Relatório do Laboratório 1: File Transfer Protocol (FTP)

Isabelle Ferreira de Oliveira
CES-35 - Engenharia da Computação 2020
Instituto Tecnológico de Aeronáutica (ITA)
São José dos Campos, Brasil
isabelle.ferreira3000@gmail.com

***Resumo*—O trabalho tem como objetivo a implementação de um servidor de transferência de arquivos básico inspirado na interface do File Transfer Protocol (FTP), com os requisitos conforme apresentados no roteiro do laboratório.**

***Index Terms*—FTP, Redes de computadores, Cliente-Servidor**

I. Implementação

A linguagem de programação utilizada foi Python 3, com o auxílio das bibliotecas: *socket* para comunicação entre os processos, *thread* para a criação das threads para cada cliente conectado ao servidor, e *shutil* e *os* para os comandos de manipulação de diretórios e arquivos.

A. Funções auxiliares

```
# Server code
def receive_message(conn):
    message_received = ""
    while True:
        data_received = conn.recv(16)
        message_received = message_received +
            data_received.decode("utf-8")
        if message_received.endswith("\r\n"):
            break
    message_received =
        message_received.split("\r\n")[0]
    return message_received
```

```
# Server code
def check_authentication(conn):
    conn.sendall(bytes("user:\r\n", 'utf-8'))
    username = receive_message(conn)

    conn.sendall(bytes("password:\r\n",
        'utf-8'))
    password = receive_message(conn)

    authenticate = False
    if username in credentials:
        if credentials[username] == password:
            authenticate = True

    return authenticate, username
```

B. Navegação e listagem de diretórios

```
# Server code
if comm == "cd":
    curr_path = curr_session.current_directory

    try:
        os.chdir(curr_path + "/" + dirname)
        curr_path = os.getcwd()
        curr_session.current_directory =
            curr_path
        conn.sendall(bytes("ok\r\n", 'utf-8'))

    except FileNotFoundError:
        conn.sendall(bytes("Error: directory
            does not exists in server \r\n",
            'utf-8'))
```

```
# Server code
elif comm == "ls":
    if len(args) != 0:
        dirname = args[0]

        try:
            path = curr_session.current_directory
                + "/" + dirname
            fileslist = os.listdir(path)
            conn.sendall(bytes(str(fileslist) +
                "\r\n", 'utf-8'))

        except FileNotFoundError:
            conn.sendall(bytes("Error: directory
                does not exists in server\r\n",
                'utf-8'))
    else:
        path = curr_session.current_directory
        fileslist = os.listdir(path)
        conn.sendall(bytes(str(fileslist) +
            "\r\n", 'utf-8'))
```

```
# Server code
elif comm == "pwd":
    path = curr_session.current_directory
    conn.sendall(bytes(path + "\r\n", 'utf-8'))
```

C. Manipulação de diretórios

```
# Server code
elif comm == "mkdir":
```

```
  try:
    path = curr_session.current_directory
     os.mkdir(path + "/" + dirname)
     conn.sendall(bytes("ok\r\n", 'utf-8'))

  except OSError:
     conn.sendall(bytes("Error: directory
         already exists in server\r\n",
         'utf-8'))
```

```
# Server code
elif comm == "rmdir":
  try:
    path = curr_session.current_directory
     shutil.rmtree(path + "/" + dirname)
     conn.sendall(bytes("ok\r\n", 'utf-8'))

  except OSError:
     conn.sendall(bytes("Error: directory
         does not exists in server\r\n",
         'utf-8'))
```

