

# GA & DNN

Jan Sołtysik

21 kwietnia 2020

## Spis treści

<b>1</b>	<b>Wstęp</b>	<b>2</b>
<b>2</b>	<b>Wprowadzenie teoretyczne</b>	<b>2</b>
2.1	Algorytm Genetyczny . . . . .	2
2.1.1	Podstawowa reprezentacja . . . . .	2
2.1.2	Selekcja . . . . .	3
2.1.3	Krzyżowanie . . . . .	4
2.1.4	Mutacja . . . . .	5
2.2	Sztuczne Sieci Neuronowe . . . . .	5
2.2.1	Perceptron . . . . .	6
2.2.2	Wielowarstwową Sieć Neuronową . . . . .	7
2.2.3	Funkcje aktywacyjne . . . . .	8
2.2.4	Inne rodzaje warstw SSN. . . . .	10
<b>3</b>	<b>Micro - GA dla SSN</b>	<b>13</b>
3.1	Reprezentacja SSN w AG . . . . .	14
3.2	Funkcja oceny dla SSN . . . . .	16
3.3	Selekcja . . . . .	16
3.4	Krzyżowanie . . . . .	17
3.5	Mutacja . . . . .	17
3.6	Warunek końca . . . . .	18
<b>4</b>	<b>Implementacja</b>	<b>18</b>
<b>5</b>	<b>Otrzymane wyniki</b>	<b>18</b>
<b>6</b>	<b>Bibliografia</b>	<b>18</b>

# 1 Wstęp

## 2 Wprowadzenie teoretyczne

### 2.1 Algorytm Genetyczny

Algorytmy genetyczne są to algorytmy poszukiwania zainspirowane mechanizmem doboru naturalnego oraz dziedziczności. Łączą w sobie zasadę przeżycia najlepiej przystosowanych osobników z systematyczną, choć zrandomizowaną wymianą informacji wprowadzoną przez operatory genetyczne takie jak krzyżowanie i mutacja, które również zainspirowane są przyrodą. W każdym pokoleniu powstaje nowy zespół sztucznych organizmów, utworzonych z połączenia fragmentów najlepiej przystosowanych przedstawicieli poprzedniego pokolenia. Oprócz tego sporadycznie wyporóbowuje się nową część składową. Element losowości nie oznacza że algorytmy genetyczne sprowadzają się do zwykłego błędzenia przypadkowego. Dzięki wykorzystaniu przeszłych doświadczeń algorytm określa nowy obszar poszukiwań o spodziewanej podwyższonej wydajności. Algorytm genetyczny jest przykładem procedury używającej wyboru losowego jako "przewodnika" w wysoce ukierunkowanym poszukiwaniu w zakodowanej przestrzeni rozwiązań [3].

Poniżej znajdują się podstawowy algorytm genetyczny przedstawiony w postaci pseudokodu [1]:

---

**Algorithm 1:** Podstawowy Algorytm genetyczny

---

**Input:** Funkcja oceny  $f(\mathbf{x})$

**Data:**  $\mathbf{x}_i(t)$  -  $i$ -ty osobnik z generacji nr.  $t$  (najczęściej reprezentowany jako ciąg znaków),  $n_x$  - wymiar każdego osobnika

**Output:** Wektor  $\hat{\mathbf{x}}$  dla którego  $f(\hat{\mathbf{x}})$  jest lokalnym minimum

Niech  $t = 0$  będzie licznikiem generacji. Wygeneruj  $n_x$  - wymiarową populację  $\mathcal{C}(0)$ , składającą się z  $n$  osobników.

**while** *Warunek końcowy nie jest prawdziwy* **do**

    Oblicz przystosowanie,  $f(\mathbf{x}_i(t))$  każdego osobnika  $\mathbf{x}_i(t)$  z populacji.

    W celu stworzenia potomstwa do najlepiej przystosowanych osobników zastosuj operatory genetyczne np. krzyżowanie i mutacja.

    Z nowo powstałych osobników stwórz nową populację  $\mathcal{C}(t + 1)$ .

    Przejdź do nowej generacji.  $t = t + 1$ .

**end**

---

#### 2.1.1 Podstawowa reprezentacja

Jednym z następstw inspiracji biologicznych AG jest słownictwo zaczerpnięte z genetyki którym będę się posługiwać w tej pracy. Osobniki występujące w populacji nazywane są **chromosomami** lub **genomami**. Idąc dalej w terminologii biologicznej chromosom składa się z **genów**, które mogą występować w

pewnej zadanej liczbie odmian, zwaną **allelami**, wyodrębnia się również umiejscowienie genu - **locus**. Materiał genetyczny składa się zazwyczaj z jednego lub więcej chromosomów, zespół ten nazywamy **genotypem**[3]. Poniższa tabela przedstawia używane przeze mnie odpowiedniki:

Genetyka	Algorytmy Genetyczne
chromosom/genom	osobnik w populacji
gen	składowe osobnika
allel	możliwe warianty składowych
locus	pozycja składowej
genotyp	zespół osobników

W elementarnym AG **genom** reprezentujemy za pomocą ciągów bitów, więc **genami** będą pojedyncze bity. Poniżej znajdują się przykładowy genom:

1	0	0	0	1	1	0	1	0	1
---	---	---	---	---	---	---	---	---	---

Rysunek 1: Przykładowy genom o długości 10.

Transformacja naszego problemu do opisu w postaci ciągów binarnych jest często bardzo trudne, a czasami wręcz niemożliwa. Lecz trud ten jest opłacalny ponieważ dla tak opisanych genomów mamy zdefiniowane operatory genetyczne które wykonując proste obliczenia zbliżają nasze osobniki do optymalnego rozwiązania.

### 2.1.2 Selekcja

Jest to etap algorytmu w którym oceniamy osobników przy pomocy funkcji określonej przez rozwiązywany przez nas problem, a następnie lepiej ocenione chromosomy mają większe prawdopodobieństwo aby zostały poddane operatorom genetycznym. Aby opisać ten proces musimy zdefiniować przedstawioną w **Algorytmie 1** funkcję oceny  $f$ .

