Badania operacyjne - gra SNAKE

Michał Lampert, Wojtek Achtelik, Tomasz Lamża

13 kwietnia 2022

1 Temat projektu:

Zastosowanie oraz porównanie algorytmów genetycznych oraz algorytmu PSO do otrzymania odpowiednich wag w sieci neuronowej dla problemu SNAKE.

2 Podejście

Sieć neuronowa na podstawie aktualnego ułożenia planszy, powinna zwracać możliwie najlepszy w danej sytuacji ruch węża. Celem jest znalezienie optymalnego genotypu G, czyli wag sieci neuronowej, tak aby maksymalizować oczekiwany końcowy wynik w grze.

3 Model matematyczny:

3.1 Funkcja fitnessu

Poszczególną grę możemy zidentyfikować jako ciąg pozycji jabłek na planszy:

$$a = a_1, a_2, a_3, ... = (x_1, y_1), (x_2, y_2), (x_3, y_3), ...$$

Wynik modelu w jednej grze możemy zapisać jako funkcję

$$f(G, a)$$

zwracającą wynik gry, uzyskany poprzez wyliczanie lokalnie najlepszego ruchu według obecnego modelu w danym ułożeniu, aż do skończenia rozgrywki.

Funkcję fitnessu, którą będziemy starali się optymalizować jest:

$$F(G) = avg\{f(G, a)|a \in A\}$$

3.2 Analiza sytuacji węża w danym momencie

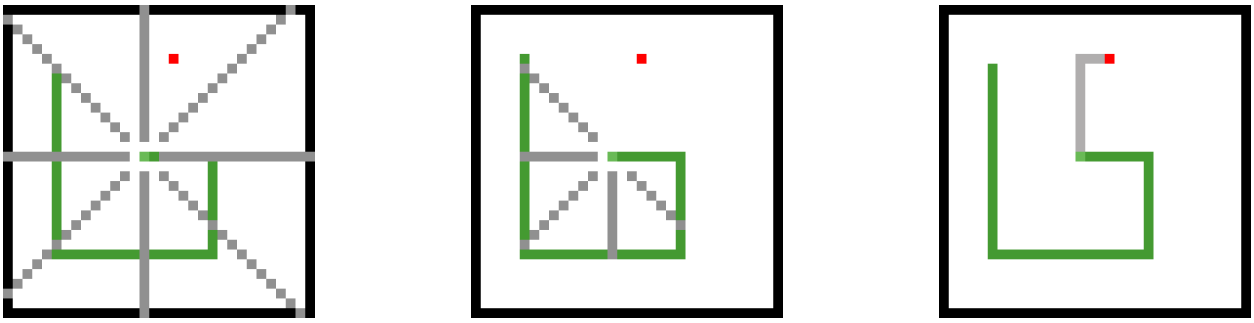
Na początku założyliśmy, że model będzie wyliczał najlepszą lokalnie wartość na podstawie 20 wartości (zmiany będą wytłumaczone w dalszej części pracy) - odległości od siebie samego, od ścian i od jabłka. Funkcja opisująca ruch węża, będzie posiadać 20 zmiennych wejściowych

(określające odległość w 8 kierunkach od ścian i samego siebie oraz 4 określające odległość od jabłka) oraz 4 wyjściowe opisujące skłonność do ruchu w danym kierunku.

$$DISTANCES = [x_1, x_2, .., x_8, y_1, .., y_8, z_1, .., z_4]$$

Gdzie odpowiednio:

- $x_1, x_2, .., x_8$ - odległości od ścian tak jak na rysunku
- $y_1, y_2, .., y_8$ - odległości od węża tak jak na rysunku
- z_1, z_2, z_3, z_4 - odległości od jabłka



Rysunek 1: Odległości od ścian (lewo), od siebie (środek) i od jabłka (prawo)

W każdym momencie wąż na podstawie odległości od ścian, siebie samego i jabłka będzie zwracał wynik reprezentowany za pomocą 4 wartości:

$$RESULT = [d1, d2, d3, d4]$$

Każda ze zmiennych d_i opisuje skłonność węża do ruchu w odpowiednim kierunku (1 -lewo, 2 - góra, 3 - prawo, 4 - dół).