D. Manipulação de arquivos

```
# Server code
elif comm == "get":
  if check_if_file_exists(conn, filename):
    # if file not exists in local
    # or if can overwrite
     feedback = receive_message(conn)

     if feedback == "can get":
       path = curr_session.current_directory
       file = open(path + "/" + filename,
           "rb")
       aux = file.read(1024)
       while aux:
          conn.send(aux)
          aux = file.read(1024)
       conn.sendall(bytes("\r\n", 'utf-8'))
```

```
# Server code
elif comm == "put":
  # if file exists in local
  feedback = receive_message(conn)

  if feedback == "files exists in local":
     filename = filename.split("/")[-1]

     # if file exists in server
     check_if_file_already_exists(conn,
         filename)

     can_continue = receive_message(conn)
     if can_continue == "Y":
       path = curr_session.current_directory
       f = open(path + "/" + filename, 'wb')
       aux = conn.recv(1024)
       while aux:
          f.write(aux)
          aux = conn.recv(1024)
          if aux.endswith(bytes("\r\n",
              'utf-8')):
             break
     else:
        conn.sendall(bytes("ok\r\n", 'utf-8'))
```

```
# Server code
elif comm == "delete":
  try:
    path = curr_session.current_directory
     os.remove(path + "/" + filename)
     conn.sendall(bytes("ok\r\n", 'utf-8'))

  except OSError:
     conn.sendall(bytes("Error: file does not
         exists in server\r\n", 'utf-8'))
```

E. Gerenciamento de conexões

```
# Server code
if command == "close" or command == "quit":
  sessions.pop(conn)
  conn.close()
  break
```

II. Diagramas de Sequência

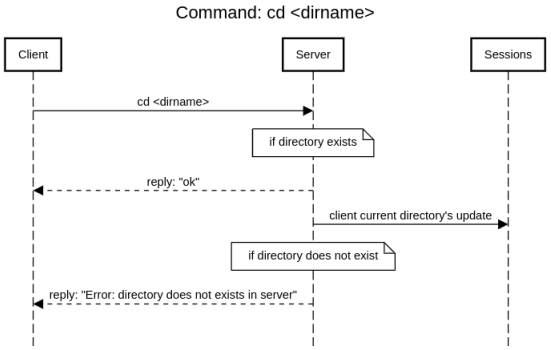

Figura 1. Sumário do modelo implementado em Keras.

.

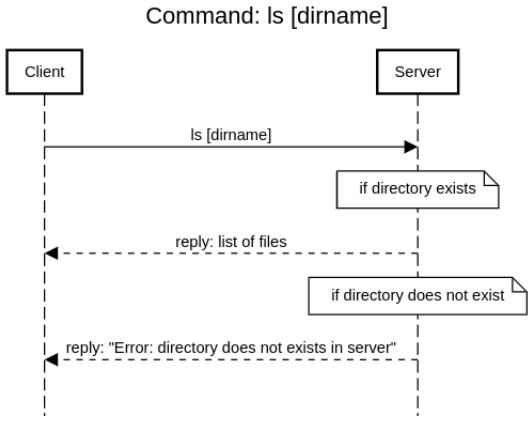

Figura 2. Sumário do modelo implementado em Keras.

.

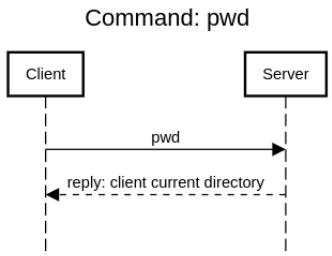

Figura 3. Sumário do modelo implementado em Keras.

.

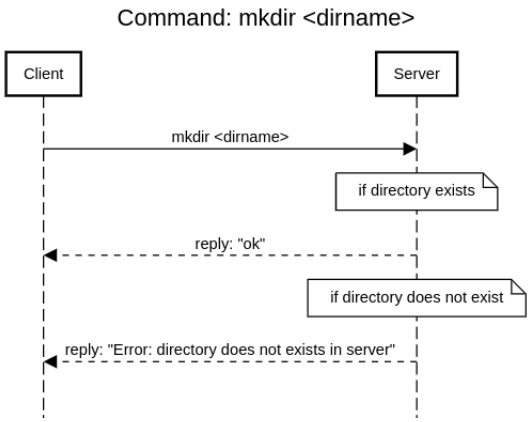

Figura 4. Sumário do modelo implementado em Keras.

.