**Funkcja oceny/przystosowania**  $f$  jest obliczana dla każdego osobnika i pozwala nam porównać które genomy są najlepiej przystosowane do zadania które optymalizujemy.

Najprostszym przykładem będzie zadanie znalezienia maximum funkcji  $f(x_1, \dots, x_n)$  gdzie  $n \in \mathbb{N}^+$ , wtedy funkcja przystosowania będzie równa wartości funkcji  $f$ , im większa wartość  $f$  tym osobnik jest lepiej przystosowany.

Istnieje wiele metod selekcji lecz zdecydowanie najpopularniejszą jest **metoda ruletki** w której prawdopodobieństwo  $p_i$  wybrania  $i$ -tego genomu do reprodukcji kolejnego pokolenia jest proporcjonalne do wartości funkcji przystosowania  $i$

jest równe:

$$p_i = \frac{f_i}{\sum_{j=1}^N f_j} \quad (1)$$

gdzie:

$f_i$  - wartość funkcji oceny dla  $i$ -tego genomu,  $N$  - rozmiar populacji.

### 2.1.3 Krzyżowanie

Jednym z dwóch podstawowych operacji wykonywanych w celu stworzenia potomstwa obecnej populacji jest **krzyżowanie**, które inspirowane jest rozmnażaniem płciowym w biologii [4].

W literaturze możemy znaleźć wiele rodzajów tej operacji, poniżej znajdują się najpopularniejsze odmiany:

- **Krzyżowanie jednopunktowe:**

W tej odmianie potomka tworzymy z dwóch wybranych przez selekcję rodziców a następnie losujemy liczbę naturalną  $l$  ze zbioru  $\{0, 1, \dots, n_x\}$ . Potomek wartości na pierwszych  $l+1$  pozycjach przyjmuje geny pierwszego rodzica a na pozostałych z drugiego rodzica.

1	0	0	0	1	1	0	1	0	1
1	1	1	0	0	1	1	0	1	0
1	0	0	0	1	1	1	0	1	0

Rysunek 2: Krzyżowanie jednopunktowe, gdzie  $n_x = 10$  i  $l = 5$ .

- **Krzyżowanie dwupunktowe:**

Sposób ten jest zbliżony do opisanego wyżej lecz zamiast jednego punktu losujemy dwie liczby naturalne  $l_1, l_2$  ze zbioru  $\{0, 1, \dots, n_x\}$ , gdzie  $l_1 < l_2$ . Potomek natomiast przyjmuje wartości z pierwszego rodzica poza elementami na pozycjach od  $l_1$  do  $l_2 - 1$  gdzie wstawiamy elementy z drugiego.

1	0	0	0	1	1	0	1	0	1
1	1	1	0	0	1	1	0	1	0
1	0	0	0	0	1	1	0	0	1

Rysunek 3: Krzyżowanie dwupunktowe, gdzie  $n_x = 10$ ,  $l_1 = 4$  i  $l_2 = 7$ .

Krzyżowania tego rodzaju można uogólnić na operacje gdzie losujemy  $k$ -punktów a potomek naprzemiennie pobiera wartości z rodziców.

- **Krzyżowanie równomierne:**

Potomek jest również tworzony przy pomocy dwóch rodziców gdzie każdy jego element jest wybierany losowo z pierwszego lub drugiego rodzica z jednakowym prawdopodobieństwem (lub wybranym innym stosunkiem prawdopodobieństw).

#### 2.1.4 Mutacja

Operator ten w klasycznej wersji algorytmu polega na zamianie zadanej wartości genu (0 na 1 lub 1 na 0) z bardzo małym prawdopodobieństwem  $\sigma_m$  (np. 0.2%).

1	0	0	0	1	1	0	1	0	1
1	0	0	0	1	1	1	1	0	1

Rysunek 4: Podstawowa mutacja dla genomu o  $n_x = 10$  i wylosowanej pozycji 6.

## 2.2 Sztuczne Sieci Neuronowe

Podobnie jak dla Algorytmów Genetycznych, inspiracją dla Sztucznych Sieci Neuronowych (SSN) była biologia a dokładniej budowa neuronów naturalnych znajdujących się w ciele człowieka. Sztuczne sieci neuronowe zostały wymyślone już w pierwszej połowie XX wieku, lecz dopiero w ostatnich latach dzięki rozwojowi technologii oraz dostępności do olbrzymiej ilości danych mogliśmy zacząć używać ich pełnego potencjału. SSN są przykładem techniki **uczenia**

**nadzorowanego** tzn. podczas uczenia sieci podajemy mu dane ze znanymi wynikami i poprzez porównywanie wygenerowanych wyników z rzeczywistymi nasza sieć uczy się generować poprawne wyniki.

### 2.2.1 Perceptron

**Perceptron** to najprostsza architektura SSN. Jednostki na wyjściach/wejściach nazywane są **neuronami**. Na neuronach wejściowych podawane są liczby oraz każde połączenie ma przypisaną wagę natomiast wyjście perceptronu jest obliczane przez wyliczenie ważonej sumy sygnałów wejściowych:

$$z = \sum_{i=1}^n w_i x_i = \mathbf{w}^T \mathbf{x} \quad (2)$$

gdzie:

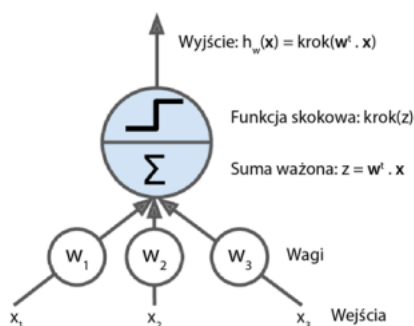
$w_i$  - waga połączenia pomiędzy  $i$ -tym neuronem wejściowym a  $j$ -tym wyjściowym,

$x_i$  -  $i$ -ta wartość wejściowa.

