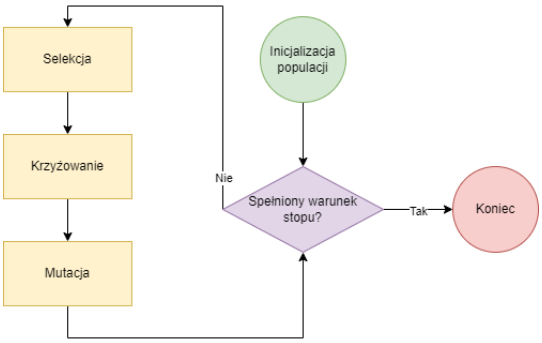
NN - Raport

Aleksandra Samsel - 313396

1 Opis zadania

Ostatnia tercja przedmiotu zakładała implementację algorytmu genetycznego. Polega on na inicjalizacji populacji, następnie wybrane osobniki są krzyżowane a niektóre z nich dodatkowo poddaje się procesowi mutacji. Ten ciąg operacji wykonuje się odpowiednią liczbę razy do uzyskania założonego wyniku funkcji celu bądź ustaloną liczbę generacji. Algorytm ma na celu tworzenie coraz lepszej populacji z której wybiera się najlepszego osobnika.



Ten projekt przewidywał implementację algorytmu a następnie przetestowanie go dla różnych zadań. Pierwsze testy wykonujemy dla prostych funkcji kilku zmiennych. Następnie główną częścią projektu jest implementacja algorytmu genetycznego, który maksymalizuje wartość prostokątów umieszczonych w okręgu o zadanym promieniu. Na koniec osobnikami, które krzyżujemy są sieci MLP, dla których również wykonujemy testy dla różnych zestawów danych.

2 Implementacja algorytmu genetycznego

Pierwsze zadanie polegało na implementacji algorytmu genetycznego i przetestowaniu go dla funkcji kwadratowej i pięciowymiarowej funkcji Rastrigina:

$$f(x, y, z) = x^2 + y^2 + 2z^2$$

$$f(\overline{x}) = An + \sum_{i=1}^{n}[x_i^2 - A\cos(2\pi x_i)] \quad (A = 10,\ n = 5)$$

Zadaniem algorytmu była minimalizacja, zatem w obydwu przypadkach oczekiwanymi wynikami były punkty o wszystkich współrzędnych równych 0 i wartości funkcji równej 0.

Algorytm korzystał z następujących metod:

- **selekcja turniejowa** - polega na losowym wyborze k osobników a następnie wybranie z nich najlepszego - wartość funkcji w tym punkcie miała być najmniejsza
- **selekcja ruletkowa** - każdy osobnik z populacji ma do siebie przypisaną wartość funkcji celu, przypisujemy do nich odpowiednie prawdopodobieństwo, w tym przypadku im mniejsza wartość funkcji celu tym większe prawdopodobieństwo i z nim następnie losujemy kolejno potrzebną liczbę rodziców

- **krzyżowanie jednopunktowe** - polega na wyborze punktu podziału z długości wektora argumentów a następnie lewa część (do punktu podziału) pierwszego rodzica i prawa część (od punktu podziału) drugiego rodzica tworzy pierwsze dziecko, drugie dziecko powstaje z pozostałych części
- **mutacja gaussowska** - polega wyborze losowego argumentu i dodaniu do niego losowej wartości z rozkładu normalnego o średniej 0 i odchyleniu standardowym równym 1