III. Funcionamento

A. Implementação da Definição da Rede Neural

Essa primeira etapa se tratou da implementação do método build_model() da classe DQNAgent de dqn_agent.py, script fornecido no código base do laboratório. Nesse método, era preciso construir uma rede em Keras de acordo com as especificações apresentadas na Tabela 3 do roteiro do laboratório [1].

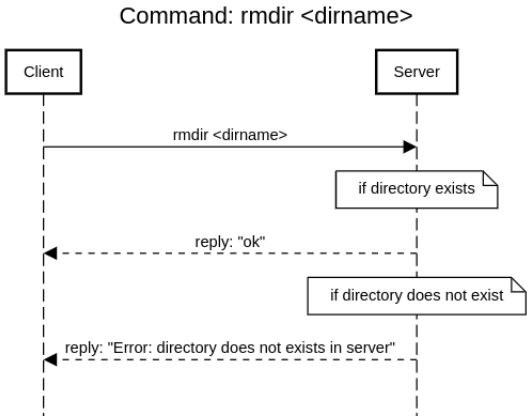

Figura 5. Sumário do modelo implementado em Keras.

.

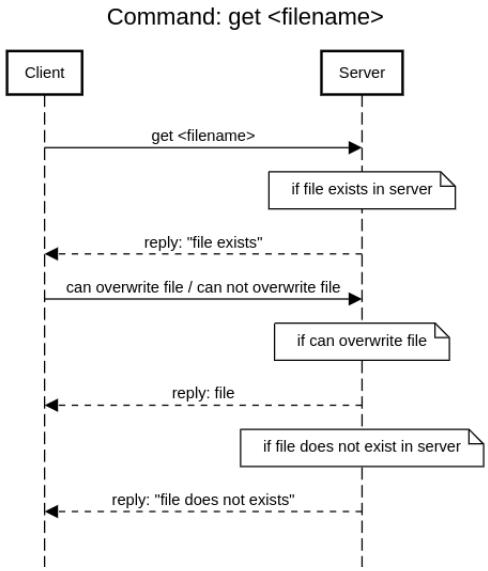

Figura 6. Sumário do modelo implementado em Keras.

.

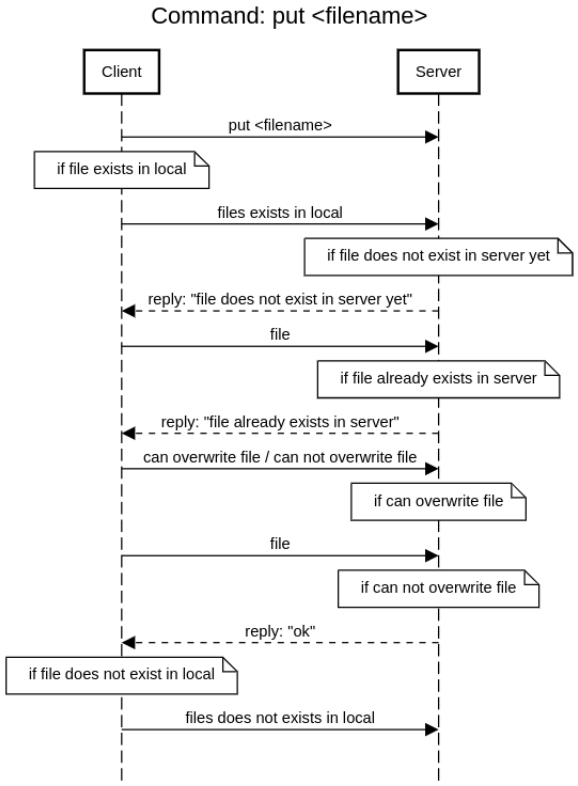

Figura 7. Sumário do modelo implementado em Keras.

.