3.3 Sposób reprezentacji danego węża:

Wąż będzie reprezentowany jako ciąg pozycji komórek na których się znajduje:

$$SNAKE = s_1, s_2, s_3, ...$$

3.4 Dodatkowe założenia

3.4.1 Przybliżenie ewaluacji

Z oczywistych względów, mało wydajne czasowo, byłoby wyliczanie rzeczywistej średniej na podstawie wszystkich możliwych ułożeń planszy.

Zamiast tego wartość oczekiwaną końcowego wyniku dla konkretnego modelu, będziemy przybliżać za pomocą średniej z kilku gier (zazwyczaj 10). Pozwoli to zaoszczędzić dużo czasu. Oczywiście zwiększa to w pewnym stopniu losowość ewaluacji modelu, jednak wydaje się, że może to mieć również korzystny wpływ na zapobieganie dążenia do lokalnego maksimum.

3.4.2 "Małe punkty"

Przy ewaluacji modelu, oprócz punktów za zdobywanie jabłek, przyznawaliśmy "małe punkty" za długość życia. Zdecydowaliśmy się na to, ponieważ zdecydowana większość początkowych modeli uzyskuje 0 punktów. Chcieliśmy odsiać te, które otrzymują taki wynik poprzez uderzenie węża w siebie, a zostawić te które robią chociaż kilka ruchów.

Ilość ruchów ograniczyliśmy, aby z kolei uniknąć zapętlenia się ruchów węża. Po wykorzystaniu limitu, gra się kończy

4 Implementacja

4.1 Język programowania

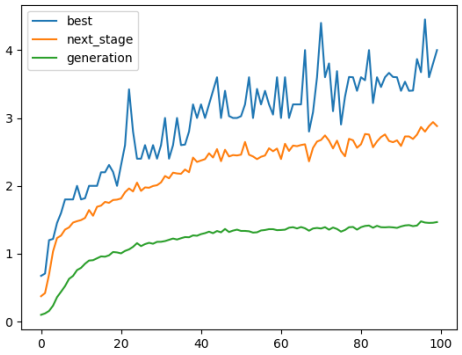
Program starający się rozwiązać problem zaimplementowaliśmy w czystym języku python. Po jakimś czasie jednak, przepisaliśmy część kodu, aby wykorzystywała bibliotekę numpy, a niektóre fragmenty do języka cython. Pozwoliło to testować więcej modeli (wcześniej 100, teraz 400) na większej liczbie gier (wcześniej 5, teraz 10) w podobnym czasie.

4.2 Sieć neuronowa

Na podstawie genotypu budowaliśmy sieć neuronową (na początku bez warstw ukrytych).

Wykorzystaliśmy (według naszej, ubogiej wiedzy) klasyczne podejście, czyli jeśli chcemy uzyskać 4 wartości końcowe, powinniśmy użyć 4 równoległe sieci neuronowe zwracające po jednej wartości.

Z tego powodu do optymalizacji mieliśmy 80 niepowiązanych ze sobą wag.



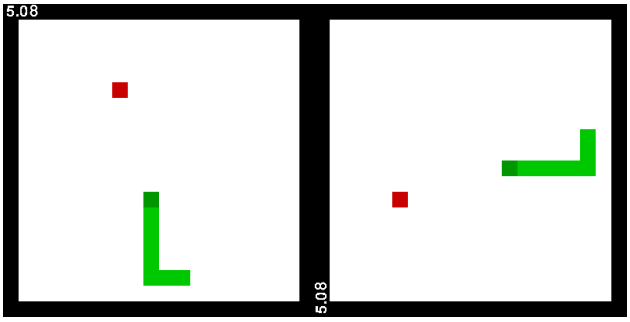
Rysunek 2: Wyniki pierwszej implementacji modelu - algorytm genetyczny.