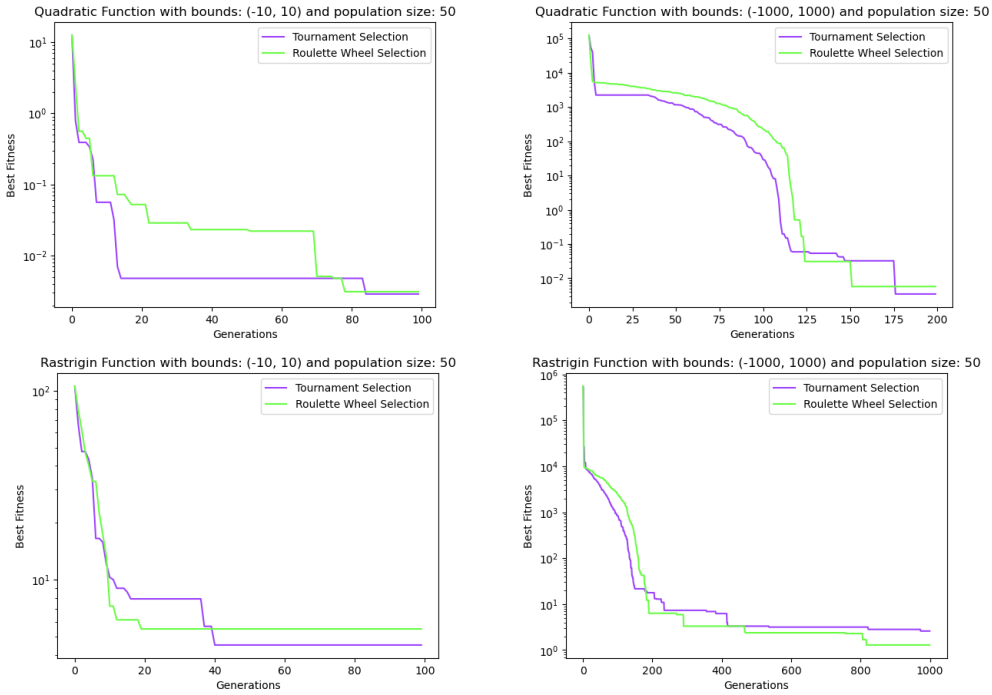
Aby sprawdzić działanie algorytmu zaczęłam od prostych eksperymentów. Poniżej przedstawione są parametry dla obydwu funkcji:

	Funkcja kwadratowa	Funkcja Rastrigina
Przedział losowania pierwszej populacji	[-1000, 1000]	[-100, 100]
Wielkość populacji	50	100
Liczba generacji	500	5000

Poniżej zostały przedstawione średnie wyniki oraz odchylenie standardowe dla 10 testów. Można zauważyć, że wyniki dla obydwu sposobów selekcji są bardzo porównywalne. Dodatkowo na poniższych wykresach zostały

		Funkcja kwadratowa	Funkcja Rastrigina
Selekcja turniejowa	mean	0.0014	0.3822
	std	0.0013	0.4841
Selekcja ruletkowa	mean	0.0039	0.47061
	std	0.0037	0.3083

przedstawione pojedyncze testy, które pokazują jak zmienia się najlepszy wyniki danego osobnika (wartość funkcji celu). Wszystkie funkcje zaczynały z populacją losowaną dla tego ustawionego tego samego ziarna.



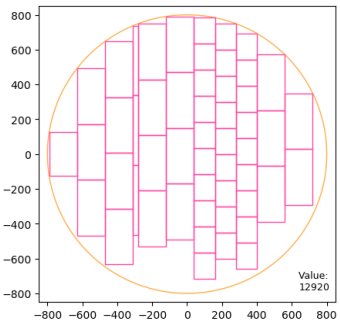
3 Cutting stock problem

Problem, którym zajmowaliśmy się w drugiej części to *cutting stock problem*. W naszym przypadku polegał o na tym, że mieliśmy do dyspozycji listę prostokątów o znanych wymiarach i ich wartości. Następnie te prostokąty miały zostać umieszczone w okręgu o ustalonym promieniu. Założeniami było to, że wszystkie rogi prostokątów muszą być w okręgu, mamy nieograniczoną liczbę każdego z prostokątów oraz prostokąty nie mogły się pokrywać.

3.1 Koncepcja rozwiązania problemu

Moja koncepcja opierała się na podziale koła na pionowe pasy, które wypełnione są jednym typem prostokąta.

Szczegółowy opis właściwości algorytmu:



- **Osobnik** — okrąg napakowany prostokątami (okrąg jest podzielony na paski (pionowe) o szerokości kolejnych prostokątów, w każdym znajduje się maksymalna liczba prostokątów, które się mieszczą w okręgu)
- **Populacja** — zbiór okręgów napakowanych prostokątami
- **Genotyp** — lista prostokątów
- **Gen** — pojedynczy prostokąt z listy
- **Funkcja przystosowania** — suma iloczynu wartości prostokątów (mieszczących się w okręgu) z listy i ich ilości w danym pasku
- **Krzyżowanie** — losujemy liczbę n z przedziału (0, 2R), a następnie dziecku dopisujemy geny rodzica1 aż suma szerokości prostokątów nie przekroczy n, a potem to samo robimy dla rodzica2 z tą różnicą, że dopiero gdy osiągniemy sumę szerokości równą n to zaczynamy dopisywać geny rodzica2 (w ten sposób otrzymujemy sumę szerokości prostokątów dziecka większą od 2R)
- **Mutacja** — losowy wybór prostokąta (genu) z listy (genotypu) i zamiana go na inny wylosowany prostokąt