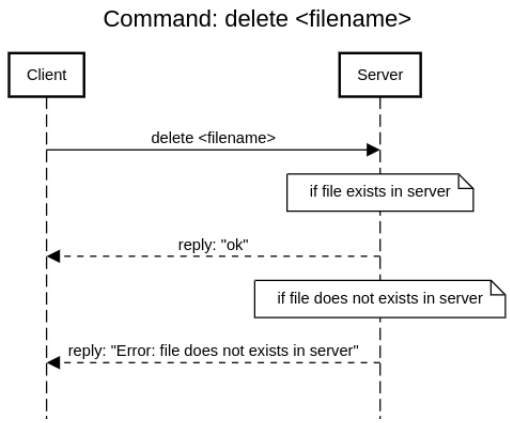

Figura 8. Sumário do modelo implementado em Keras.

.

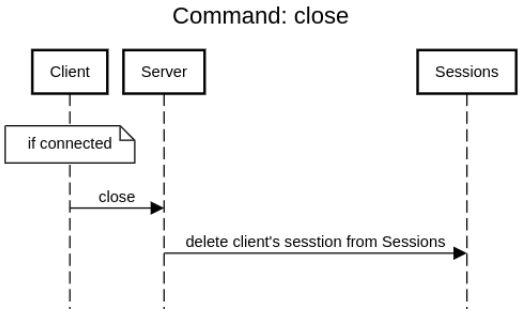

Figura 9. Sumário do modelo implementado em Keras.

.

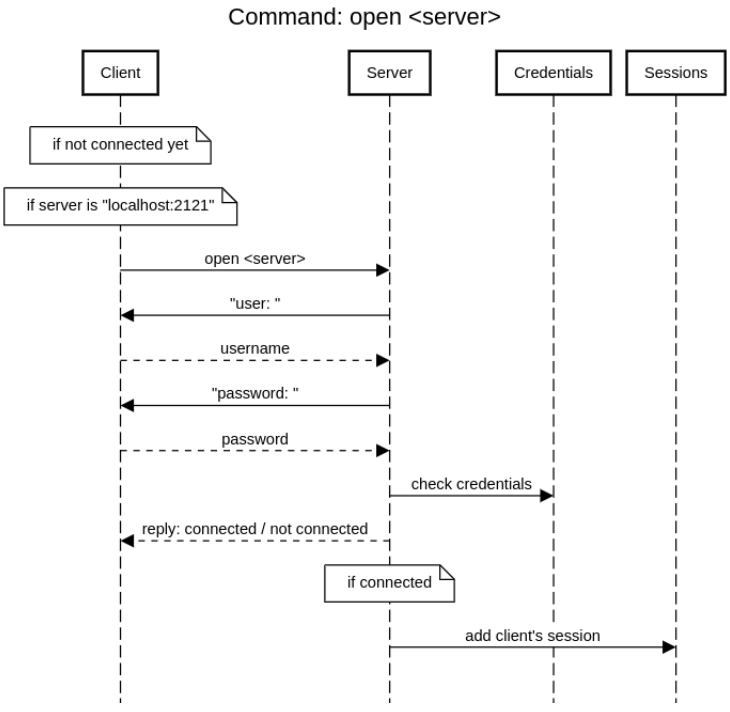

Figura 10. Sumário do modelo implementado em Keras.

.

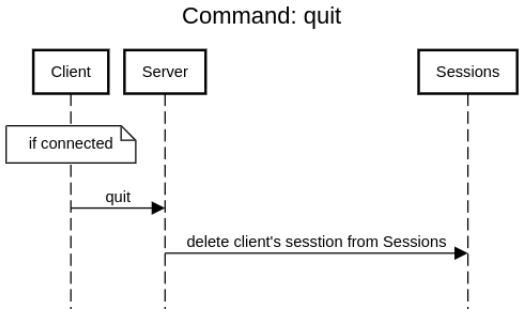

Figura 11. Sumário do modelo implementado em Keras.

.

