

Universidade Federal do Rio de Janeiro
Instituto Alberto Luiz Coimbra de
Pós-Graduação e Pesquisa de Engenharia



Programa de Engenharia de Sistemas e
Computação

CPS769 - Introdução à Inteligência Artificial e Aprendizagem Generativa

Prof. Dr. Edmundo de Souza e Silva (PESC/COPPE/UFRJ)

Profa. Dra. Rosa M. Leão (PESC/COPPE/UFRJ)

Participação Especial: Gaspare Bruno (Diretor Inovação, ANLIX)

Lista de Exercícios 1a

Luiz Henrique Souza Caldas

email: lhscaldas@cos.ufrj.br

15 de julho de 2024

Questão 1

Esse exemplo simples é para auxiliar a discussão do artigo “Serial Order A Parallel Distributed Processing Approach” que todos já devem ter lido. O objetivo é prever um padrão de figura, por exemplo um quadrado, usando uma Rede Neural Recorrente (RNN). Fornecemos o código em Python de um exemplo de geração do padrão 2-D de quadrados e treinamento de uma RNN para prever a sequência cíclica $[0, 25, 0, 25]$, $[0, 75, 0, 25]$, $[0, 75, 0, 75]$, $[0, 25, 0, 75]$, $[0, 25, 0, 25]$.

1. Entenda o código e explique qual a RNN que ele modela (faça o desenho). Explique a parte do código que define a RNN.

Resposta:

O código pode ser explicado dividindo-o em 6 partes:

- (a) Definição do caminho quadrado: O código define um conjunto de coordenadas que formam um caminho quadrado na variável *square_path*.

```
1 square_path = np.array([
2     [0.25, 0.25],
3     [0.75, 0.25],
4     [0.75, 0.75],
5     [0.25, 0.75],
6     [0.25, 0.25]
7 ])
```

- (b) Geração dos dados de treinamento: O caminho quadrado é repetido várias vezes para formar os dados de treinamento.

```
1 num_repeats = 4
2 data = np.tile(square_path, (num_repeats, 1))
3 x_train = data[:-1].reshape(-1, 1, 2)
4 y_train = data[1:].reshape(-1, 2)
```

- (c) Definição e compilação do modelo RNN: O modelo RNN é definido usando uma camada LSTM (*long short-term memory*) seguida de uma camada densa e depois é compilado configurando o algoritmo ADAM como otimizador e o Erro Médio Quadrático como função de perda.

```
1 model = models.Sequential([
2     layers.LSTM(50, activation='relu', input_shape=(num_repeats, 2)),
3     layers.Dense(2)
4 ])
5 model.compile(optimizer='adam', loss='mse')
```

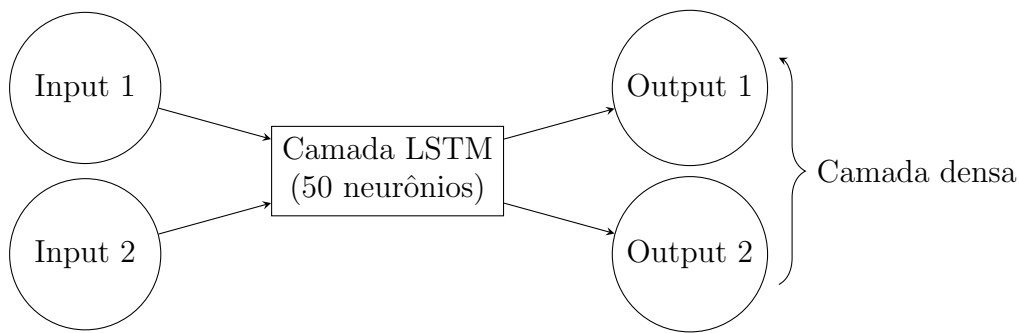


Figura 1: Diagrama de uma Rede Neural Recorrente (RNN) com dois neurônios de entrada e dois neurônios de saída (camada densa) e uma camada LSTM modelada no código fornecido.