Patrząc na wyniki zdaliśmy sobie sprawę, że nie jest to najlepsze podejście.

4.3 Zmniejszenie liczby wymiarów

Słaby wynik jest spowodowany "przekleństwem wymiarowości". Aby zmniejszyć liczbę wymiarów, wykorzystaliśmy kilka obserwacji:

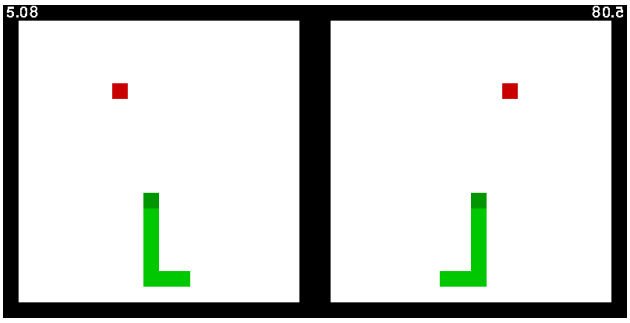
4.3.1 Środek symetrii



Rysunek 3: Analogiczne ułożenie planszy, tylko obrócone o 90°

Dla tej sytuacji (rys.3) warto zauważyć, że są identyczne, z dokładnością do obrotu wokół środka symetrii planszy. Dlatego jeśli dla sytucji 1. ruch który sugeruje model to UP - dla sytuacji 2. powinno to być LEFT.

4.3.2 Oś symetrii

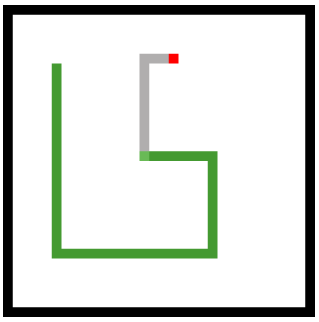


Rysunek 4: Analogiczne ułożenie planszy, tylko odbite lustrzanie

Dla tych sytuacji (rys.4) warto zauważyć, że są identyczne, z dokładnością do odbicia względem osi symetrii planszy. Dlatego jeśli dla sytucji 1. ruch który sugeruje model to UP - dla sytuacji 2. również powinno to być UP.

4.3.3 Odległość od jabłka

Warto również zauważyć, że odległość w 4 kierunkach (rys.5, nawet nie da się tego za bardzo zaznaczyć na rysunku) nie ma sensu. Wystarczą 2 wartości, ponieważ pozostałe 2 będą takie same z dokładnością co do znaku.



Rysunek 5: Odległość od jabłka "w 4 kierunkach"

4.3.4 Podsumowanie minimalizacji wymiarowości

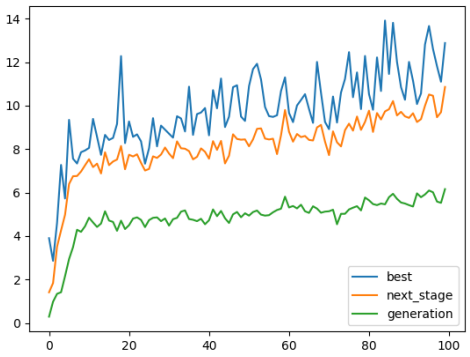
Dzięki powyższym obserwacjom, udało się zminimalizować model (genotyp) do 12 wartości (w przypadku sieci bez warstw ukrytych). Teraz jako dane wejściowe przekazujemy macierz odległości o wymiarach 4x12, która jest "skonstruowana" na podstawie 20 wartości (odległości od ścian, siebie i jabłka), dzięki czemu na końcu otrzymujemy macierz 1x4, która odpowiada wartościom sugestii danego ruchu