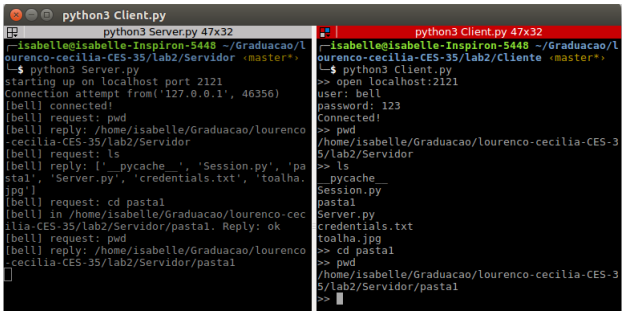

Figura 12. Sumário do modelo implementado em Keras.

.

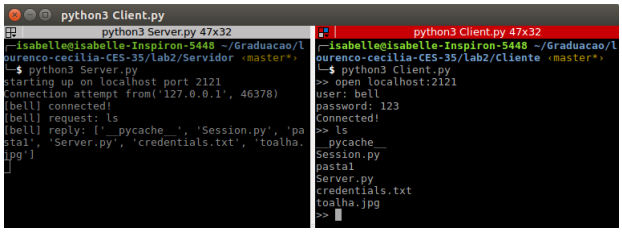

Figura 13. Sumário do modelo implementado em Keras.

.

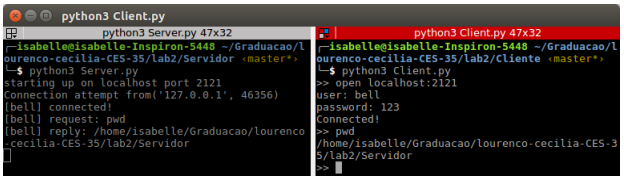

Figura 14. Sumário do modelo implementado em Keras.

.

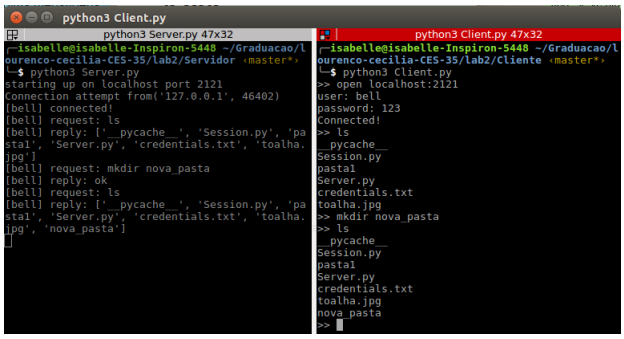

Figura 15. Sumário do modelo implementado em Keras.

.

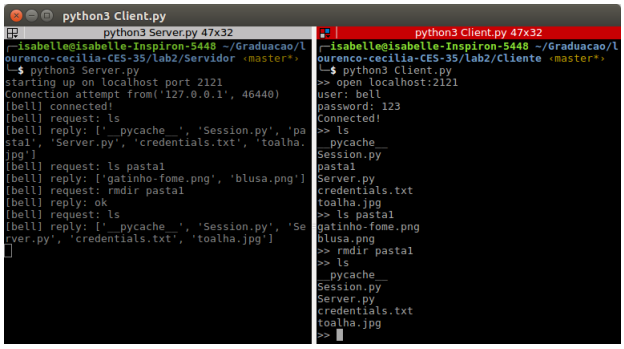

Figura 16. Sumário do modelo implementado em Keras.

.

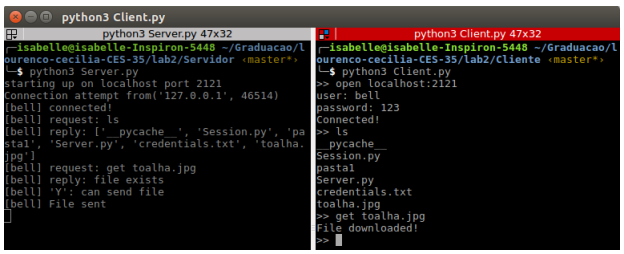

Figura 17. Sumário do modelo implementado em Keras.

.

