پروژه پایانی درس (قسمت ب)

کیمیا اسماعیلی - 610398193

الگوریتم ژنتیک

مقدمه:

در این بخش پیاده‌سازی بازی کانکت 4 با استفاده از شبکه‌ی عصبی و الگوریتم ژنتیک انجام شده است. شبکه عصبی طراحی شده همه‌ی خانه‌های روی صفحه به همراه مقدار آن‌ها که می‌تواند خالی، دیسک بازیکن شماره ۱ و مهره بازیکن شماره ۲ باشد را دریافت می‌کند. این شبکه عصبی به عنوان خروجی شماره یکی از ستون‌ها را برای انداختن دیسک در این دست اعلام می‌کند.

به دلیل اینکه معیاری به تنهایی برای ارزیابی نحوه عملکرد این شبکه عصبی و بهتر کردن وزن‌ها نداریم، از الگوریتم ژنتیک استفاده شده است تا بهترین شبکه عصبی در طی نسل‌های مختلف انتخاب شود. در این روش پس از تولید جمعیت نمونه در هر نسل، به اندازه ۱۰۰ بار بین این نسل و بازیکن مینیماکس که در بخش الف توضیح داده شد بازی انجام می‌شود. تابع fitness به عنوان درصد بردهای بازیکن شبکه عصبی در این بازی‌ها تعیین شده است. این تابع fitness کمک می‌کند که بتوانیم به کمک الگوریتم شبکه عصبی در هر مرحله بهترین گونه‌های نسل را پیدا کنیم و مجددا جمعیت نمونه تولید کنیم تا در نهایت به حالت هدف دست پیدا کنیم.

توضیح کد:

در ابتدا تمام کتابخانه‌های مورد نیاز اضافه شد.

```
import numpy as np
import math
import pygad
import pygad.nn
import pygad.gann
```

همچنین توابعی که برای بخش الف و مینیماکس تعریف شده بود در این کد نیز اضافه شده‌اند تا شبکه عصبی بتواند با بازیکن مینیماکس رقابت کند و نحوه عملکرد هر شبکه عصبی در جمعیت نمونه مشخص شود.

برای استفاده از الگوریتم ژنتیک از کتابخانه pygad استفاده شد که اجازه می‌دهد تا شبکه‌ی عصبی مورد نظر خود را در طی نسل‌های مختلف با استفاده از فرایند ژنتیک بهبود دهیم.

به صورت زیر با استفاده از ماژول GANN شبکه‌های عصبی برای استفاده در فرایند ژنتیک ساخته می‌شوند:

```
gann = pygad.gann.GANN(num_solutions=5,
                       num_neurons_input=42,
                       num_neurons_hidden_layers=[32, 16],
                       num_neurons_output=7,
                       hidden_activations=["relu", "relu"],
                       output_activation="softmax"
                      )
```

در آن تعداد نورون‌های ورودی همان تعداد خانه‌های روی صفحه است که به شبکه عصبی ورودی داده می‌شود. لایه‌های میانی با تعداد نورون ۳۲ و ۱۶ مشخص شده‌اند. خروجی نیز ۷ نورون است که نشان‌دهنده‌ی هر کدام از ۷ ستون است. تابع activation نیز برای لایه‌های میانی و پایانی همان‌طور که در تصویر مشخص است تعیین شده است.

مهم‌ترین بخش تعیین تابع fitness است که ابتدا تابع کمکی تعریف شده است تا با کمک آن بتوانند مینیماکس و شبکه‌های عصبی بازی کنند و نتیجه گزارش شود.

```
def play_with_minimax(is_neural_net_first_player, idx):
    is_neural_net_winner = None
    board = create_board()
    if is_neural_net_first_player:
        game_over = False
        while not game_over:
            #MiniMax Turn
            col, minimax_score = minimax(board, 5, -math.inf, math.inf, False)
            row = get_next_open_row(board, col)
            drop_piece(board, row, col, 0)
            if winning_move(board, 0):
                game_over = True
                is_neural_net_winner = False
            #Neural Net Turn
            pred = pygad.nn.predict(last_layer=gann.population_networks[idx],
                                    data_inputs=board
                                   )
            row = get_next_open_row(board, pred)
            drop_piece(board, row, col, 1)
            if winning_move(board, 1):
                game_over = True
                is_neural_net_winner = True
```

در این تابع دو حالت مختلف امکان دارد. اولی این است که شبکه عصبی ابتدا شروع کند و دیگری اینکه مینیماکس ابتدا شروع به بازی کند.
در حالت اول، به این صورت بازی انجام می‌شود که با استفاده از تابع minimax یک ستون انتخاب می‌شود و دیسک بازیکن مینیماکس روی صفحه قرار می‌گیرد. سپس بررسی می‌شود که آیا این عمل منجر به برد این بازیکن شده است یا خیر. در ادامه، دیسک بازیکن شبکه عصبی قرار داده می‌شود. با

