Universidade Federal de Santa Catarina - UFSC

Centro Tecnológico - CTC

Departamento de Automação e Sistemas - DAS

DAS5341 - Inteligência Artificial Aplicada a Controle e Automação

CarRacing-v0

Florianópolis - SC

3 de dezembro de 2021

Sumário

1 *Setup* **2**

1.1 *Pycharm* . . . 2

1.2 VS Code . . . 2

2 Notas sobre o código fonte **3**

2.1 Controle manual . . . 3

2.2 Simulação . . . 3

2.3 Salvar trajetórias . . . 4

2.4 *Miscellaneous* . . . 4

3 Escolha e treinamento da Rede Neural em *PyTorch* **4**

3.1 Arquitetura da Rede Neural . . . 4

3.2 Treinamento da Rede Neural . . . 5

4 Controle do veículo com a Rede Neural **5**

5 Referências **6**

1 *Setup*

1.1 *Pycharm*

O processo a seguir é caso você vá utilizar o *Pycharm*.

1. Instale o *Pycharm* com *Python 3.6*;
2. Crie um novo projeto em *File* > *New Project*;
3. Apague o que estiver dentro de *main.py*;
4. Entre em <https://github.com/openai/gym/blob/master/gym/envs/box2d/car_racing.py> e copie o código fonte do simulador para o arquivo *main.py* que você criou;
5. Instale as seguintes bibliotecas:
   (a) *gym*
   (b) *pickle-mixin*
   (c) *torch*
   (d) *Box2D*
   (e) *numpy*
   (f) *copy*
   - No *Pycharm*, você pode instalar as bibliotecas ao clicar em *File* > *Settings*
   - Na janela que abrirá, selecione na barra da esquerda a opção *Project* > *Project Interpreter*
   - Nesta nova janela, clique no + que está logo acima de *packages*
   - Agora, procure as bibliotecas supracitadas e instale-as ao clicas em *Install package*
6. Por fim, no menu *Run*, clique em *run main*. Uma janela com o simulador deve ser abrir. Você pode controlar o veículo com as teclas direcionais do teclado.

1.2 VS Code

Caso você esteja mais habituado com o *Visual Studio Code*, poderá também utilizá-lo para rodar o simulador.

1. Crie um novo arquivo. Pode chamá-lo como quiser, salvando-o como .py.
2. Entre em <https://github.com/openai/gym/blob/master/gym/envs/box2d/car_racing.py> e copie o código fonte do simulador para o arquivo que você criou;
3. Instale as mesmas bibliotecas citadas anteriormente.
   - No *VS Code* você pode instalá-las com o seguinte comando no terminal: **pip install biblioteca**. Ex: pip install numpy;
   - Você sempre pode checar quais bibliotecas já estão instaladas ao digitar *pip freeze* no terminal.
4. Execute arquivo com o código do simulador. (Dica: **Ctrl** + **alt** + **n** serve para dar *run* e **Ctrl** + **alt** + **m** para a execução do arquivo). Uma janela deverá se abrir e você pode controlar o veículo com as setas do teclado.

2 Notas sobre o código fonte

2.1 Controle manual

O controle manual do veículo está entre as linhas 595-616. Esta implementação gera um *numpy array* com 3 posições chamado a, que serão as entradas no laço principal do simulador. A primeira posição de a contém informações da direção do carrinho, a segunda é relativa à aceleração e a terceira ao freio.

```
595     def key_press(k, mod):
596         global restart
597         if k == 0xFF0D:
598             restart = True
599         if k == key.LEFT:
600             a[0] = -1.0
601         if k == key.RIGHT:
602             a[0] = +1.0
603         if k == key.UP:
604             a[1] = +1.0
605         if k == key.DOWN:
606             a[2] = +0.8  # set 1.0 for wheels to block to zero rotation
607 
608     def key_release(k, mod):
609         if k == key.LEFT and a[0] == -1.0:
610             a[0] = 0
611         if k == key.RIGHT and a[0] == +1.0:
612             a[0] = 0
613         if k == key.UP:
614             a[1] = 0
615         if k == key.DOWN:
616             a[2] = 0
```

2.2 Simulação

O *loop* principal do jogo está entre as linhas 628-643.

```
628     while isopen:
629         env.reset()
630         total_reward = 0.0
631         steps = 0
632         restart = False
633         while True:
634             s, r, done, info = env.step(a)
635             total_reward += r
636             if steps % 200 == 0 or done:
637                 print("\naction " + str([f"{x:+0.2f}" for x in a]))
638                 print(f"step {steps} total_reward {total_reward:+0.2f}")
639             steps += 1
640             isopen = env.render()
641             if done or restart or isopen == False:
642                 break
643     env.close()
```

Na linha 629 o jogo é "resetado"e o estado inicial s_0 é definido. (Dica: você pode armazenar o estado inicial ao atribui-lo a uma variável. Ex: s_prev = env.reset()).

Na linha 634, dado o estado atual s_t o jogador toma uma ação a_t. Essa ação é dada como entrada em env.step(a), gerando um novo estado s e outras informações (como a recompensa r).