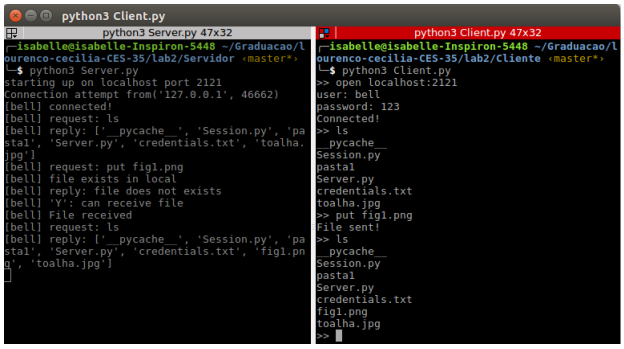

Figura 18. Sumário do modelo implementado em Keras.

.

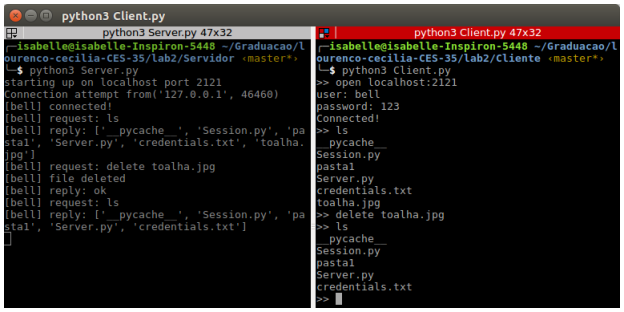

Figura 19. Sumário do modelo implementado em Keras.

.

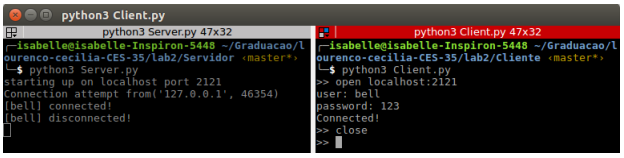

Figura 20. Sumário do modelo implementado em Keras.

.

Essa implementação foi feita de forma bastante análoga à maneira do laboratório 8 [2], ou seja, seguindo o apresentado no pseudo-código em Python a seguir.

```
# Adds the first layer
model.add(layers.Dense(num_neurons,
  activation=activations.some_function,
  input_dim=state_size))

# Adds another layer (not first)
model.add(layers.Dense(num_neurons,
  activation=activations.some_function))
```

Vale ressaltar que, para atender os critérios requisitados, some_function do pseudo-código acima se tratou de *relu* para as duas primeiras camadas, e de *linear* para terceira camada. Além disso, num_neurons foram 24, 24 e action_size para as

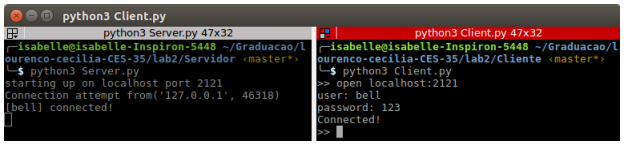

Figura 21. Sumário do modelo implementado em Keras.

.

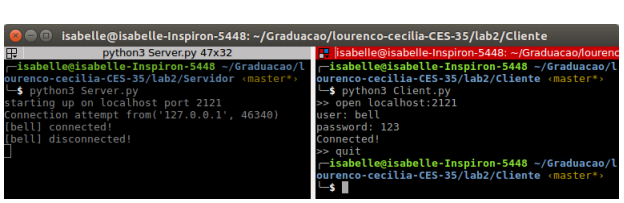

Figura 22. Sumário do modelo implementado em Keras.

.

primeira, segunda e terceira camada, respectivamente.

Fora isso, bastou-se descomentar as linhas de criação de uma pilha linear de camadas, as linhas compilação do modelo e impressão do summary do modelo, apresentado futuramente na seção IV (Resultados e Conclusões), e a linha de retorno da função.

B. Escolha de Ação usando Rede Neural

Já essa etapa se tratou da implementação do método act() também da classe DQNAgent de dqn_agent.py. Nesse método, era escolhido e retornado uma ação de acordo com a política ϵ-greedy.

