparado com a de gerações anteriores, seria o mais aconselhável.

5.1 Definindo a população

```
i = 0
while i < max_population:
    chromosome = []
    #Setting chromosome genes
    chromosome.append(round(np.random.random(),2)) #epsilon
    chromosome.append(round(np.random.random(),2)) #alpha
    chromosome.append(round(np.random.random(),2)) #gamma
    population.append(chromosome)
    i += 1
```

Figura 18: População inicial

Cada elemento da população vai ser composto por um cromossoma, composto por três genes referentes aos hyper-parâmetros alpha, gamma, epsilon, que deverão ser gerados de forma aleatória numa fase inicial.

5.2 Função de aptidão

```
def Fitness(population):
    global max_population

    accuracys = []
    i = 0
    while i < max_population:
        parameters = population[i]

        cmd_parameters = "-a epsilon="+str(parameters[0])+",alpha="+str(parameters[1])+",gamma="+str(parameters[2])
        byteOutput = subprocess.check_output("python pacman.py -q -p PacmanQAgent -x 2000 -n 2010 -l smallGrid "+cmd_parameters, shell=True)
        strOutput = byteOutput.decode('UTF-8').rstrip()
        match = re.search(r"\(([0-9]\.[0-9][0-9])\)", strOutput)  #Regular expression to extract the accuracy from command output
        accuracy = match.group(1)
        accuracys.append(float(accuracy))

        i += 1

    return accuracys
```

Figura 19: Fitness

De maneira semelhante à função fitness implementada no enxame de partículas, objetivo é injetar os hyper-parâmetros de cada elemento da população no modelo de aprendizagem por reforço, e devolver o resultado da accuracy para uma análise posterior.

5.3 Definindo os pais

```
i = 0
while i < max_parents:
    max_fitness_idx = np.where(fitness == np.max(fitness))
    max_fitness_idx = max_fitness_idx[0][0]

    chromosome = []
    chromosome.append(population[max_fitness_idx][0])
    chromosome.append(population[max_fitness_idx][1])
    chromosome.append(population[max_fitness_idx][2])
    parents.append(chromosome)

    fitness = np.delete(fitness, max_fitness_idx, 0)
    i += 1
```

Figura 20: Escolha dos mais aptos

A escolha dos pais mais aptos, tem como objetivo definir aqueles que vão fazer cruzamento entre si, de forma a gerar novos descendentes para população.

Neste caso a escolha desses elementos, é feita com base nos elementos mais aptos, ou seja, aqueles que obtiveram os valores mais elevados de toda a população.

O número de elementos escolhidos para o cruzamento varia consoante o problema que estamos a tratar, no entanto é preciso ter em conta que definindo um elevado número de elementos pode levar a um decréscimo na otimização, isto porque corremos o risco de escolher elementos com pouca aptidão.

5.4 Cruzamento

```
crossover_point = len(parents[0])/2

offsprings = []

i = 0
while i < max_offspring:
    #selecting parents to crossover
    parent1  = parents[i%len(parents)]
    parent2 = parents[i%len(parents)]

    offspring = []
    for gene in range(len(parents[0])):

        if gene <= crossover_point:
            offspring.append(parent1[gene])
        else:
            offspring.append(parent2[gene])

    offsprings.append(offspring)
    i += 1
return offsprings
```

Figura 21: Cruzamento dos pais

O cruzamento implica que seja necessário definir um ponto de corte, a partir do qual o novo descendente vai receber os genes de cada um dos pais. Neste caso sendo três genes definiu-se um ponto de corte no segundo gene.

5.5 Mutação

```
for i in range(len(crossover)):
    #Generating a gene id to be mutated.
    gene_idx = np.random.randint(0, len(crossover[0]))
    #Saving current gene value
    current_value = crossover[i][gene_idx]

    while(True):
        new_value = round(np.random.random(),2)
        if current_value != new_value:
            break

        crossover[i][gene_idx] = new_value
```

Figura 22: Mutação dos genes

A mutação é aplicada em um gene aleatório de entre os 3 genes em cada novo descendente. O valor de mutação deve ser sempre diferente do valor atual do gene, conforme mostra a figura 22.

6 Resultados

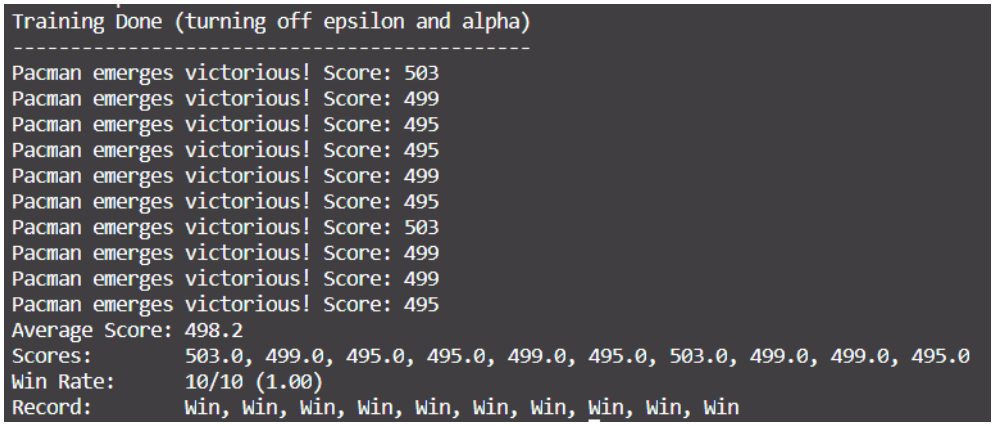

Figura 23: Execução com parâmetros já definidos

A figura 23, mostra o resultado da execução sem otimização, ou seja, apenas com a execução dos parâmetros definidos por defeito.