$$DISTANCES = [x_1, x_2, .., x_8, y_1, .., y_8, z_1, .., z_4]$$

$$INPUT = \begin{bmatrix} x_1 & (x_2+x_8)/2 & (x_3+x_7)/2 & (x_4+x_6)/2 & x_5 & y_1 & (y_2+y_8)/2 & (y_3+y_7)/2 & (y_4+y_6)/2 & y_5 & z_1 & z_2 \\ x_3 & (x_2+x_4)/2 & (x_1+x_5)/2 & (x_8+x_6)/2 & x_7 & y_3 & (y_2+y_4)/2 & (y_1+y_5)/2 & (y_8+y_6)/2 & y_7 & -z_2 & z_1 \\ x_5 & (x_4+x_6)/2 & (x_3+x_7)/2 & (x_2+x_8)/2 & x_1 & y_5 & (y_4+y_6)/2 & (y_3+y_7)/2 & (y_2+y_8)/2 & y_1 & -z_1 & -z_2 \\ x_7 & (x_6+x_8)/2 & (x_1+x_5)/2 & (x_2+x_4)/2 & x_3 & y_7 & (y_6+y_8)/2 & (y_1+y_5)/2 & (y_2+y_4)/2 & y_3 & z_2 & -z_1 \end{bmatrix}$$

4.3.5 Wyniki po redukcji wymiarowości

Dzięki redukcji wymiarów, uzyskane rezultaty od razu okazały się dużo lepsze



Rysunek 6: Wyniki po zmniejszeniu liczby wymiarów - algorytm genetyczny

4.4 Warstwy ukryte

W późniejszej fazie testów dodaliśmy możliwość korzystania z sieci neuronowej z warstwami ukrytymi. W przypadkach testowych wykorzystywaliśmy 2 dodatkowe warstwy składające się z 6 oraz 4 neuronów, więc w sumie optymalizowaliśmy wagi sieci neuronowej składającej się z 12+6+4+4 neuronów. Jako funkcje na warstwach pośrednich wykorzystaliśmy: pierwiastek kwadratowy oraz funkcję sigmoid

4.5 Początkowa populacja

Jako początkowe genotypy, losowaliśmy 400 modeli składających się z losowych wag w sieci neuronowej z zakresu [-1,1]

```
first_generation(N){
  population = []
  for i in 1:N:
    g = random_genotype()
    population.add(g)
  return population
}
```

5 Algorytmy

Wykorzystaliśmy i staraliśmy się porównać 2 algorytmy: genetyczny oraz PSO

5.1 Algorytm genetyczny

5.1.1 Krzyżowanie

Testowaliśmy 4 krzyżowania:

- krzyżowanie losowe - dla dwóch genotypów, losowaliśmy 1 gen spośród dwóch na danej pozycji

```
cross(g1, g2){
  new_genotype = new Genotype()
  for i in new_genotype:
    r = random_bit()
    if r == 0:
      new_genotype[i] = g1[i]
    else:
      new_genotype[i] = g2[i]
  return new_genotype
}
```

- ważone krzyżowanie losowe - tak jak powyżej, jednak geny z pierwszego gonotypu losowane są z większym prawdopodobieństwem

```
cross(g1, g2){
  new_genotype = new Genotype()
  for i in new_genotype:
    r1 = random_bit()
    r2 = random_bit()
    if r1 * r2 == 0:
      new_genotype[i] = g1[i]
    else:
      new_genotype[i] = g2[i]
  return new_genotype
}
```

- średnia - dla dwóch genotypów, wyliczaliśmy gen jako średnią dwóch na danej pozycji

```
cross(g1, g2){
  new_genotype = new Genotype()
  for i in new_genotype:
    new_genotype[i] = (g1[i] + g2[i]) / 2
  return new_genotype
}
```

- średnia ważona - tak jak powyżej, jednak geny z pierwszego mają wyższą wagę przy wyliczaniu średniej

```
cross(g1, g2, ratio){
  new_genotype = new Genotype()
  for i in new_genotype:
    new_genotype[i] = g1[i]*ratio + g2[1]*(1-ratio)
  return new_genotype
}
```