Essa implementação foi feita de forma bastante análoga à maneira do laboratório 12 [3]. Assim, gerou-se um número aleatório entre 0 e 1 e, caso esse valor aleatório seja menor que epsilon, então uma ação aleatória é escolhida; caso contrário, é escolhida a ação gulosa, através do retorno do índice do máximo elemento do array *model.predict(state)[0]*.

C. Reward Engineering

Nesse momento, foi implementado o método reward_engineering_mountain_car() de utils.py, script também fornecido no código base do laboratório. Nesse método, eram calculadas e retornadas as recompensas intermediárias "artificias", chamadas reward engineering, a fim de tornar o treino mais rápido no ambiente do Mountain Car.

Essa implementação foi feita conforme as equações apresentadas na seção 4.3 do roteiro do laboratório [1], ou seja, assim como apresentado no pseudo-código em Python a seguir.

```
reward = reward + (position - start) *
    (position - start) + velocity * velocity

aux = 0
if next_position >= 0.5:
   aux = 1

reward += 50 * aux
```

Os valores de position, start, velocity e next_position também eram fornecidos no roteiro [1], e bastava substituí-los no pseudo-código acima.

D. Treinamento usando DQN

Bastava treinar o modelo implementado, executando o script train_dqn.py, também do código base, e observar os resultados e os gráficos obtidos.

E. Avaliação da Política

Bastava aplicar o modelo implementado no ambiente do Mountain Car, executando o script evaluate_dqn.py, também do código base, e observar a simulação, os resultados e os gráficos obtidos.