3.2 Doświadczenia

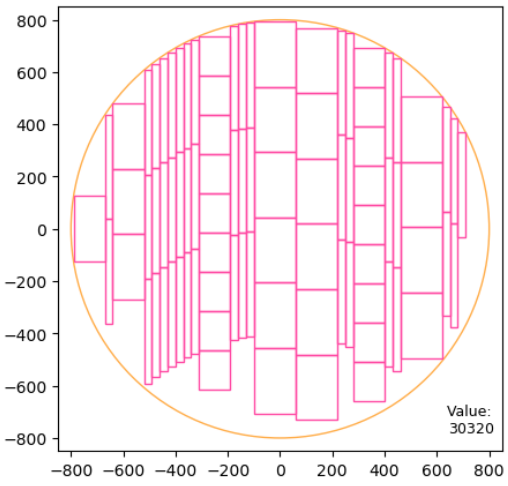
Aby przetestować działanie algorytmu zostały przeprowadzone następujące testy:

- Okrąg o promieniu: *800*, oczekiwany wynik: *30000*
- Okrąg o promieniu: *850*, oczekiwany wynik: *brak*
- Okrąg o promieniu: *1000*, oczekiwany wynik: *17500*
- Okrąg o promieniu: *1100*, oczekiwany wynik: *25500*
- Okrąg o promieniu: *1200*, oczekiwany wynik: *30000*

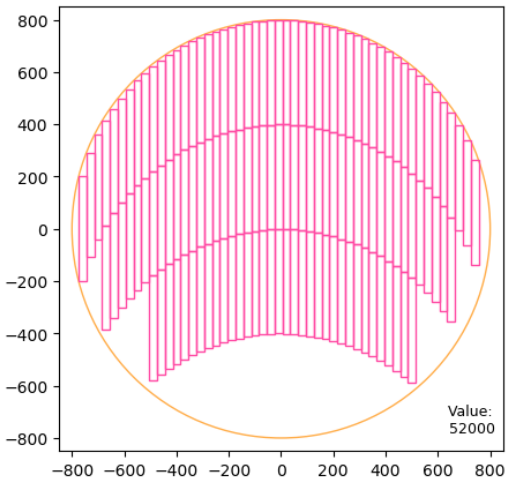
Dla powyżej opisanego algorytmu założone wartości dla poszczególnych okręgów nie są zbyt wymagające, więc już losowa inicjalizacja jest w stanie spełnić warunki, natomiast z drugiej strony gdy algorytm działa przez zbyt dużo generacji to popada w skrajność i używa tylko najbardziej opłacalnych prostokątów. Takie sytuacje również zostaną przedstawione na poniższych grafikach. Z opisanych względów pomimo ustawienia pewnej liczby generacji, program zatrzymuje się przy osiągnięciu oczekiwanego wyniku.

3.3 Wyniki

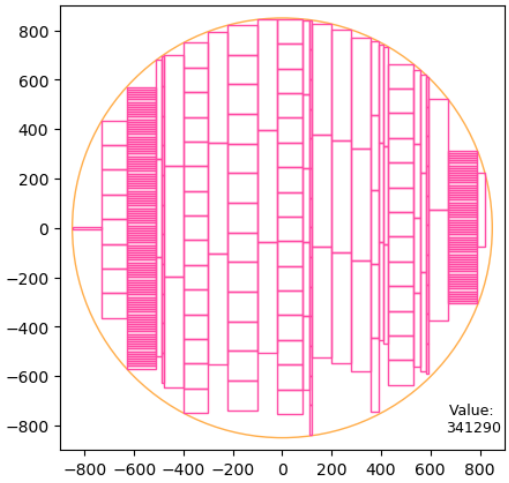
Dla każdego z wyżej opisanych eksperymentów wykonałam podstawowy test tak aby otrzymać zakładany wynik oraz dodatkowy test, który zmaksymalizuje wynik. Wartość funkcji celu jest w prawym rogu wykresów.



