

Inteligência Artificial

Prof. Denis Fantinato

3º Quadrimestre de 2018

Inteligência Artificial

Implementação Neuro Evolução

Características de entrada das Redes Neurais Artificiais:

<http://playground.tensorflow.org>

Dados de Entrada (Sensores)

Usar todos os pixels do jogo do Super Mario como dado de entrada pode acabar dificultando o processamento da rede neural artificial (RNA).

Ideia:

- Dividir os pixels em blocos com resolução aceitável para a RNA: blocos com 16x16 pixels.

Além disso, ao invés de coletar os pixels da tela do jogo e fazer algum processamento de imagem, podemos tentar ler direto da memória RAM do console.

O arquivo `rominfo.py` faz a leitura dos dados na RAM e retorna algumas características do jogo.

`getSprites(ram)`: retorna os sprites (blocos, inimigos, itens) exibidos na tela.

`getTile(dx, dy, ram)`: retorna se tem um bloco que o mario possa pisar na posição dx, dy.

`getInputs(ram)/getState(ram,radius)`: retorna uma array de inimigos, obstáculos dentro de um raio em torno do agente.

Assim, uma tela



Seria lida como:

```
[ ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ]
[ ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ]
[ ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ]
[ ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ]
[ ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ]
[ ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ]
[ ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ]
[ ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ]
[ '$$' '$$' '$$' '$$' '$$' '$$' '$$' '$$' '$$' '$$' '$$' '$$' '$$' ]
[ ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ]
[ ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ]
[ ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ]
[ ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ]
```

Responda:

- Qual a vantagem de ler a memória RAM ao invés de usar diretamente os pixels da imagem?

O Agente

Características do agente

Quais movimentos estarão disponíveis:

```
moves = {'direita':128, 'corre':130, 'pula':131, 'spin':386}
```

Visão:

```
# raio de visão (quadriculado raio x raio em torno do Mario)
```

```
raio = 6
```

Cada bloco de imagem representa 16x16 pixels portanto tendo um x, y de referência do Mario devemos caminhar de 16 em 16.

Vetor de entrada expandido (para conseguir capturar da RAM alguns sprites maiores):

$(raio * 2 + 1) \times (raio * 2 + 1)$

Ambiente/Simulação

Usa coordenadas x, y de referência do Mario.

Joga uma partida usando uma sequência de ações

```
def emula(nn):

    dxtot = 0
    for stage in ['YoshiIsland1', 'YoshiIsland2']:
        env = retro.make(game='SuperMarioWorld-Snes', state=stage, players=1)
        env.reset()
        it = 0
        xn = 1
        dx = 1
        deltax = 0

        while it < 50 and xn != 0 and (not env.data.is_done()):
            dx = xn
            estado, xn, y = getState(getRam(env), raio)
            dx = xn - dx

            it = it+1 if dx==0 else 0
            deltax = xn if xn > deltax else deltax

            a = nn.getAction(list(map(float, estado.split(','))))
            reward, done = performAction(a, env)

            if mostrar:
                env.render( )
        env.close()
        dxtot += deltax
    return dxtot
```

Responda:

- O que faz deltax? E dxtot?
- O que a variável it está contando?
- Quais são as condições para se manter no while?

Função getState definido em rominfo.py:

Recupera o estado atual como uma string

```
def getState(ram, radius):
    state, x, y = getInputs(ram, radius=radius)
    return ','.join(map(str,state)), x, y
```

getInputs(ram): retorna uma array de inimigos, obstáculos dentro de um raio em torno do agente

```

# faz as ações até mudar de estado
def performAction(a, env):
    reward = 0
    if a == 64 or a == 128:
        for it in range(8):
            ob, rew, done, info = env.step(dec2bin(a))
            reward += rew
    elif a == 66 or a == 130:
        for it in range(4):
            ob, rew, done, info = env.step(dec2bin(a))
            reward += rew
    elif a == 131 or a == 67:
        for it in range(8):
            ob, rew, done, info = env.step(dec2bin(a))
            reward += rew
    elif a == 386 or 322:
        for it in range(4):
            ob, rew, done, info = env.step(dec2bin(a))
            reward += rew
    else:
        ob, rew, done, info = env.step(dec2bin(a))
        reward += rew
    return reward, done

```

reward é a recompensa recebida ao se avançar no jogo (pode ou não ser utilizada).
done é usado para verificar se o final do jogo foi alcançado.
ob, info contêm variáveis do jogo e seus endereços na memória.

A Rede Neural Artificial

```
class NeuralNet:
    def __init__(self, n_hidden, Whidden = None, Wout = None):

        input_size = 4*raio*raio + 4*raio + 1

        self.Whidden, self.Wout = Whidden, Wout
        if Whidden is None:
            self.Whidden = np.random.randn(input_size, n_hidden)
        if Wout is None:
            self.Wout = np.random.randn(n_hidden, len(moves))

        deltax = emula(self)
        self.fitness = deltax

    def getAction(self, x):
        x = np.array(x, dtype='float64')
        z = np.tanh(x @ self.Whidden)
        y = np.exp(z @ self.Wout)
        y = y / y.sum()
        idx = np.argmax(y)
        move_list = list(moves.items())

        return move_list[idx][1]
```

Responda:

- Qual a dimensão dos dados de entrada?
- Quantas camadas intermediárias existem?
- Quantos neurônios existem em cada camada intermediária? E na camada de saída?
- Qual a função de ativação dos neurônios na camada intermediária? E na camada de saída?
- Qual o fitness (função objetivo) sendo utilizada?
- O que a função `getAction` está retornando?