- (d) Treinamento do modelo: O modelo é treinado com os dados gerados, utilizando inicialmente 300 épocas.

```
1 model.fit(x_train, y_train, epochs=300, verbose=0)
```

- (e) Geração das previsões: As previsões são geradas.

```
1 predictions = model.predict(x_train[:5])
```

- (f) Plotagem dos resultados: As previsões são plotadas e comparadas com o caminho original.

```
1 plt.plot(data[:, 0], data[:, 1], label='Original Path', linestyle='
    dashed', color='gray')
2 plt.plot(predictions[:, 0], predictions[:, 1], label='Predicted Path',
    color='blue')
3 plt.scatter(square_path[:, 0], square_path[:, 1], color='red')
4 plt.legend()
5 plt.show()
```

Este código foi levemente alterado por mim, criando-se a classe *PathPredictor* e fazendo as alterações necessárias para responder os próximos itens.

2. Treine a rede. Aprenda como fazer, e explique.

Resposta:

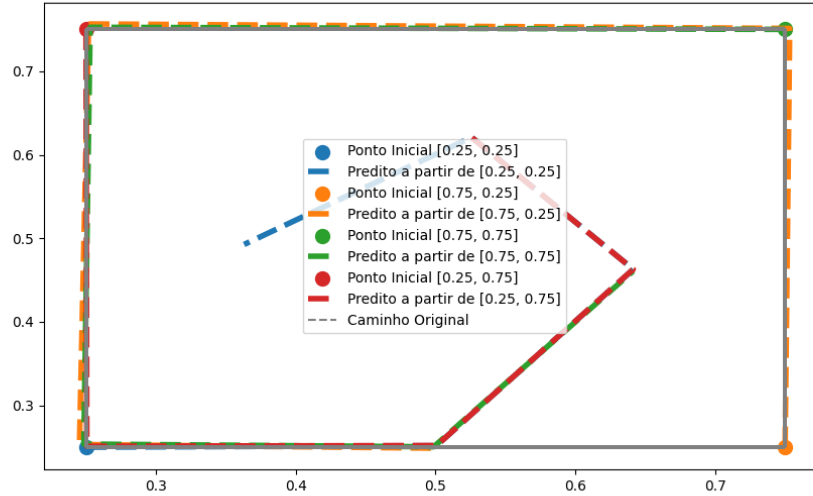
Como dito no passo (d) do item anterior, o treinamento é realizado utilizando a função "fit" do modelo. Utilizando a configuração inicial, com 300 épocas, o treinamento demorou cerca de 11 segundos.

3. Faça a previsão de algumas trajetórias, quando o ponto inicial varia. O que você conclui?

Resposta:

Foi feita a previsão para o ponto inicial original do código dado, ($x_1 = 0.25$ e $x_2 = 0.25$) e depois foram testados os outros vértices do quadrado. Os pontos são previstos em sequência, sendo a previsão anterior a entrada da próxima previsão, somando um total de 4 previsões para cada ponto inicial. O resultado pode ser observado na figura abaixo.

Figura 2: Previsão para diferentes pontos iniciais



Todas as previsões divergem do caminho original no ponto $(x_1 = 0.50$ e $x_2 = 0.25)$, o que indica uma provável incompatibilidade do modelo para prever esse caminho, uma vez que foram testadas diversas variações de número de épocas e de repetições do caminho original no treinamento.

4. Modifique a RNN usada e observe o que acontece.

Resposta:

Para este teste, o ponto inicial foi retornado para a configuração original ($x_1 = 0.25$ e $x_2 = 0.25$) e foram testadas diferentes combinações de épocas e número de repetições.

Tabela 1: Resultados das modificações

Nº Repetições	Épocas	Tempo Treinamento (s)	Tempo Total (s)
4	300	12	12.2
40	300	13.5	13.7
400	300	26.4	26.6
40	600	19.6	19.8
40	900	29.2	29.1

Pela tabela 1 é possível observar o aumento do tempo de execução, mais especificamente do tempo de treinamento, tanto com o aumento do número de repetições quanto com o aumento do número de épocas. Os resultados das previsões podem ser visualizados nas figuras abaixo.