2.3 Salvar trajetórias

Dica: antes de salvar as trajetórias, como cada volta gera uma pista aleatória, é possível fixar a pista. Nas linhas 139-141, é onde ocorre a geração da *seed* da pista. (Dica: dê um *print* em *seed* e rode o código. Após obter esse valor, atribua-o a *seed* logo após a linha 139)

```
139     def seed(self, seed=None):
140         self.np_random, seed = seeding.np_random(seed)
141         return [seed]
```

A fim de armazenar as trajetórias que irão servir para treinar o modelo, é possível criar uma lista de pares de estado/ação. Editando o código fonte, você irá adicionar a cada iteração o estado e a ação tomada naquele estado. Ex: no início, linhas 629-623, adiciona-se o estado inicial e a ação na lista: listaEstadoAcao.append((s_prev,a)).

É possível que você encontre problemas ao salvar a ação, *i.e.*, todas as ações salvas são iguais. Neste caso, faça um *deep copy* de a e, apenas então, adicione-a à lista (<https://docs.python.org/3/library/copy.html>).

Por fim, ao terminar o episódio, devemos salvar essa lista como um arquivo. Para salvá-la, podemos utilizar o *pickle*. (<https://stackoverflow.com/questions/52444921/save-numpy-array-using-pickle>).

2.4 *Miscellaneous*

- Os controles manuais (*direction, throttle, brake*) estão inicialmente ajustados para o máximo. Valores menores podem auxiliar no controle do carro para gerar as trajetórias manualmente;
- O *loop* original é infinito, mas pode ser editado para salvar uma trajetória por arquivo, por exemplo;
- Por fim, após colher os dados de treinamento, é necessário definir uma rede neural será o agente.

3 Escolha e treinamento da Rede Neural em *PyTorch*

Agora, devemos escolher uma arquitetura para a rede neural e então treiná-la. Utilizaremos a biblioteca *PyTorch*.

3.1 Arquitetura da Rede Neural

Como iremos trabalhar com imagens, devemos utilizar Redes Neurais Convolucionais. Para a arquitetura dessa rede, podemos nos basear na mesma utilizada por Mnih *et al* (2015). Como estamos trabalhando com uma imagem RGB, o formato da imagem é $3 \times 96 \times 96$.

A arquitetura exata é a seguinte: na **primeira** camada oculta, camada essa que é convolucional, temos como entrada a imagem com 3 canais. É aplicado um filtro/*kernel* de 8×8 com um *stride* (passo que passamos o filtro) de 4. Temos como saída 32 canais, e por fim aplicamos a função *ReLU*. Para a

segunda camada oculta, temos como entrada a saída da camada anterior, mas agora com um filtro de 4×4 e *stride* 2, novamente aplicando a função *ReLU* na saída, que agora possui 64 canais. A última camada de convolução, a **terceira** camada oculta, recebe a saída anterior como entrada e utiliza um filtro de 3×3 com *stride* 1. A saída será de tamanho 64 e, novamente, aplica-se a *ReLU*. A **última** camada oculta é uma camada totalmente conectada e possui saída igual a 512. Por fim, a camada de **saída/final** é constituída de outra camada totalmente conectada, que possui como saída 3 neurônios, a direção, aceleração e freio, no nosso caso. Aplicamos a função sigmoide na saída da rede.

3.2 Treinamento da Rede Neural

Agora, deve-se treinar a rede. Passando as imagens (estados) como entradas da rede, teremos saídas (ações) previstas. Faz-se a comparação entre as ações previstas e as reais utilizando uma *loss function*, que, no caso, pode ser a *Binary Cross Entropy*. Note que, como a nossa *loss function* espera valores entre 0 e 1, devemos aplicar uma função que coloque o *array* de ações esperadas, mais especificamente o termo de direção (direita/esquerda), na escala de 0 a 1. Só então aplicamos a *loss function* em cada ação prevista e esperada (direção, aceleração, freio) e somamos para ter o erro total da iteração.

Um bom exemplo de treinamento de uma rede neural pode ser visto em <https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html>.

Por fim, após treinar a rede, salve-a em um arquivo. (<https://stackoverflow.com/questions/42703500/best-way-to-save-a-trained-model-in-pytorch>)

4 Controle do veículo com a Rede Neural

Agora, para que se possa utilizar a rede treinada para controlar o veículo, deve-se criar um novo arquivo baseado no código fonte oficial do *CarRacingV0*.

Altere o código de forma a carregar a rede neural treinada. Utilize o estado inicial s_0 como entrada da rede neural, gerando uma saída a. Utilize esse a como entrada em env.step(a). Isso irá gerar um novo estado s, e esse estado será a entrada da rede neural, gerando uma nova ação. Assim, repete-se esse processo, permitindo que o carro dê uma volta completa.

5 Referências

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski et al., "**Human-level control through deep reinforcement learning**" nature, vol. 518, no. 7540, pp. 529–533, 2015.