Następnie wynik otrzymany w (2) jest poddawany **funkcji skoku**, gdzie najczęściej jest ona równa **funkcji Heaviside'a** lub **signum** które są równe:

$$\text{Heaviside}(z) = \begin{cases} 0, & z < 0 \\ 1, & z \geq 0 \end{cases} \quad \text{sgn}(z) = \begin{cases} -1, & z < 0 \\ 0, & z = 0 \\ 1, & z > 0 \end{cases}$$

Nazywane są one **funkcjami aktywacji**.



Rysunek 5: Schemat perceptronu z jednym neuronem wyjściowym i trzema wejściowymi [2]

Lecz aby na wyjściu otrzymywać oczekiwane wyniki musimy najpierw perceptron wytrenować. Proces ten polega na modyfikacji wag na podstawie porównanie wyniku oczekiwanego z wynikiem otrzymanym. Wagi są aktualizowane

według wzoru [2]:

$$w_{ij} = w_{ij} + \eta(y_j - \hat{y}_j)x_i \quad (3)$$

gdzie:

$w_{ij}$  - waga połączenia pomiędzy  $i$ -tym neuronem wejściowym a  $j$ -tym wyjściowym,

$\hat{y}_j$  - otrzymany wynik na  $j$ -tym wyjściu,

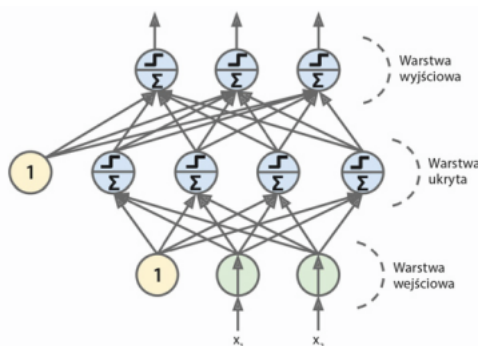
$y_j$  - docelowy wynik  $j$ -tego neuronu,

$\eta$  - współczynnik uczenia.

Perceptron jednak nie nadaje się do skomplikowanych zadań ponieważ przez swoją prostą budowę jest jedynie zdolny do klasyfikowania danych które są liniowo separowalne.

### 2.2.2 Wielowarstwowa Sieć Neuronowa

Ograniczenia perceptronu można wyeliminować tworząc SSN z wielu warstw perceptronów. SSN tego typu nazywamy **perceptronem wielowarstwowym**, który jest złożony z **warstwy wejściowej**, co najmniej jednej **warstwy ukrytej** (jako wejście przyjmują one wyjście poprzedniej warstwy a ich wyjście propagowane do kolejnej jako wejście) ostatnią warstwę nazywamy **warstwą wyjściową**. Dodatkowo w każdej warstwie znajdują się **neuron obciążający**, którego zadaniem jest wysyłanie na wejście następnej warstwy wartości 1. Ilość neuronów w warstwie wejściowej i wyjściowej jest określana przez zestaw danych na których sieć ma pracować np. jeśli zadaniem sieci jest rozpoznawanie ręcznie pisanych cyfr to na wyjściu powinno znaleźć się 10 neuronów a na wejściu każdy neuron powinien odpowiadać jednemu pikselowi wczytanego obrazu. SSN która zawiera co najmniej dwie warstwy ukryte nazywamy **głęboką siecią neuronową** (GSN) [2].



Rysunek 6: Perceptron wielowarstwowy z jedną warstwą ukrytą [2].

Zaproponowana w wzorze (3) metoda uczenia perceptronu nie zadziałała w przypadku sieci wielowarstwowej. Najpopularniejszą metodą uczenia sieci neuronowych jest **wsteczna propagacja**. Cały proces możemy przedstawić za pomocą kroków:

---

**Algorithm 2:** Procedura uczenia wielowarstwowej SSN.

---

- 1 Wybór parametrów sieci (liczba warstw, liczba neuronów w warstwach itd.)
  - 2 Wagi inicjujemy losowo.
  - 3 Dla każdego neuronu obliczany jest błąd równy różnicy pomiędzy otrzymanym wynikiem  $\hat{\mathbf{y}}$  a wartością oczekiwaną  $\mathbf{y}$ .
  - 4 Błędy propagowane są do poprzednich warstw.
  - 5 Modyfikacja wag na podstawie wartości błędu.
  - 6 Powtarzaj od **3** dla kolejnych wektorów uczących.
  - 7 Skończ algorytm jeśli przekroczymy ustaloną liczbę epok, lub średni błąd przestanie zauważalnie maleć.
- 

Średni błąd może być przedstawiony przez wzór:

$$d = \frac{1}{2}(\hat{\mathbf{y}} - \mathbf{y})^2 \quad (4)$$

Podczas aktualizowania wag wybieramy te wartości dla których średni błąd jest najmniejszy, możemy je znaleźć np. używając metodę gradientu prostego przy użyciu odwrotnego różniczkowania automatycznego.

### 2.2.3 Funkcje aktywacyjne

Aby powyższy algorytm przebiegał prawidłowo zamiast funkcji skokowej/signum wprowadzono inne funkcje aktywacji, ponieważ funkcje używane w przypadku pojedynczego perceptronu są złożone jedynie z płaskich segmentów, co uniemożliwia korzystać z gradientu. Poniżej znajdują się najczęściej używane **funkcje aktywacji** w SSN:

- **funkcja logistyczna:**

W każdym punkcie ma zdefiniowaną pochodną niezerową, dzięki czemu algorytm gradientu prostego może na każdym etapie uzyskiwać lepsze wyniki. Jest ona używana w warstwie wyjściowej jeśli zadaniem naszej sieci jest klasyfikacja obiektów na przynależność do dwóch rozłącznych klas. Dodatkowo jest to funkcja używana przez neurony naturalne przez co uważana była za najlepszą funkcję aktywacyjną, lecz jak pokazała praktyka w przypadku SSN inne funkcje sprawują się lepiej. Przyjmuje ona wartości z zakresu  $(0, 1)$ .