5.1.2 Mutacje

Mutowany gen był przemnażany przez losową wartość z przedziału [0.9,1.1]. Część testów przeprowadziliśmy dla $\epsilon = 0.3$ - każdy nowy gen miał 30% szans na mutację, a część dla $\epsilon = 1$ - wtedy wszystkie genotypy były mutowane.

```
mutate(g1, eps){
  r = random()
```

```
  if r < eps:
    for i in g1:
      weight = random(0.9, 1.1)
      g1[i] *= weight
  return g1
}
```

5.1.3 Generacja

Jedna generacja zazwyczaj liczyła $n = 400$ osobników. Ocenialiśmy je metodą opisaną w 3.4. Zostawialiśmy $\sqrt{n} = 20$ najlepszych genotypów (pierwiastek z liczebności generacji, aby mieć mniej więcej stałą liczebność zbioru genotypów). Następnie krzyżowaliśmy genotypy każdy z każdym, jedną z metod i dokonywaliśmy mutacji.

5.1.4 Pierwsza populacja

Pierwsza populacja była losowana z rozkładem jednostajnym - każda waga sieci była losowana z zakresu [-1, 1].

```
random_genotype(){
  g = new Genotype()
  for i in g:
    g[i] = random(-1,1)
  return g
}
```

5.1.5 Pseudokod algorytmu genetycznego:

```
next_generation(population, N, f, eps){
  fitnesses = []
  for g in population:
    fitness = f(g)
    fitnesses.add([fitness, g])
  fitnesses = sort_by(fitnesses, [f,g] -> f)
  elite = take_first(fitnesses, sqrt(N))
  elite = map(elite, [f,g] -> g)
  new_population = []
  for g1 in elite:
    for g2 in elite:
      new_gene = cross(g1, g2)
      new_gene = mutate(new_gene, eps)
      new_population.add(new_gene)
  return new_population
}
```

5.2 Algorytm PSO

Dla algorytmu PSO testowaliśmy rozwiązania w zależności od wag:

$$\omega \in [0.05, 1], c_1, c_2 \in [0.5, 2.5]$$

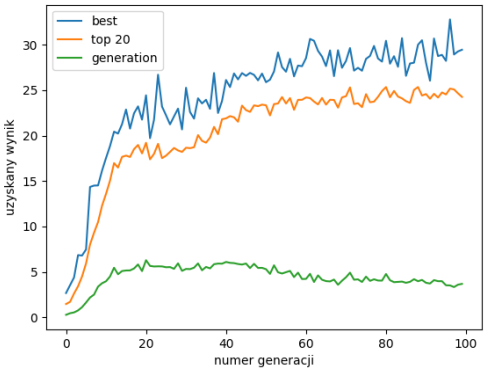
5.2.1 PSO - Pseudokod:

```
for each particle i = 1, ..., S do
    Initialize the particle's position with a uniformly distributed random vector: x_i ~ U(b_lo, b_up)
    Initialize the particle's best known position to its initial position: p_i ← x_i
    if f(p_i) < f(g) then
        update the swarm's best known position: g ← p_i
    Initialize the particle's velocity: v_i ~ U(-|b_up-b_lo|, |b_up-b_lo|)
while a termination criterion is not met do:
    for each particle i = 1, ..., S do
        for each dimension d = 1, ..., n do
            Pick random numbers: r_p, r_g ~ U(0,1)
            Update the particle's velocity: v_i,d ← ω v_i,d + φ_p r_p (p_i,d-x_i,d) + φ_g r_g (g_d-x_i,d)
        Update the particle's position: x_i ← x_i + lr v_i
        if f(x_i) < f(p_i) then
            Update the particle's best known position: p_i ← x_i
            if f(p_i) < f(g) then
                Update the swarm's best known position: g ← p_i
```