Figura 3: Previsão para 4 repetições e 300 épocas

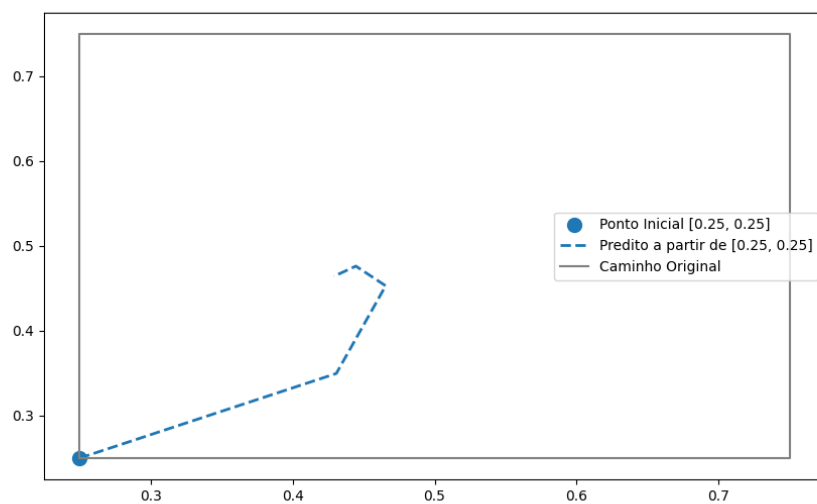


Figura 4: Previsão para 40 repetições e 300 épocas

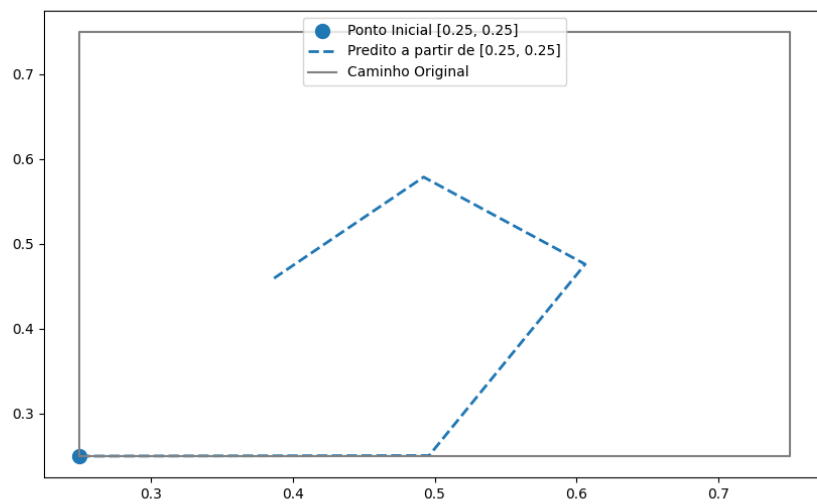


Figura 5: Previsão para 400 repetições e 300 épocas

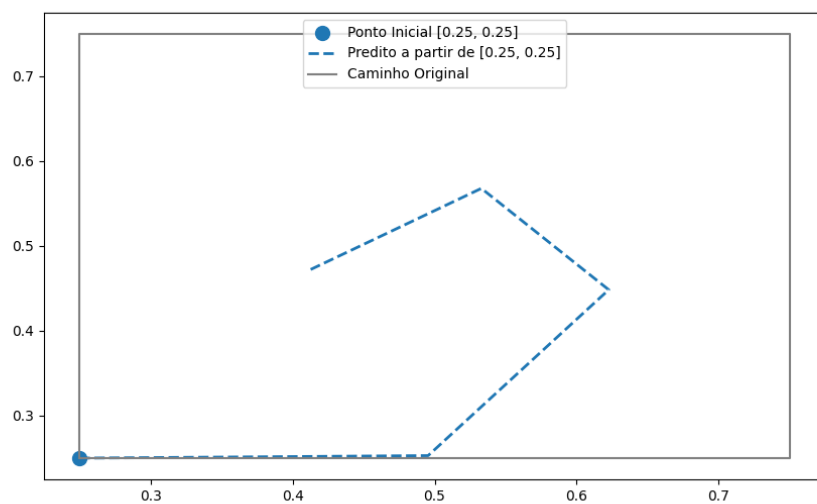


Figura 6: Previsão para 40 repetições e 600 épocas

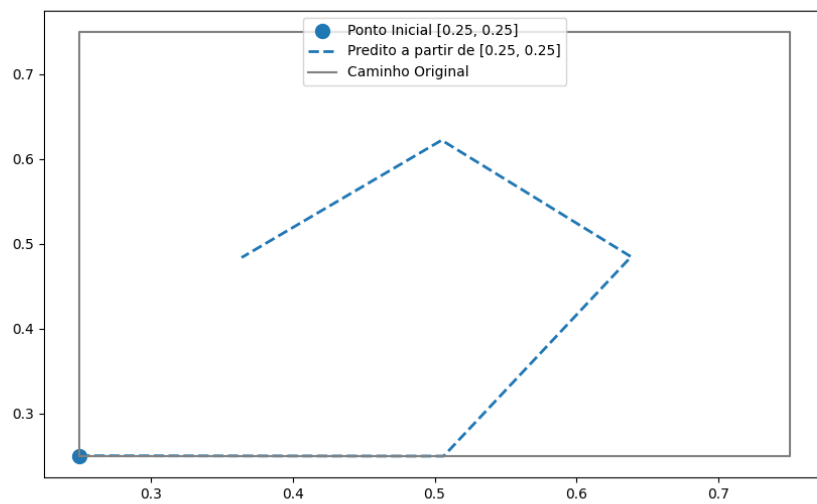
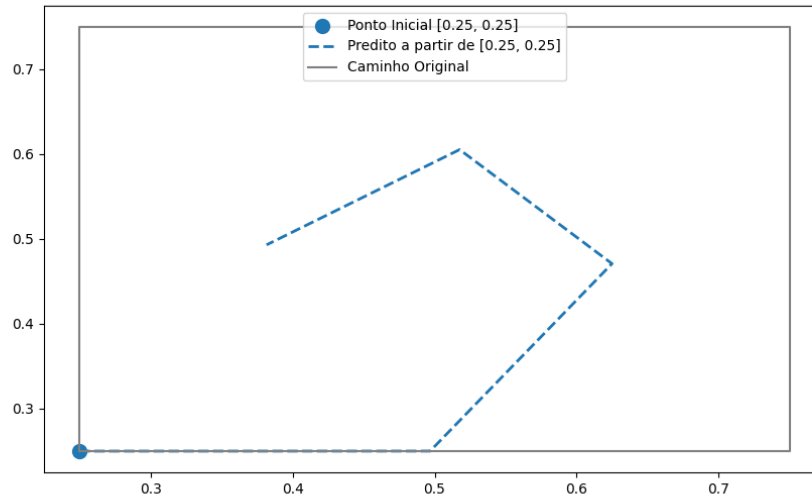


Figura 7: Previsão para 40 repetições e 900 épocas



5. Quais os pontos principais que você concluiu do artigo “Serial Order A Parallel Distributed Processing Approach”?

Resposta:

A teoria de Michael I. Jordan sobre ordem serial em sequências de ações usa redes neurais para entender e reproduzir a ordem das ações ao longo do tempo. Essas redes mantêm uma “memória” do que já aconteceu, usando conexões que alimentam as saídas de volta para as entradas, ajudando a lembrar das ações passadas. A rede aprende ajustando seus parâmetros para reduzir erros entre o que foi previsto e o que realmente aconteceu.

Isso faz com que a rede consiga generalizar a partir de sequências aprendidas e continuar funcionando bem, mesmo com pequenas perturbações. Essencialmente, a rede se torna uma memória dinâmica que pode voltar às suas trajetórias aprendidas, garantindo que as sequências de ações sejam produzidas corretamente, mesmo começando de pontos diferentes.