Algoritmo Genético (Adaptado)

```
def GA(n_iter, n_pop, n_hidden):
    P = start_or_load(n_hidden, n_pop)

    for it in range(n_iter):
        print(it, melhor(P).fitness, np.mean([p.fitness for p in P]))

        Psel = np.random.choice(P, 10)
        F = [muta(nn) for nn in Psel]
        P = P + F
        P = [melhor(P)] + seleciona(P, n_pop - 1)

        pickle.dump(P, open('NeuroGACurState.pkl', 'wb'))

    return melhor(P)

def start_or_load(n_hidden, n_pop):
    if os.path.exists('NeuroGACurState.pkl'):
        return pickle.load(open('NeuroGACurState.pkl', 'rb'))
    return [NeuralNet(n_hidden) for _ in range(n_pop)]

def melhor(P):
    fitness = [(-p.fitness, i) for i, p in enumerate(P)]
    idx = sorted(fitness)[0][1]
    return P[idx]
```

Resposta:

- O que é um indivíduo armazenado em P?
- Por que usa-se `-p.fitness` na função `melhor(P)`?
- Quantos indivíduos são selecionados em `Psel`?

```

def muta(nn):
    alpha = np.random.random()
    Whidden = nn.Whidden
    Wout = nn.Wout

    return NeuralNet(Whidden.shape[1],
                    Whidden + alpha*np.random.randn(*Whidden.shape),
                    Wout + alpha*np.random.randn(*Wout.shape))

```

*Whidden.shape e *Wout.shape expande os elementos da lista para a chamada da função.

Resposta:

- O que está sendo mutado na função muta()?
- alpha pode variar entre quais valores?

```

def seleciona(P, n):
    fitness = [(-p.fitness, i) for i, p in enumerate(P)]
    S = [P[idx] for f, idx in sorted(fitness)[1:n+1]]
    return S

```

Resposta:

- O que acontece na linha com $P = P + F$?
- Como os indivíduos estão sendo selecionados?

Chamada da função principal:

```

def main():
    global mostrar
    global env

    mostrar = False

    melhor = GA(50, 100, 20)
    print(melhor.fitness)

    mostrar = True
    emula(melhor)

if __name__ == "__main__":
    main()

```

Resultado após 200 gerações (n_pop = 100, hidden = 20):

- Yoshilsland 1:
<https://www.youtube.com/watch?v=dQ818AfU9c0>
- Yoshilsland 2:
<https://www.youtube.com/watch?v=B4X91h2hfYA>

Estratégia Evolutiva

```
def ES(n_iter, n_pop, n_hidden):
    P = start_or_load(n_hidden, n_pop)

    for it in range(n_iter):
        print(it, P.fitness)
        F = muta(P, n_pop)
        P = P if P.fitness > F.fitness else F

        pickle.dump(P, open('NeuroEvoCurState.pkl', 'wb'))

    return P

def start_or_load(n_hidden, n_pop):
    if os.path.exists('NeuroEvoCurState.pkl'):
        return pickle.load(open('NeuroEvoCurState.pkl', 'rb'))
    return NeuralNet(n_hidden)
```

Resposta:

- Quantos indivíduos existem?

```
def muta(nn, n_pop):
    alpha = 0.05
    sigma = 0.1

    Whidden = nn.Whidden
    Wout = nn.Wout

    fit = np.zeros(n_pop)
    Ghid = np.random.randn(n_pop, *Whidden.shape)
    Gout = np.random.randn(n_pop, *Wout.shape)

    for i in range(n_pop):
        nni = NeuralNet(Whidden.shape[1],
                        Whidden + sigma*Ghid[i,:,:],
                        Wout + sigma*Gout[i,:,:])
        fit[i] = nni.fitness
    A = (fit - fit.mean())/fit.std()

    Wmuth = Whidden + (alpha/(n_pop*sigma)) * np.dot(Ghid.T, A).T
    Wmut = Wout + (alpha/(n_pop*sigma)) * np.dot(Gout.T, A).T

    return NeuralNet(Whidden.shape[1], Wmuth, Wmut)
```

Resposta:

- Ghid e Gout podem assumir quais valores?

- Por que é necessário usar `np.dot`? Qual a dimensão de A?
- O que A faz?
- Quantos indivíduos há agora?

Chamada da função principal:

```
def main():
    global mostrar
    global env

    mostrar = False

    melhor = ES(50, 50, 75)
    print(melhor.fitness)

    mostrar = True
    emula(melhor)

if __name__ == "__main__":
    main()
```