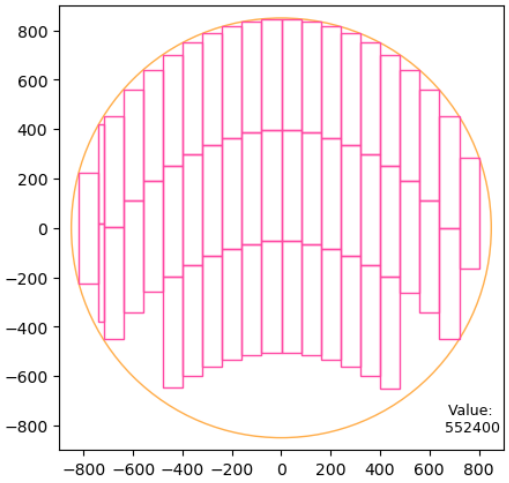
(a) podstawowy test - 800



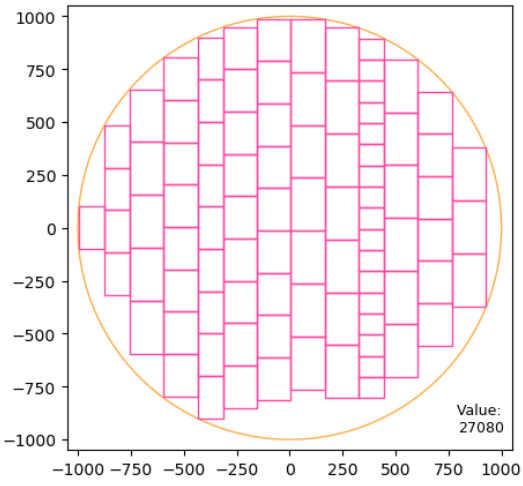
(b) maksymalizacja - 800



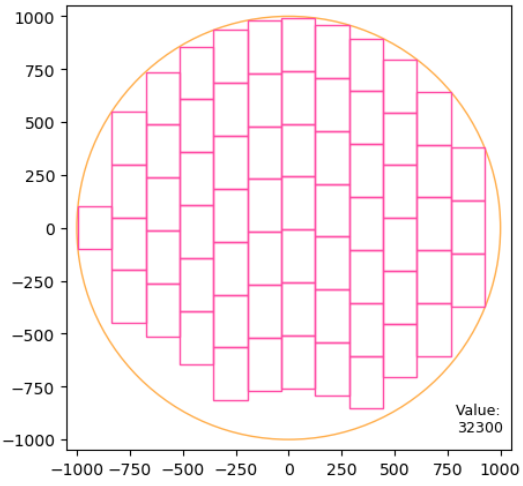
(c) podstawowy test - 850



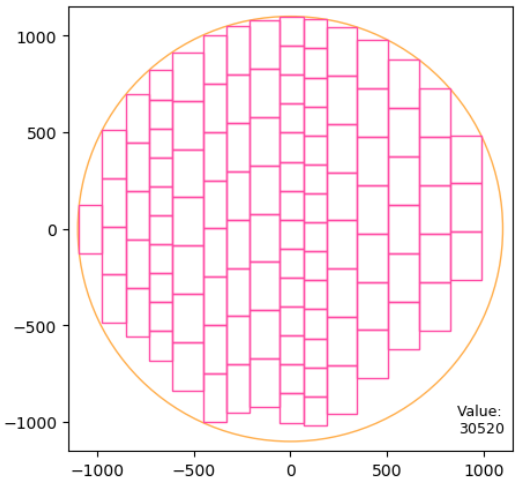
(d) maksymalizacja - 850



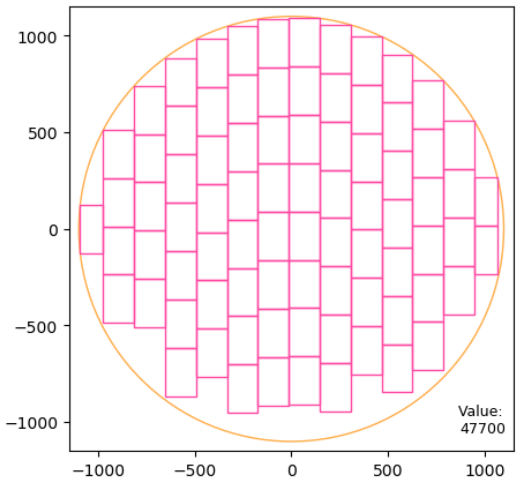
(e) podstawowy test - 1000



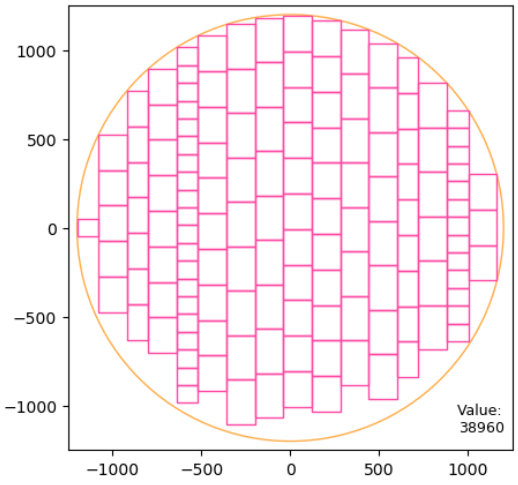
(f) maksymalizacja - 1000



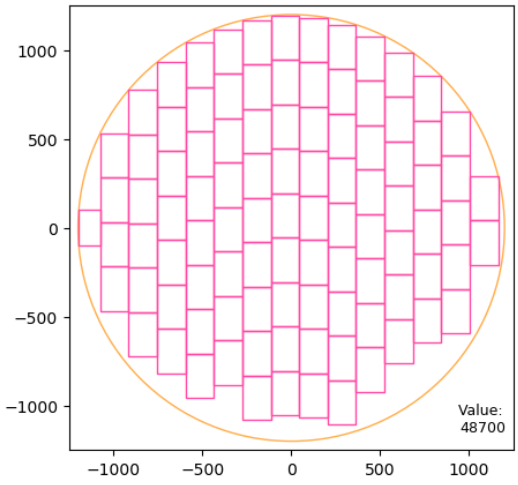
(a) podstawowy test - 1100



(b) maksymalizacja - 1100



(c) podstawowy test - 1200



(d) maksymalizacja - 1200

Wyniki świadczą o bardzo dobrym działaniu algorytmu - spełnia on swoje zadanie jakim była maksymalizacja wartości prostokątów w okręgu. Wszystkie algorytmu potrzebowały poniżej 400 generacji przy populacji o wielkości 100 osobników aby otrzymać maksymalne wartości. Można by też usprawnić algorytm umożliwiając mu obracać prostokąty o 90°, jednak obecny algorytm jest wystarczający.

4 Optymalizacja wag w sieci MLP

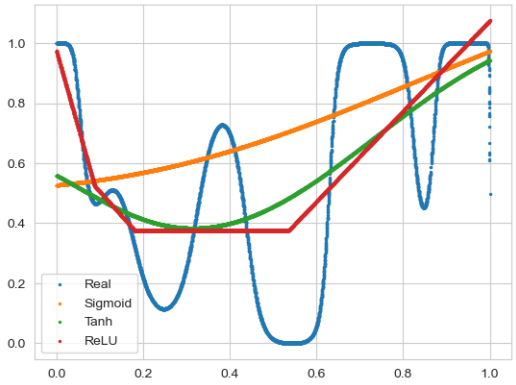