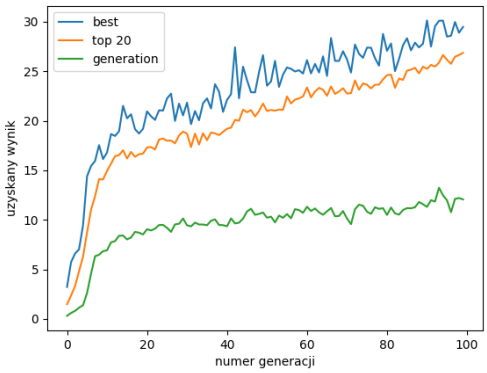
6 Wyniki

6.1 Algorytm genetyczny

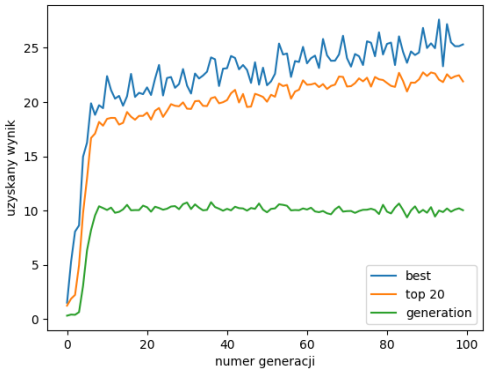
6.1.1 krzyżowanie losowe



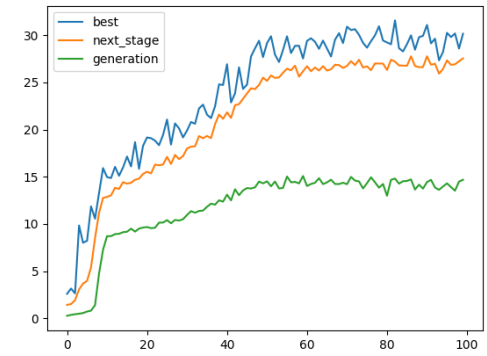
6.1.2 ważone krzyżowanie losowe



6.1.3 średnia

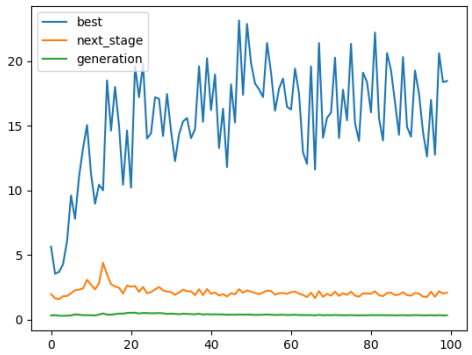


6.1.4 średnia ważona

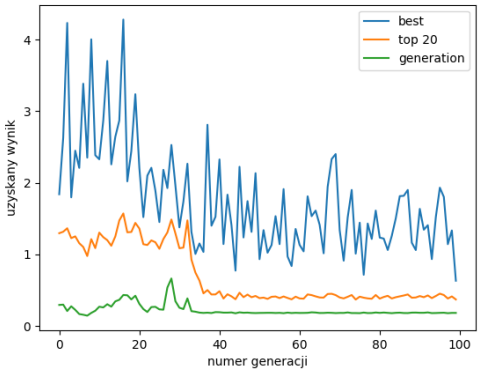


6.2 Algorytm PSO

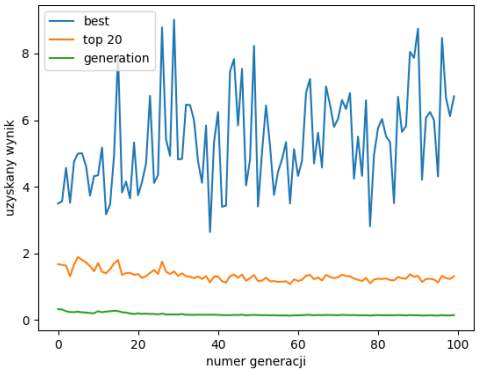
6.2.1 $\omega = 0.1, c_1 = 1, c_2 = 1$



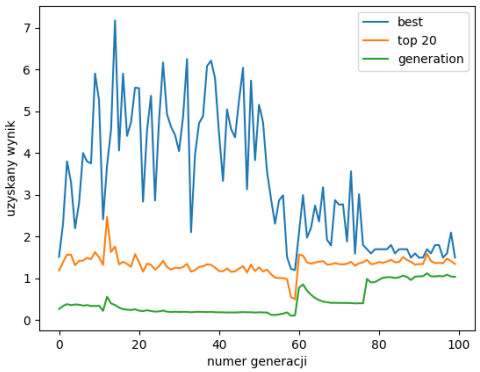
6.2.2 Inne kombinacje parametrów



Rysunek 7: $\omega = 0.1, c_1 = 0.5, c_2 = 2$



Rysunek 8: $\omega = 0.1, c_1 = 2, c_2 = 0.5$



Rysunek 9: $\omega = 0.1, c_1 = 2, c_2 = 2$.