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (5)$$

- **funkcja tangensa hiperbolicznego:**

Jest S - kształtna, ciągła i różniczkowalna, podobnie jak funkcja logistyczna, ale zakres wartości wynosi  $(-1, 1)$  a nie  $(0, 1)$  jak w przypadku funkcji logistycznej  $\sigma$ .

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad (6)$$



- **funkcja ReLU:**

Jest ciągłą, ale nieróżniczkowalną dla  $z = 0$ . W praktyce jednak spisuje się znakomicie dodatkowo jest bardzo szybko obliczana. Aktualnie jest ona oraz jej odmiany są najczęściej używanymi funkcjami aktywacji dla warstw ukrytych i wejściowej.

$$ReLU(z) = \max(0, z) \quad (7)$$

- **funkcja softmax:**

Używana jest ona w warstwie wyjściowej jeśli nasza sieć ma obliczać prawdopodobieństwa przynależności otrzymywanych obiektów do klas (jest ich więcej niż 2 i wszystkie prawdopodobieństwa mają się sumować do 1). Model najpierw oblicza dla każdej klasy -  $k$ :

$$s_k(\mathbf{x}) = (\mathbf{w}^{(k)})^T \mathbf{x} \quad (8)$$

gdzie:  $\mathbf{w}^{(k)}$  - wyspecjalizowany wektor wag dla klasy  $k$ .

Po wyliczeniu  $s_k(\mathbf{x})$  dla każdej klasy, obliczane jest odpowiednio znormalizowane prawdopodobieństwo przynależności danej próbki do klasy  $k$ :

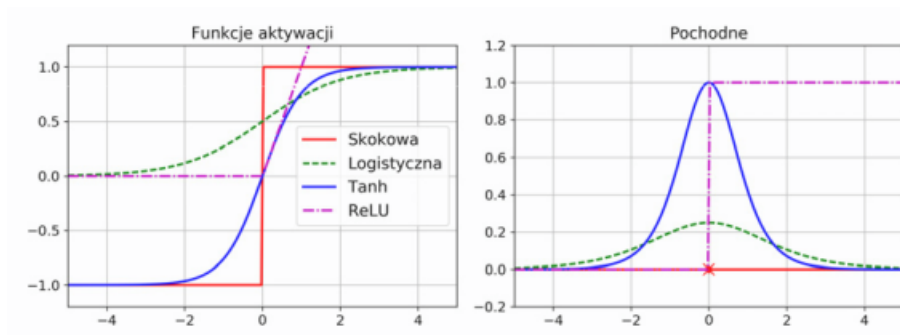
$$p_k = \frac{\exp(s_k(\mathbf{x}))}{\sum_{i=1}^K \exp(s_i(\mathbf{x}))} \quad (9)$$

gdzie:  $K$  - liczba klas.

Model prognozuje klasę o najwyższym prawdopodobieństwie:

$$\hat{y} = \underset{k}{\operatorname{argmax}}(p_k) \quad (10)$$

Gdzie funkcja **argmax** zwraca indeks klasy dla której wartość  $p_k$  jest największa.

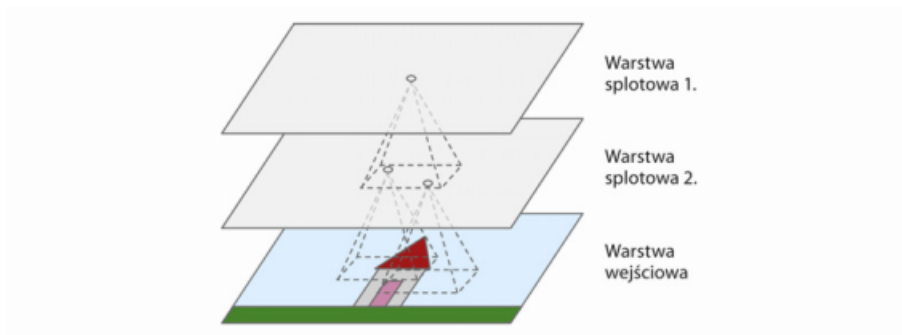


Rysunek 7: Funkcje aktywacyjne i ich pochodne. [2]

### 2.2.4 Inne rodzaje warstw SSN.

W SSN oprócz **warstw gęstych** tzn. gdzie każdy neuron w  $i$ -tej warstwie jest połączony z wszystkimi neuronami znajdującymi się w warstwie nr.  $i + 1$ , używane są warstwy innego typu które pomagają sieci osiągać lepsze wyniki. Oto kilka przykładowych warstw:

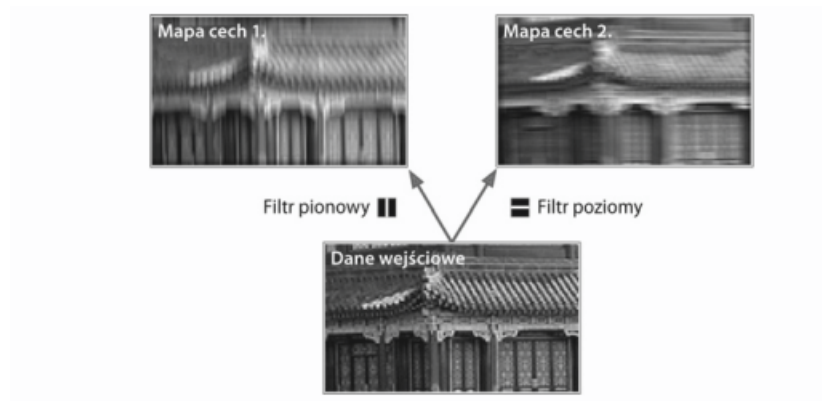
- **warstwa splotowa:** Używane są one w zadaniach wizualnych. Neurony w pierwszej warstwie splotowej nie są połączone z każdym pikselem obrazu wejściowego, lecz wyłącznie z pikselami znajdującymi się w ich polu recepcyjnym.