Zwieńczeniem przedmiotu jak i projektu było zadanie optymalizacji wag w sieci MLP za pomocą algorytmu genetycznego. Teraz osobnikami są sieci neuronowe z macierzami wag. Algorytm genetyczny korzysta z metod takich jak:

- selekcja - korzysta z wcześniej opisanej selekcji ruletkowej
- krzyżowanie - polega na wyborze numeru wiersza z macierzy [wagi—bajasy] a następnie górna część pierwszego rodzica do wybranego rzędu idzie do dziecka i dolna część od wybranego rzędu idzie do dziecka (z dwóch rodziców zawsze powstaje jedno dziecko)
- mutacja - tutaj używałam 3 różnych typów mutacji gaussowskiej, które różniły się liczbą współczynników, które mutowały (1 dla całej macierzy [wagi—bajasy], po jednym dla każdej warstwy i po 5 dla każdej warstwy)

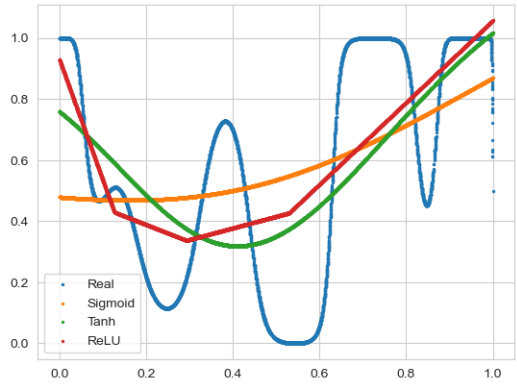
Implementację naszego algorytmu mieliśmy przetestować na 3 zbiorach danych: *multimodal_large* (zadanie regresji z poprzedniego projektu) oraz *iris* i *auto-mpg*, które mieliśmy okazje poznać na innych przedmiotach. Dla każdego z tych zbiorów przeprowadziłam testy dotyczące wyboru funkcji aktywacji sieci MLP, wartości współczynników mutacji i krzyżowania oraz wielkości populacji i liczby generacji.