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 24)                72
_________________________________________________________________
dense_2 (Dense)              (None, 24)                600
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 75
=================================================================
Total params: 747
Trainable params: 747
Non-trainable params: 0
```

Figura 23. Sumário do modelo implementado em Keras.

IV. Resultados e Conclusões

O summary do modelo implementado em make_model() foi apresentado na Figura 23, e condiz com os requisitos pedidos na Tabela 3 do roteiro do laboratório [1].

Já a Figura 24 representa as recompensas acumulativas advindas do treinamento do modelo em 300 episódios. Esse resultado dependem diretamente da correta implementação e funcionamento dos métodos make_model() e act().

Pode-se dizer que esse gráfico condiz com o esperado, uma vez que é possível notar inicialmente recompensas pequenas para os primeiros episódios e, mais ou menos a partir do episódio 80, tornou-se frequente recompensas com valores elevados, chegando a valores próximos de 40, indicando um aprendizado significantemente correto.

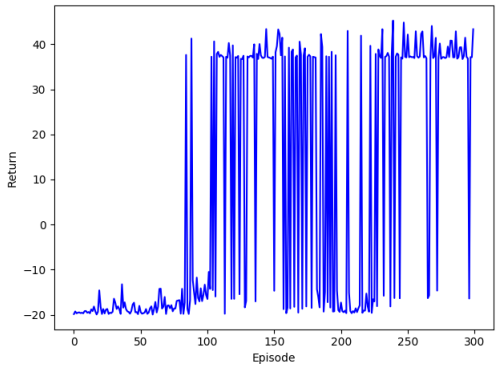

Figura 24. Recompensa acumulativa com o passar dos episódios, no treinamento do modelo para 300 episódios.

Já a aplicação do modelo implementado no ambiente do Mountain Car gerou as Figuras de 25 a 27.

A partir da Figura 25, pode-se concluir que a implementação e treino chegaram em resultados satisfatórios, uma vez que grande parte das recompensas acumuladas foi alta, próximas de 40, chegando no final de 30 episódios a uma média de 27.8, conforme apresentado na Figura 26.

Por fim, acerca da Figura 27, pode-se observar que:

- Para velocidades para direita, quase unanimamente a decisão do carro é continuar para direita. Exclui-se disso as situações de posição muito à esquerda e velocidades altas, na qual é decidido fazer nada, e de velocidades para direita muito baixas, na qual pouquíssimas vezes o

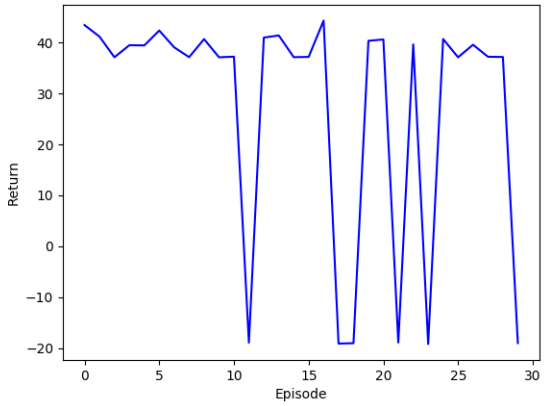

Figura 25. Representação em cores da tabela de action-value calculada, para algoritmo de Sarsa.

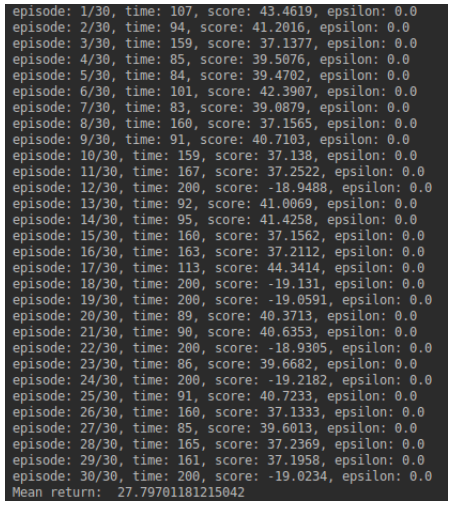

Figura 26. Recompensa acumulada em função das iterações, para algoritmo de Sarsa.

carro decide ir para esquerda, talvez já se enquadrando nas intenções descritas no próximo item.

- Para velocidades para esquerda, as decisões do carro diferem bastante da posição na qual ele se encontra. Para posições mais a esquerda, o carro decide continuar indo para esquerda, talvez para pegar impulso da subida e, quando por fim chegar em posições mais a esquerda (consequentemente mais altas) possíveis, decidir ir com velocidade para direita. Já para posições relativamente próximas da posição objetivo, aparecem também decisões de não fazer nada, indicando que o carro irá mais para esquerda e cairá na situação anteriormente descrita, na qual ele decidirá continuar indo para esquerda e pegará o impulso da elevação.

Como as decisões aprendidas e tomadas pelo carro fizeram sentido e puderam ser interpretadas satisfatoriamente, pode-se dizer que a proposta do laboratório foi corretamente implementada e se mostrou satisfatória em resolver o problema proposto.

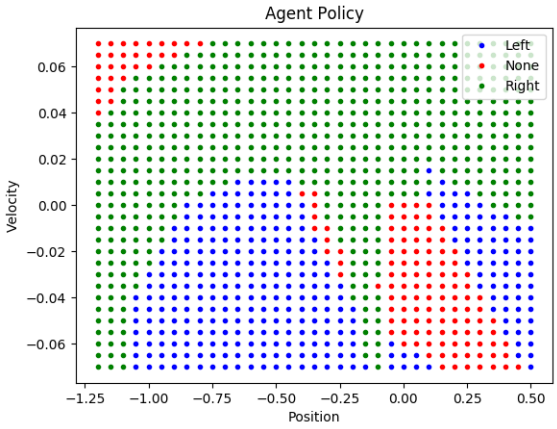

Figura 27. Representação em cores da tabela de greedy-policy calculada, para algoritmo de Sarsa.

Referências

[1] M. Maximo, “Roteiro: Laboratório 12 - Deep Q-Learning”. Instituto Tecnológico de Aeronáutica, Departamento de Computação. CT-213, 2019.

[2] M. Maximo, “Roteiro: Laboratório 8 - Imitation Learning com Keras”. Instituto Tecnológico de Aeronáutica, Departamento de Computação. CT-213, 2019.

[3] M. Maximo, “Roteiro: Laboratório 12 - Aprendizado por Reforço Livre de Modelo”. Instituto Tecnológico de Aeronáutica, Departamento de Computação. CT-213, 2019.