Rysunek 8: Warstwy CNN z prostokątnymi lokalnymi polami recepcyjnymi [2].

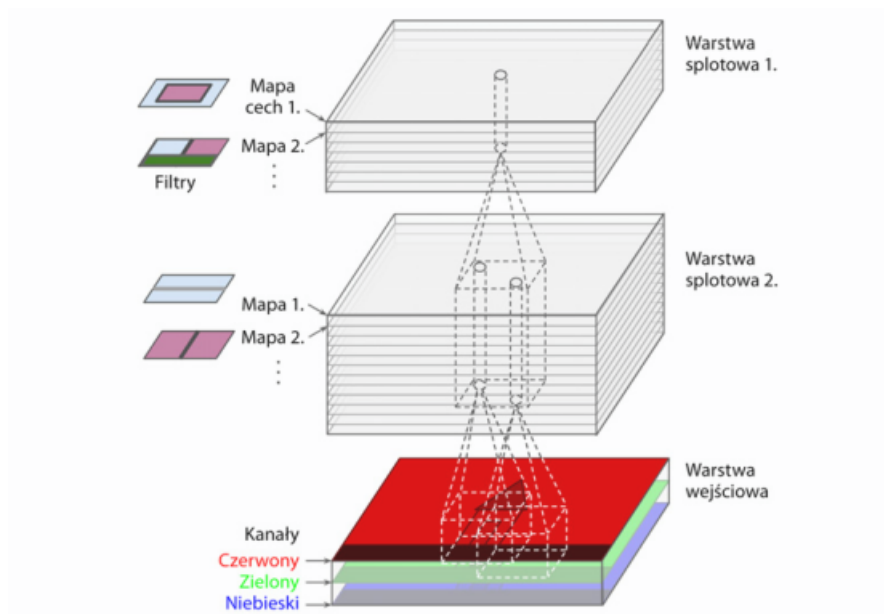
Dzięki powyższej strukturze sieć może koncentrować się na ogólnych cechach w pierwszej warstwie ukrytej, następnie łączyć je w bardziej złożone kształty w kolejnych warstwach. Ogólnie neuron znajdujący się w  $i$ -tym wierszu oraz  $j$ -tej kolumnie danej warstwy jest połączony z wyjściami neuronów poprzedniej warstwy zlokalizowanymi w rzędach od  $i \cdot s_h$  do  $i \cdot s_h + f_h - 1$  i kolumnach od  $j \cdot s_w$  do  $j \cdot s_w + f_w - 1$  gdzie:  $f_w/f_h$  - szerokość/wysokość pola recepcyjnego,  $s_h, s_w$  definiują wartość **kroków** odpowiednio w kolumnach i rzędach. **Krokiem** nazywamy odległość pomiędzy dwoma kolejnymi polami recepcyjnymi. Aby uzyskać takie same wymiary dla każdej warstwy najczęściej dodawane są zera wokół wejść.

Wagi neuronu mogą być przedstawione jako niewielki obraz o rozmiarze pola recepcyjnego, tak zwane **filtry**. Przykładowo filtrem może być macierz wypełniona zerami oprócz środkowej kolumny zawierającej jedynki, neurony posiadające taki filtr będą ignorować wszystkie elementy oprócz tych które znajdują się w środkowej linii.



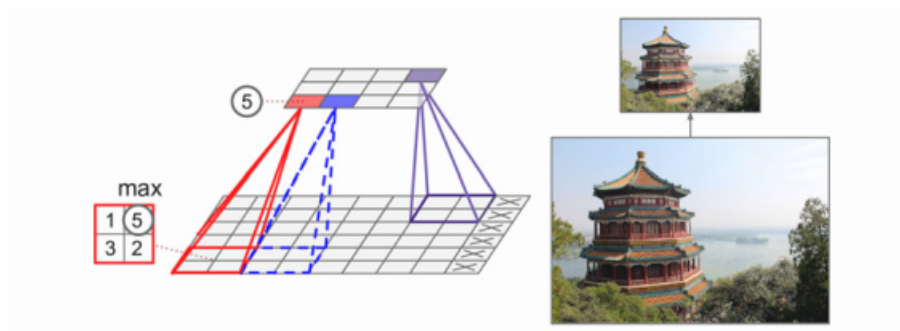
Rysunek 9: Dwie mapy cech otrzymane przy pomocy dwóch różnych filtrów [2].

Warstwa wypełniona neuronami o takim samym filtrze nazywamy **mapą cech** która pomaga nam wyszczególnić elementy przypominające użyty filtr. Warstwa splotowa składa się z kilku map cech o identycznych rozmiarach. Każda mapa ma swoje wartości parametrów, dzięki czemu stosując różne filtry warstwa splotowa jest w stanie wykryć wiele cech w dowolnym obszarze obrazu. Dodatkowo obrazy wejściowe składają się również z kilku warstw, po jednej na każdy kanał barw. Zazwyczaj są to trzy kanały - czerwony, zielony i niebieski lub jeden dla obrazów czarno-białych.



Rysunek 10: Warstwa splotowa zawierająca wiele cech, oraz zdjęcie z trzema kanałami barw [2].