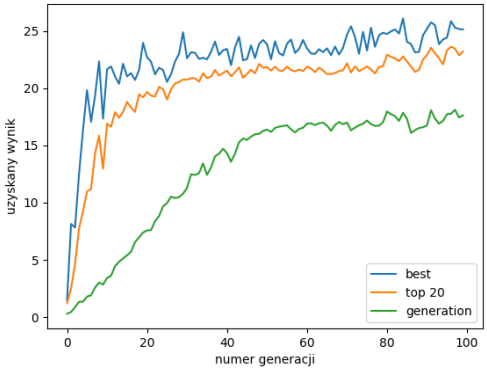
6.2.3 Zmniejszenie genotypu

Obserwując wyniki dla czterech powyższych kombinacji parametrów w algorytmie PSO zauważyliśmy, że wyniki są bardzo chaotyczne i słabe (oprócz pierwszego przypadku), a do tego nie zbiegają (przynajmniej na tak krótkim dystansie) do żadnej konkretnej wartości.

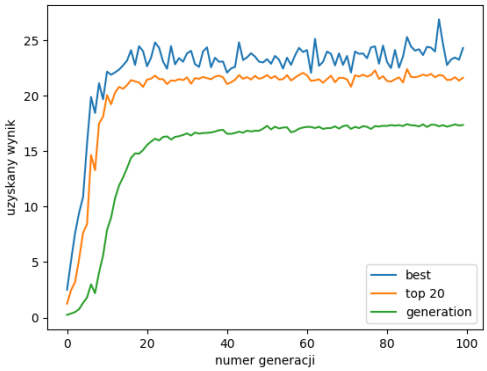
Postanowiliśmy więc zmniejszyć wielkość sieci neuronowej z 12+6+4+4 neuronów na 12+4+4, co pozwoliło zmniejszyć ilość parametrów do optymalizacji około dwukrotnie.

Poniższe wyniki algorytmu PSO są wyliczane dla właśnie takiej - zmniejszonej sieci.

6.2.4 $\omega = 0.729, c_1 = 2.05, c_2 = 2.05$



6.2.5 $\omega = 0.4, c_1 = 1.8, c_2 = 2.2$



7 Testy na innych planszach

Dla uzyskanych najlepszych modeli, sprawdziliśmy jakie wyniki uzyskamy na planszach o innych wymiarach:

- 12x12
- 30x30

- 25x15

I porównywaliśmy wyniki ze standardową planszą, na której model był uczony (20x20)