استفاده از تابع predict موجود در ماژول pygad.nn این کار انجام می‌شود. این تابع یکی از شبکه‌های عصبی در جمعیت نمونه را با استفاده از idx ورودی دریافت می‌کند و خروجی شبکه‌ی عصبی را اعلام می‌کند. مشابه با مینیماکس پس از انجام این مرحله، دیسک انداخته می‌شود و بررسی می‌شود که آیا منجر به برد بازیکن شبکه عصبی شده است یا خیر. این فرایند تکرار می‌شود تا در نهایت یکی از دو بازیکن پیروز شوند.
حالتی که شبکه عصبی شروع کند در شکل زیر مشخص شده است:

```
    else:
        game_over = False
        while not game_over:
            #Neural Net Turn
            pred = pygad.nn.predict(last_layer=gann.population_networks[idx], data_inputs=board)
            row = get_next_open_row(board, pred)
            drop_piece(board, row, col, 1)
            if winning_move(board, 1):
                game_over = True
                is_neural_net_winner = True
            #MiniMax Turn
            col, minimax_score = minimax(board, 5, -math.inf, math.inf, True)
            row = get_next_open_row(board, col)
            drop_piece(board, row, col, 0)
            if winning_move(board, 0):
                game_over = True
                is_neural_net_winner = False
    return is_neural_net_winner
```

این بخش نیز مشابه توضیح قبلی است با این تفاوت که ترتیب بازی تفاوت می‌کند و در تابع minimax بیشینه یا کمینه کردن متفاوت شده است.

به کمک این تابع کمکی، تابع fitness به شکل زیر تعریف شد:

```
def fitness(sol, idx):
    count_neural_net_wins = 0
    for i in range(50):
        result = play_with_minimax(True, idx)
        if result == True:
            count_neural_net_wins += 1
    for i in range(50):
        result = play_with_minimax(False, idx)
        if result == True:
            count_neural_net_wins += 1
    return count_neural_net_wins / 100
```

این تابع به این صورت عمل می‌کند که ۵۰ بار شبکه عصبی به عنوان بازیکن اول و ۵۰ بار مینیماکس به عنوان بازیکن اول با یکدیگر بازی می‌کنند. تعداد بردهای شبکه‌ی عصبی مورد بررسی از این فرایند مشخص می‌شود و درصد برد شبکه عصبی نمونه به عنوان خروجی این تابع گزارش می‌شود.

همچنین برای آپدیت شدن وزن‌های شبکه‌های عصبی یک callback تعریف شده است که وزن‌ها توسط pygad آپدیت می‌شوند.

```
def callback(genetic_algorithm_instance):
  population_matrices = pygad.gann.population_as_matrices(population_networks=gann.population_networks,
                                                          population_vectors=genetic_algorithm_instance.population)
  gann.update_population_trained_weights(population_trained_weights=population_matrices)
```

در نهایت الگوریتم ژنتیک با استفاده از تکه کد زیر ساخته شد:

```
genetic_algorithm_instance = pygad.GA(num_generations=100,
                                      num_parents_mating=4,
                                      initial_population=pygad.gann.population_as_vectors(population_networks=gann.population_networks).copy(),
                                      fitness_func=fitness,
                                      mutation_percent_genes=10,
                                      callback_generation=callback
                                     )
```

در آن پارامترهای الگوریتم ژنتیک مشخص شده است. Num_generations مشخص می‌کند که چند نسل باید الگوریتم اجرا شود که ۱۰۰ بار تعیین شده است. همچنین num_parents_mating تعیین می‌کند که چند نمونه در نسل قبل بایستی ترکیب شوند تا یک نمونه در نسل جدید شکل پیدا کند.
Initial_population جمعیت اولیه را تعیین می‌کند. تابع fitness که پیش‌تر تعریف کردیم نیز به این الگوریتم ارسال شده است.
برای اینکه الگوریتم ژنتیک به خوبی کار کند لازم است که در بعضی از نمونه‌های هر نسل جهش رخ دهد تا بتوانیم به حالت‌های بهتر برسیم که در این تمرین ۱۰ درصد تعیین شده است. در نهایت callback نیز برای آپدیت شدن وزن‌ها مشخص شده است.

با استفاده از کد run که در تکه کد زیر مشخص است الگوریتم می‌تواند اجرا شود:

```
genetic_algorithm_instance.run()
```