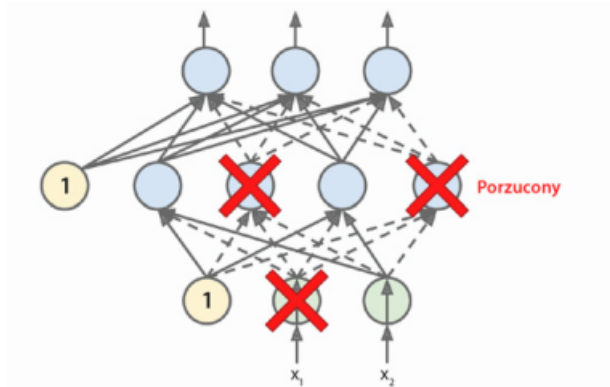
- **warstwa łącząca:** Jej celem jest zmniejszenie obrazu wejściowego w celu zredukowania obciążenia obliczeniowego, wykorzystania pamięci i liczby parametrów. Tak samo jak w warstwach splotowych neurony łączą się z wyjściami określonej liczby neuronów poprzedniej warstwy, które mieszczą się w obszarze pola recepcyjnego. Jednakże warstwa ta nie zawiera wag, jej zadaniem jest gromadzenie danych przy pomocy funkcji agregacyjnej np. maksymalizującej lub uśredniającej.



Rysunek 11: Maksymalizująca warstwa łącząca, gdzie rozmiar jądra łączącego to 2x2 [2].

- **warstwa porzucania:**

Warstwa ta dla poprzedniej warstwy aplikuje technikę **porzucania** tzn. że każdy neuron znajdujący się w tej warstwie w poszczególnym przebiegu może zostać całkowicie pominięty w procesie uczenia. Szansa na porzucenie jest hiperparametrem i nazywana jest **współczynnikiem porzucenia**.



Rysunek 12: Przykład porzucania [2].

Istnieje wiele innych rodzajów warstw w SSN (np. rekurencyjne) lecz opisałem tylko warstwy użyte w moim programie. Z tego samego powodu jedyną opisaną przeze mnie architekturą jest **jednokierunkowa sieć neuronowa** (sygnał biegnie w jednym kierunku).

### 3 Micro - GA dla SSN

SSN posiadają dużą liczbę hiperparametrów i wpływ każdego z nich na efektywność sieci jest zależny od wykonywanego zadania, więc jeżeli wybieramy je ręcznie musimy to robić metodą prób i błędów. Aby użyć AG to tego zadania musimy zmodyfikować jego tradycyjną odmianę. Ponieważ zależy nam na szybkości wykonywanych obliczeń zastosowana została odmiana **micro-GA** gdzie liczba dozwolonych generacji jest niewielka. Dodatkowo aby przyspieszyć działanie algorytmu SSN są trenowane przez ograniczoną liczbę epok tzw. **uczenie częściowe**.

---

**Algorithm 3:** Zmodyfikowany Algorytm Genetyczny wybierający hiperparametry SSN.

---

**Data:** Treningowy i walidacyjny zbiór danych  $\mathcal{D}_t, \mathcal{D}_v$   
**Input:** Hiperparametry algorytmu.  
**Output:** Najlepiej przystosowana SSN  $\phi^*$   
Niech  $t_e = 0$  będzie licznikiem wykonanych przebiegów AG.  
Niech  $\mathcal{S} = \{\}$  zbiór na najlepsze rozwiązania.  
**while**  $t_e < \gamma_r$  **do**  
    Niech  $t = 0$  będzie licznikiem generacji.  
    Stwórz i losowo zainicjalizuj początkową populację  $\mathcal{C}(0)$ , zawierającą  $n$  osobników, gdzie  $n \leq 10$ .  
    **while**  $t < \gamma_g \vee$  *warunek wczesnego końca dla  $\mathcal{C}(t)$  nie jest spełniony* **do**  
        Oblicz funkcję przystosowania  $f(\phi)$  dla każdego osobnika w populacji.  
        Zastąp najgorsze modele w  $\mathcal{C}(t)$  najlepszymi modelami  $\mathcal{C}(t-1)$ .  
        Wykonaj selekcję tworząc populację potomstwa  $\mathcal{O}(t)$ .  
        Wykonaj operację krzyżowania i mutacji na osobnikach z  $\mathcal{O}(t)$ .  
         $\mathcal{C}(t+1) = \mathcal{O}(t)$ .  
         $t = t + 1$ .  
    **end**  
    Do  $\mathcal{S}$  dodaj najlepszego osobnika z poprzedniego przebiegu.  
     $t_e = t_e + 1$ .  
**end**  
Znormalizuj koszt dla każdego modelu w  $\mathcal{S}$ .  
Najlepszym modelem jest osobnik z  $\mathcal{S}$  dla którego funkcja przystosowania przyjmuje najniższą wartość.

---

### 3.1 Reprezentacja SSN w AG

Z powodu złożoności SSN musiałem zrezygnować z standardowej reprezentacji osobników jako ciągi binarne. Użyty przeze mnie modelem jest kodowanie oparte na liście tablic tzn. każda warstwa sieci jest reprezentowana jako tablica w której każde pole ma z góry określone znaczenie. W tej reprezentacji warstwy przyjmują postać:

Typ Warstwy	Liczba Neuro-nów	Funkcja Aktywacji	Liczba Filtrów	Rozmiar Pola Receptyjnego	Krok Pola Receptyjnego	Rozmiar Pola Łączącego	Współczynnik porzucenia

Gdzie każde pole może przyjmować pewien zakres wartości:

Nazwa pola	Używane Przez	Wartości
Typ warstwy	Perceptron wielowarstwowy/Sieć splotowa	$x \in \{1, \dots, 4\}$
Liczba neuronów	Perceptron wielowarstwowy	$8 * x$ dla $x \in \{1, \dots, 128\}$
Funkcja aktywacji	Perceptron wielowarstwowy/Sieć splotowa	$x \in \{1, \dots, 4\}$
Liczba filtrów	Sieć splotowa	$8 * x$ dla $x \in \{1, \dots, 64\}$
Rozmiar pola recepcyjnego	Sieć splotowa	$3^x$ dla $x \in \{1, \dots, 6\}$
Krok pola recepcyjnego	Sieć splotowa	$x \in \{1, \dots, 6\}$
Rozmiar pola łączącego	Sieć splotowa	$2^x$ dla $x \in \{1, \dots, 6\}$
Współczynnik porzucenia	Perceptron wielowarstwowy/Sieć splotowa	$x \in [0, 1]$

Osobniki więc będą listą takich tablic. Warstwa wejściowa i wyjściowa jest ustalona przez zadanie jakie ma wykonywać sieć np. jeśli trenowana sieć ma rozpoznawać ręcznie pisane cyfry wiemy że w warstwie wyjściowej musi pojawić się 10 neuronów z funkcją aktywacyjną typu softmax. Generując jednak pozostałe warstwy musimy pamiętać że warstwy nie mogą być ułożone losowo (np. nie mogą występować dwie warstwy porzucania po sobie). Poniżej znajdują się odpowiednie mapowania typów warstw i funkcji aktywacji na liczby całkowite oraz odpowiednie zasady łączenia warstw w budowanych osobnikach.