6. Qual a diferença da RNN usada no código em relação ao artigo “Serial Order A Parallel Distributed Processing Approach”?

Resposta:

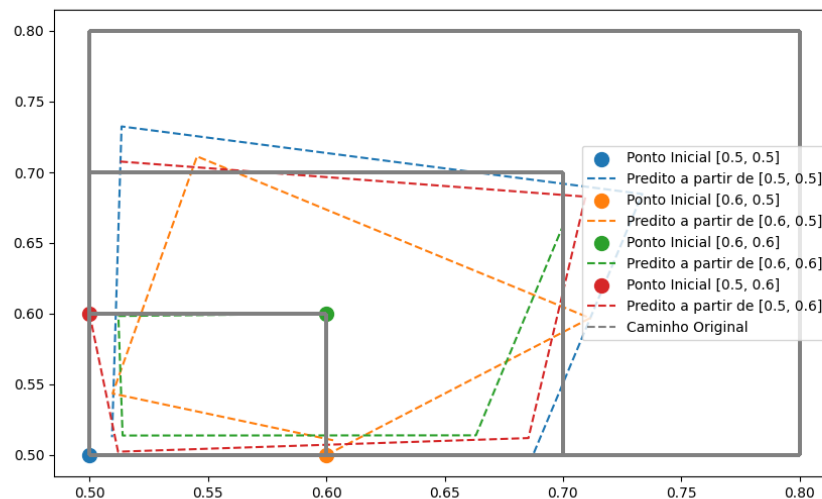
No artigo, a rede é composta por unidades de plano, estado e saída, com conexões recorrentes que definem a função de próximo estado, e enfatiza a capacidade de aprender e generalizar trajetórias no espaço de estado. Em contraste, a RNN do código usa uma arquitetura LSTM simples, que captura dependências temporais através de sua memória interna, sem distinções explícitas entre plano e estado. A abordagem de Jordan foca em representações distribuídas e paralelismo, enquanto a RNN do código adota técnicas convencionais de redes neurais recorrentes.

7. Inclua, nos dados de treino uma figura de uma espiral quadrada, e experimente o que a rede aprendeu.

Resposta:

Para gerar o caminho em espiral foi implementado o método *generate_spiral_square* (código no final do relatório), o qual gera uma lista no mesmo formato da *square_path* fornecida, porém com os pontos formando uma espiral quadrada. O resultado da previsão pode ser observado na figura abaixo.

Figura 8: Previsão do caminho espiral



Código

O código abaixo encontra-se no repositório <https://github.com/lhscaldas/cps769-ai-gen>, bem como o arquivo LaTeX com o relatório.