No entanto o objetivo é tentar encontrar estes mesmos parâmetros, tentando assim simular uma situação real de tentar otimizar o nosso modelo de aprendizagem por reforço.

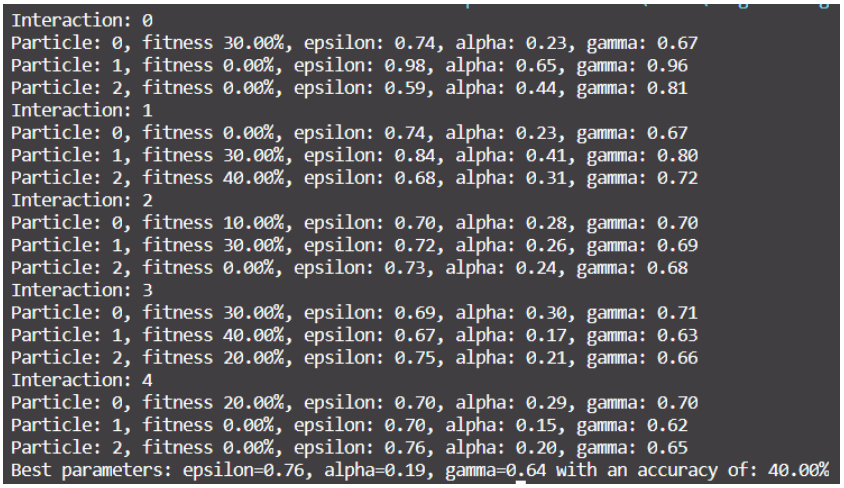

Figura 24: Otimização com exame de partículas

A otimização com exame de partículas nem sempre possibilitou que fosse possível obter uma solução ótima, conforme mostra a figura 24. A execução decorreu durante 5 iterações, com 3 partículas cada, no qual o resultado final foi uma accuracy de 40%.

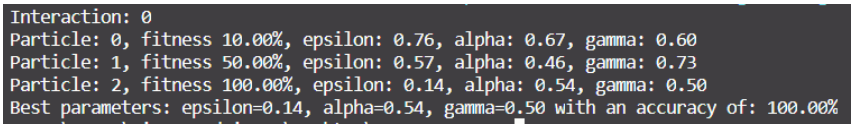

Figura 25: Segunda execução com exame de partículas

Apesar dos maus resultados obtidos na primeira execução com exame de partículas, a figura 25 mostra o resultado de uma segunda execução, em que foi possível obter uma accuracy de 100%.

A dificuldade em obter resultados satisfatórios com o exame de partículas, pode ser devido ao facto de este tipo de algoritmos ter tendência para puder cair num mínimo global, levando a que todo o exame não apresente grandes ganhos. Em conjunto com a dificuldade em definir alguns dos parâmetros que compõem este tipo de algoritmos, pode ser umas das razões para os resultados obtidos.

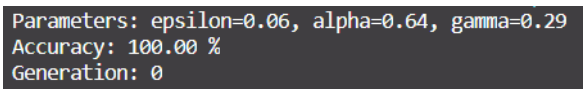

Figura 26: Execução com algoritmos genéticos

A figura 26, mostra o resultado da execução com algoritmos genéticos, no qual é possível verificar que numa primeira geração foi capaz de obter 100% de accuracy.

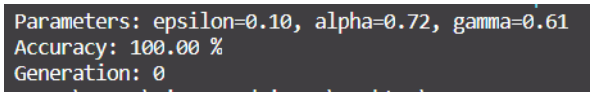

Figura 27: Segunda execução com algoritmos genéticos

O resultado de uma segunda execução pode ser observado na figura 27, obtendo os mesmos resultados em relação à primeira execução.

No geral os algoritmos genéticos apresentam-se como um tipo de algoritmo mais simples de parametrizar, quando comparado com o exame de partículas, no entanto o único cuidado que devemos ter com este tipo de algoritmos e o número de pais aptos para o cruzamento, assim como a condição de paragem do próprio algoritmo e a mutação dos novos descendentes da população.

7 Conclusão

A possibilidade de ter uma perceção sobre o funcionamento da aprendizagem por reforço e de como a mesma pode ser aplicada na tentativa de solucionar vários problemas.

O uso de enxame de partículas, assim como as vantagens e desvantagens neste tipo de algoritmos.

Foi possível comparar o exame de partículas juntamente com os algoritmos genéticos, de forma a ter uma perceção em relação ao tipo de problemas em que ambos podem ser aplicados.

Como trabalho futuro seria interessante criar um modelo capaz de jogar mapas mais complexos de pacman, e verificar a influencia da existência de mais ações/estados no desempenho final do modelo, assim como adicionar mais métodos de otimização de forma a fazer uma comparação mais profunda entre eles.