Typ warstwy	Nazwa	Dozwoleni następcy	Typ funkcji	Nazwa
1	Warstwa Gęsta	1, 4	1	Sigmoid
2	Splotowa	1, 2, 3, 4	2	Tangens hiperboliczny
3	Łącząca	1, 2	3	ReLU
4	Porzucenia	1, 2	4	Softmax
			5	Liniiowa

Dodanie innych typów warstw (np. rekurencyjnych) i funkcji nie powinno być problemem, jedynie trzeba pamiętać o dodaniu odpowiednich zasad budowania dla nich. Przykładowy genomem reprezentującym SSN jest:

[[1, 784, 3, 0, 0, 0, 0, 0], [5, 0, 0, 0, 0, 0, 0, 0.54], [1, 300, 2, 0, 0, 0, 0, 0]  
[5, 0, 0, 0, 0, 0, 0, 0.28], [1, 100, 3, 0, 0, 0, 0, 0], [1, 10, 4, 0, 0, 0, 0, 0]]

Dekodując powyższy genom otrzymujemy sieć złożoną z następujących warstw:

Typ warstwy	Liczba Neuronów	Funkcja Aktywacji	Współczynnik Porzucenia
Warstwa Gęsta	784	ReLU	N/A
Warstwa porzucania	N/A	N/A	0.54
Warstwa Gęsta	300	ReLU	N/A
Warstwa porzucania	N/A	N/A	0.28
Warstwa Gęsta	100	ReLU	N/A
Warstwa Gęsta	10	Softmax	N/A

Generując genomy w populacji warstwa pierwsza wyznacza typ SSN(np. jeśli pierwszą warstwą jest warstwa gęsta to sieć będzie perceptronem wielowarstwowym, a jeśli pierwszą warstwą będzie warstwa splotowa sieć będzie siecią splotową). Po każdym wygenerowaniu warstwy ukrytej obliczamy losową liczbę  $\rho_l$  według danego rozkładu prawdopodobieństwa:

$$\rho_l = 1 - \sqrt{1 - U} \quad (11)$$

gdzie:  $U$  - liczba wylosowana przy pomocy płaskiego rozkładu prawdopodobieństwa.

Wtedy jeśli  $\rho_l < \gamma_l$  i wygenerowana warstwa jest zgodna z zasadami przedstawionymi w tabeli (**odniesienie do tabeli**) dodajemy warstwę do genomu. Gdzie  $\gamma_l$  - zdefiniowane przez użytkownika prawdopodobieństwo dodawania nowej warstwy. Dodawanie kończymy jeśli przekroczymy maksymalną liczbę warstw lub  $\rho_l \geq \gamma_l$ .

### 3.2 Funkcja oceny dla SSN

Celem algorytmu jest znalezienie sieci  $\phi^*$  osiągającej jak najlepszy wynik jednocześnie ograniczając liczbę trenowalnych parametrów, więc jest to problem optymalizacyjny przedstawiony za pomocą wzoru:

$$\min_{\phi} (p(\phi), w(\phi)) \quad (12)$$

gdzie:

$p(\phi)$  - miara jakości sieci (np. precyzja predykcji),

$w(\phi)$  - liczba trenowalnych parametrów zwana **rozmiarem sieci**.

Jeżeli jednak chcielibyśmy znaleźć bezpośrednio rozwiązanie powyższego równania prowadziłoby to do zadania optymalizacji wieloobiektywnej. Aby uprościć to zadanie definiujemy funkcję oceny, względem której osobniki w populacji będą porównywane:

$$g(\phi) = (1 - \alpha)p(\phi) + \alpha w(\phi) \quad (13)$$

Jednak powinniśmy jeszcze przeskalować  $p(\phi)$ ,  $w(\phi)$  aby ich wartości były podobnego rzędu. Niech  $\mathbf{p} = \{p(\phi_1), \dots, p(\phi_n)\}$  będzie wektorem ocen aktualnej populacji, wtedy po znormalizowaniu  $\mathbf{p}^* = \frac{\mathbf{p}}{\|\mathbf{p}\|}$  zakres wartości ocen sieci będzie z przedziału  $p^*(\phi_i) \in [0, 1]$  dla każdego  $i$ . Niech  $\mathcal{A}$  oznacza maksymalną liczbę warstw w sieci, oraz  $\mathcal{N}$  oznacza maksymalną liczbę neuronów w warstwie wtedy maksymalna wartość rozmiaru sieci jest równa  $\mathcal{W}$ . **skonczyć**.

Odpowiednio przeskalowana funkcja oceny wyrażona jest wzorem:

$$g(\phi) = 10(1 - \alpha)p^* + \alpha w^*(\phi) \quad (14)$$

### 3.3 Selekcja

Aby wygenerować  $n$  potomków potrzebujemy  $2n$  rodziców wybranych z bieżącej populacji. Do tego celu wykonamy odmianę **binarnej selekcji turniejowej**:

---

**Algorithm 4:** Binarna selekcja turniejowa dla SSN.

---

**Input:** Aktualna generacja  $\mathcal{C}(t)$ .

**Output:** Najlepiej przystosowani osobnicy z populacji.

Niech  $\mathcal{P}$  - zbiór rodziców.