4.1 Multimodal_large

Dla tego zestawu danych skupiłam się przede wszystkim na porównaniu funkcji aktywacji. Przeprowadziłam doświadczenia dla następujących funkcji aktywacji: `Sigmoid()`, `Tanh()` i `ReLU`. Dla każdej z nich przeprowadziłam doświadczenia dla wielkości populacji równej 100 i wielkości sieci - [5,1] oraz wielkości populacji równej 500 i wielkości sieci równej - [8,1], natomiast pozostałe parametry były takie same: liczba generacji - 1000, współczynnik krzyżowania - 0.7 i współczynnik mutacji - 0.2. Poniżej zostały przedstawione otrzymane wyniki regresji.



(a) populacja = 100, sieć = [5,1]



(b) populacja = 500, sieć = [8,1]

Zdecydowanie widzimy, że pomimo znormalizowanego zbioru danych wyniki jakie osiągamy są nieporównywalnie słabe do tradycyjnego uczenia sieci MLP. Jednak widać, że funkcja `Tanh()` radzi sobie najlepiej starając się dogiąć do naszych danych.

4.2 Iris

Dla zbioru danych *iris* również porównałam ze sobą różne funkcje aktywacji a dodatkowo porównałam różne wersje sposobu mutacji, które opisałam wyżej. Poniżej znajduje się zestawienie wyników, czyli wartości f-score dla różnych architektur.

Funkcje aktywacji	Liczba zmutowanych parametrów		
	1 w całej sieci	po 1 na każdą warstwę	po 5 na każdą warstwę
Sigmoid()	0.5669	0.5608	0.6073
Tanh()	0.5863	0.6209	0.6070
ReLU()	0.5828	0.6406	0.6069

Widzimy, że najbardziej optymalnym rozwiązaniem jest mutacja jednej wartości dla każdej warstwy. Dalej postanowiłam zastosować algorytm genetyczny do wyboru najlepszych hiperparametrów. Dla tej samej sieci z takimi samymi parametrami początkowymi wykonałam testy, które wybiorą optymalną wartość współczynnika krzyżowania i mutacji. Poniżej przedstawione zostały wyniki.

		Współczynnik krzyżowania		
		0.2	0.5	0.8
Współczynnik mutacji	0.2	0.5543	0.5922	0.5845
	0.5	0.5601	0.5645	0.5749
	0.8	0.6353	0.6167	0.5623

Widzimy, że algorytm zdecydowanie bardziej polega na mutacji niż na krzyżowaniu. Najlepszym wynikiem jaki udało mi się uzyskać podczas testów było accuracy na poziomie 70%.

4.3 Auto-mpg

Ostatnie testy na zbiorze *auto-mpg* zaczęłam od wyboru najlepszych parametrów. Ze względu na to, że we wcześniejszym doświadczeniu funkcja `ReLU()` okazała się być najbardziej skuteczna to teraz tylko z niej korzystałam. Chciałam określić jaka kombinacja wielkości sieci i wielkości populacji będzie najlepszą. Dokonałam testy dla 1000 generacji i poniżej zostały przedstawione wyniki, czyli wartości MSE (dla danych znormalizowanych).

		wielkość populacji		
		100	500	800
rozmiar sieci	[5,1]	427.5237	398.2113	407.9934
	[8,1]	462.3640	418.5590	393.1682
	[2,2,1]	177.3541	127.8235	122.2827
	[30,1]	488.2308	483.7653	474.9770

Widać, że sieć z 3 warstwami jest zdecydowanie najlepsza i im większa populacja tym lepsze rezultaty. Wykonałam ostatni test dla najlepszych parametrów - wielkość populacji podniosłam do 1000 a sieć jest wielkości [2, 2, 1] i algorytm działał przez 5000 generacji. Wynik MSE jaki uzyskałam był równy 71.7779.

5 Podsumowanie

Dzięki projektowi mieliśmy okazję zaimplementować działające narzędzie jakim jest algorytm genetyczny. Bardzo dobrze sprawdził się do rozwiązania zadania*cutting stock problem*, jednak optymalizacja wag sieci MLP nie była już taka wydajna. Jego standardowym zastosowaniem przy sieciach MLP na pewno będzie dostrajanie hiperparametrów.