7.1 Algorytmy genetyczne

wymiar	wynik
20x20	24.35
12x12	14.81
30x30	15.28
25x15	4.0

Tabela 1: krzyżowanie losowe

wymiar	wynik
20x20	23.61
12x12	13.76
30x30	23.88
25x15	0.85

Tabela 2: ważone krzyżowanie losowe

wymiar	wynik
20x20	19.34
12x12	12.97
30x30	21.98
25x15	0.56

Tabela 3: krzyżowanie średnią

wymiar	wynik
20x20	5.76
12x12	5.49
30x30	2.8
25x15	1.57

Tabela 4: krzyżowanie średnią ważoną

7.2 Algorytmy PSO

wymiar	wynik
20x20	19.62
12x12	11.79
30x30	24.23
25x15	0.51

Tabela 5: $\omega = 0.729, c_1 = 2.05, c_2 = 2.05$

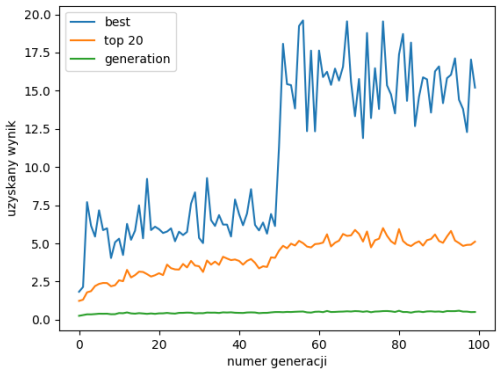
wymiar	wynik
20x20	17.53
12x12	11.74
30x30	19.68
25x15	0.5

Tabela 6: $\omega = 0.4, c_1 = 1.8, c_2 = 2.2$

8 Losowe genotypy

Na końcu postanowiliśmy przetestować, czy wszystko co do tej pory zrobiliśmy w ogóle miało sens.

Nie korzystając z żadnych metod polepszania genotypów, wygenerowaliśmy ok. 40 000 genotypów (400 losowych genotypów w 100 generacjach - podobna ilość do tej którą sprawdzaliśmy w każdym przypadku testowym)



Rysunek 10: Wyniki losowych genotypów

Całe szczęście ta metoda nie dała lepszych rezultatów zarówno od algorytmu genetycznego, jak i od PSO. Oczywiście wszystko jest kwestią szczęścia, ale tylko 2 razy poprawiliśmy najlepszy genotyp (ok. 5 i 50 ”generacji”)

wymiar	wynik
20x20	15.87
12x12	11.92
30x30	6.71
25x15	0.22

Tabela 7: Losowe genotypy na innych planszach

Co ciekawe, w przeciwieństwie do poprzednich modeli, na większej planszy model osiąga niski wynik w porównaniu do 20x20.

9 Wnioski

Na podstawie tych kilku przypadków testowych, możemy wyciągnąć kilka wniosków:

1. Przy zastosowaniu algorytmów genetycznych, uzyskiwaliśmy z reguły lepsze rezultaty, ok. 30 pkt, przy wynikach PSO - ok. 25
2. Dodatkowo na podstawie monotoniczności wyników, można wyciągnąć wniosek, że te mogłyby jeszcze ulec poprawie, przy algorytmie PSO, najlepszy wynik stabilizował się szybciej - ok. 20 kroku
3. Przy algorytmie PSO z kolei uzyskiwaliśmy lepszy średni wynik w całej populacji (ok. 16 vs. ok. 10)
4. Na kwadratowych planszach o innych wymiarach nasze modele poradziły sobie całkiem nieźle. Zaskoczyło nas jedynie, że na planszach o nieregularnych wymiarach wyniki były bardzo złe (zazwyczaj prawie jak dla genotypów losowych)
5. Przy 40 000 losowych genotypów nie udało się (całe szczęście) uzyskać lepszego modelu, co oznacza że problem nie jest trywialny

10 Podsumowanie

Generalnie jesteśmy zadowoleni z rezultatów naszej pracy - przy pomocy 2 różnych podejść byliśmy w stanie uzyskać wagi sieci neuronowej, które pozwalały na długie rozgrywki. Dodatkowo ważne było podejście iteracyjne - w każdym kolejnym kroku ulepszaliśmy jakąś część naszego projektu:

1. Implementacja algorytmu genetycznego,
2. Zmniejszenie liczby wymiarów,
3. Implementacja PSO,
4. Dodanie warstw sieci neuronowej,

5. Wydajniejsza implementacja - cython i numpy,
6. Zmniejszenie wielkości sieci neuronowej do PSO.

11 Literatura

1. Wykłady z przedmiotu BO :)
2. Dyskusja na Research Gate na temat parametrów PSO,
3. Wikipedia: https://en.wikipedia.org/wiki/Particle˙swarm˙optimization
4. Introduction to Evolutionary Algorithms - Yu, Gen
5. Do projektu zainspirował nas ten filmik