Código 1: código fornecido completo com algumas modificações

```
1 import os
2 import numpy as np
3 import tensorflow as tf
4 from tensorflow.keras import layers, models, Input
5 import matplotlib.pyplot as plt
6
7 # Desabilitar GPU para treinamento (se necessário)
8 os.environ['CUDA_VISIBLE_DEVICES'] = '-1'
9
10 # Desativar operações personalizadas oneDNN
11 os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'
12
13 class PathPredictor:
14     def __init__(self, path_type='quadrado', epochs=500, num_repeat=40):
15         self.path_type = path_type
16         self.epochs = epochs
17         self.model = self.build_model()
```



```

18     self.data = self.generate_data(num_repeat)
19     self.x_train, self.y_train = self.prepare_data()
20
21     def build_model(self):
22         model = models.Sequential([
23             Input(shape=(1, 2)),
24             layers.LSTM(50, activation='relu'),
25             layers.Dense(2)
26         ])
27         model.compile(optimizer='adam', loss='mse')
28         return model
29
30     def generate_square_path(self):
31         return np.array([
32             [0.25, 0.25],
33             [0.75, 0.25],
34             [0.75, 0.75],
35             [0.25, 0.75],
36             [0.25, 0.25]
37         ])
38
39     def generate_spiral_square(self, turns=5, step=0.1):
40         path = []
41         x, y = 0.5, 0.5
42         for turn in range(turns):
43             path.append([x, y])
44             x += step * (turn + 1)
45             path.append([x, y])
46             y += step * (turn + 1)
47             path.append([x, y])
48             x -= step * (turn + 1)
49             path.append([x, y])
50             y -= step * (turn + 1)
51             step /= 2
52         return np.array(path)
53
54     def generate_data(self, num_repeat):
55         if self.path_type == 'quadrado':
56             return np.tile(self.generate_square_path(), (num_repeat, 1))
57         elif self.path_type == 'espiral':
58             return np.tile(self.generate_spiral_square(), (num_repeat, 1))
59         else:
60             raise ValueError("path_type must be 'quadrado' or 'espiral'")
61
62     def prepare_data(self):
63         data = self.data
64         x_train = data[:-1].reshape(-1, 1, 2)
65         y_train = data[1:].reshape(-1, 2)
66         return x_train, y_train
67
68     def train_model(self):
69         self.model.fit(self.x_train, self.y_train, epochs=self.epochs, verbose
70                         =0)

```

```

71 def plot_predictions(self, initial_points):
72     plt.figure(figsize=(10, 6))
73
74     for point in initial_points:
75         current_input = np.array(point).reshape(1, 1, 2)
76         predictions = [point]
77
78         for _ in range(4):
79             next_prediction = self.model.predict(current_input)
80             predictions.append(next_prediction.flatten())
81             current_input = next_prediction.reshape(1, 1, 2)
82
83         predictions = np.array(predictions)
84
85         # Plotar o ponto inicial
86         plt.scatter(point[0], point[1], marker='o', s=100, label=f'Ponto
87             Inicial {point}')
88
89         # Plotar as previsões
90         plt.plot(predictions[:, 0], predictions[:, 1], label=f'Predito a
91             partir de {point}', linestyle='dashed', linewidth=2)
92
93         # Plotar o caminho original para referência
94         # plt.plot(self.data[:, 0], self.data[:, 1], label='Caminho Original',
95             # linestyle='dashed', color='gray')
96         if self.path_type == 'quadrado':
97             plt.plot(self.generate_square_path()[:, 0], self.
98                 generate_square_path()[:, 1], color='gray', label='Caminho
99                 Original')
100         elif self.path_type == 'espiral':
101             plt.plot(self.generate_spiral_square()[:, 0], self.
102                 generate_spiral_square()[:, 1], color='gray', label='Caminho
103                 Original')
104         plt.legend()
105         plt.show()
106
107 def selecao(epochs=300, num_repeat=4):
108     square_predictor = PathPredictor(path_type='quadrado', epochs=epochs,
109         num_repeat=num_repeat)
110     square_predictor.train_model()
111     initial_points_square = [
112         [0.25, 0.25],
113     ]
114     square_predictor.plot_predictions(initial_points_square)
115
116 def quadrado():
117     square_predictor = PathPredictor(path_type='quadrado')
118     square_predictor.train_model()
119     initial_points_square = [
120         [0.25, 0.25],
121         [0.75, 0.25],
122         [0.75, 0.75],
123         [0.25, 0.75]
124     ]

```

```

117     square_predictor.plot_predictions(initial_points_square)
118
119 def espiral():
120     spiral_predictor = PathPredictor(path_type='espiral')
121     spiral_predictor.train_model()
122     initial_points_spiral = [
123         [0.5, 0.5],
124         [0.6, 0.5],
125         [0.6, 0.6],
126         [0.5, 0.6]
127     ]
128     spiral_predictor.plot_predictions(initial_points_spiral)
129
130
131 if __name__ == "__main__":
132     # Seleção de hiperparâmetros
133     # selecao(epochs=100, num_repeat=40)
134
135     # Quadrado
136     # quadrado()
137
138     # Espiral quadrada
139     espiral()

```