Wybierz losowo  $m$  rodziców gdzie  $m < n$ .

**while** nie wybrano  $2n$  rodziców **do**

    Porównaj parami wybrane genomy, lepiej przystosowany dodaj do  $\mathcal{P}$ .

---



W powyższej procedurze im większa wartość  $m$  tym większe jest prawdopodobieństwo wybrania najlepszych osobników jako rodzica.

### 3.4 Krzyżowanie

Ponieważ standardowe operatory genetyczne nie będą działać dla zmodyfikowanego kodowania genomu, je również trzeba zmodyfikować. Wybrany krzyżowaniem jest zmodyfikowane krzyżowanie dwupunktowe. Wykonując krzyżowanie musimy pamiętać, że nie każda warstwa może występować po każdej innej. Przykładowa dla dwóch rodziców  $S_1, S_2$  po wybraniu punktów  $(r_1, r_2)$  z  $S_1$  i  $(r_3, r_4)$  z  $S_4$  (gdzie  $r_i$  oznacza indeks warstwy), może się stać że warstwa  $r_3$  nie może być zamieniona z warstwą  $r_1$  lub  $r_4$  z  $r_2$  z powodu niekompatybilności z warstwą poprzednią **odniesienie do tabeli**.

---

**Algorithm 5:** Krzyżowanie dwupunktowe dla SSN.

---

**Input:** Dwa genomy  $S_1, S_2$ .

**Output:** Nowy potomek  $S_3$ .

Losowo wybierz dwa punkty z  $S_1$  gdzie  $r_1 \leq r_2$ .

**if**  $r_1 = r_2$  **then**

$r_2 = \text{len}(S_1) - 1$

Znajdź wszystkie pary punktów  $(r_3, r_4)_i$  w  $S_2$  kompatybilne z  $(r_1, r_2)$ ,  
gdzie  $r_3 < r_4$  i  $(r_4 - r_3) - (r_2 - r_1) < \mathcal{A}$ .

Losowo wybierz jedną z par  $(r_3, r_4)_i$ .

W  $S_1$  zastąp warstwy z przedziału  $[r_1, r_2]$  warstwami z przedziału  
 $[r_3, r_4]$  wzięte z  $S_2$ . Nowy genom oznacz jako  $S_3$ .

Skoryguj funkcję aktywacyjną  $S_3$  aby były zgodne z funkcjami  $S_1$ .

---

[DODAJ PRZYKŁAD].

### 3.5 Mutacja

Mimo że mutacja nie jest potrzebna w mikro-AG, została ona dodana aby jeszcze bardziej zróżnicować modele i potencjalnie zwiększyć szanse na otrzymanie lepszych rozwiązań. Zmodyfikowana operacja mutacji jest bardzo podobna do swojej klasycznej wersji. Każdy model z populacji potomków może podlegać mutacji przy prawdopodobieństwie  $\rho_m$ , wtedy:

- Losowo wybierz jedną z warstw, a następnie zmień jeden z jej używanych parametrów zgodnie z tabelą **odniesienie do niej**.
- Zmień funkcję aktywacji, a następnie skoryguj funkcję w pozostałych warstwach oprócz ostatniej.
- Dodaj warstwę porzucenia jeśli jest ona kompatybilna z zmienianą warstwą.

### 3.6 Warunek końca

W Algorytmie Genetycznym może dojść do sytuacji w której zmiany w kolejnych pokoleniach są niezauważalne, wtedy bez konsekwencji można zakończyć aktualny przebieg algorytmu. Możemy to zaimplementować definiując odpowiedni warunek końca. Warunek ten można oprzeć na funkcji oceny wiemy jednak że sieci o podobnej strukturze będą miały zbliżone wartości dla tej funkcji. Dlatego warunek końca możemy oprzeć definiując **odległość** między genomami mówiąca nam jak zbliżone są do siebie dwie SSN.

Niech  $S^{(i)}$  - oznacza  $i$ -tą warstwę sieci  $S$ . Wtedy przy założeniu  $\text{len}(S_2) \geq \text{len}(S_1)$  odległość  $d(S_1, S_2)$  między dwoma genomami reprezentującymi SSN możemy obliczyć według algorytmu:

---

**Algorithm 6:** Odległość między dwoma SSN.

---

**Input:** Dwa genomy  $S_1, S_2$ .

**Output:** Odległość między  $S_1$  a  $S_2$ .

Niech  $d = 0$  - odległość między  $S_1$  a  $S_2$ .

**for** Dla każdego  $i$ , gdzie  $1 \leq i \leq \text{len}(S_1)$  **do**

$d = d + \|S_2^{(i)} - S_1^{(i)}\|$ .

**for** Dla każdego  $i$ , gdzie  $\text{len}(S_1) < i \leq \text{len}(S_2)$  **do**

$d = d + \|S_2^{(i)}\|$ .

---

Dla powyższej definicji  $d$  jeśli  $S_1 = S_2$  to  $d(S_1, S_2) = 0$ , więc pozwala nam ona określić stopień podobieństwa dwóch osobników.

Aktualny przebieg AG kończymy więc gdy co najmniej  $m_c$  par spełnia  $d(S_1, S_2) \leq d_t$ , gdzie zarówno  $d_t$  i  $m_c$  są hiperparametrami AG.

## 4 Implementacja

## 5 Otrzymane wyniki

## 6 Bibliografia

### References

- [1] Oliver Schütze David Laredo Yulin Qin and Jian-Qiao Sun. “Automatic Model Selection for Neural Networks”. In: (2019).
- [2] Aurélien Géron. *Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow*. 2018.
- [3] David E. Goldberg. *Algorytmy genetyczne i ich zastosowania*. 1989.
- [4] Wikipedia. *Crossover (genetic algorithm)*. [https://en.wikipedia.org/wiki/Crossover\\_\(genetic\\_algorithm\)](https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)).