

GA & DNN

Jan Sołtysik

23 marca 2020

Spis treści

1	Wstęp	2
2	Wprowadzenie teoretyczne	2
2.1	Algorytm Genetyczny	2
2.1.1	Podstawowa reprezentacja	2
2.1.2	Selekcja	3
2.1.3	Krzyżowanie	3
2.1.4	Mutacja	4
2.2	Sztuczne Sieci Neuronowe	5
2.2.1	Perceptron	5
2.2.2	Wielowarstwową Sieć Neuronową	6
2.2.3	Funkcje aktywacyjne	7
2.2.4	Inne rodzaje warstw SSN.	9
3	Micro - GA dla SSN	12
3.1	Reprezentacja SSN w AG	13
3.2	Funkcja kosztu i selekcja	15
3.3	Krzyżowanie	15
3.4	Mutacja	15
3.5	Warunek końca	15
4	Implementacja	15
5	Otrzymane wyniki	15
6	Bibliografia	15

1 Wstęp

2 Wprowadzenie teoretyczne

2.1 Algorytm Genetyczny

Algorytmy genetyczne są to algorytmy poszukiwania zainspirowane mechanizmem doboru naturalnego oraz dziedziczności. Łączą w sobie ewolucyjną zasadę przeżycia najlepiej przystosowanych osobników z systematyczną, choć zrandomizowaną wymianą informacji. 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 wysocze ukierunkowanym poszukiwaniu w zakodowanej przestrzeni rozwiązań [2].

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

Algorithm 1: Podstawowy Algorytm genetyczny

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

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.

 Zastosuj operatory genetyczne aby stworzyć potomstwo.

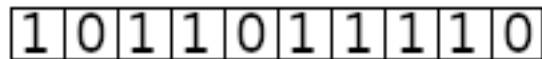
 Wybierz nową populację. $\mathcal{C}(t + 1)$.

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

end

2.1.1 Podstawowa reprezentacja

W elementarnym AG osobniki nazywane **genomami** reprezentujemy za pomocą ciągów bitów. Poniżej znajduje się przykładowy genom:



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

Rysunek 1: Przykładowy genom o długości 10.

Transformacja naszego problemu do opisu w postaci ciągów binarnych jest często bardzo trudne, a czasami wręcz niemożliwe. Lecz trud ten jest opłacalny ponieważ dla tak opisanych genomów mamy zdefiniowane operatory genetyczne których poprawność jest dowiedziona matematycznie. Natomiast w przypadku użycia innej reprezentacji osobników musimy zdefiniować własne operatory oraz nie mamy pewności co do ich efektywności.

2.1.2 Selekcja

Jest to etap algorytmu w którym wybieramy poszczególne genomy które później zostaną 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 $g(x_1, \dots, x_n)$ gdzie $n \in \mathbb{N}^+$, wtedy funkcja przystosowania będzie równa wartości funkcji g , im większa wartość g 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.

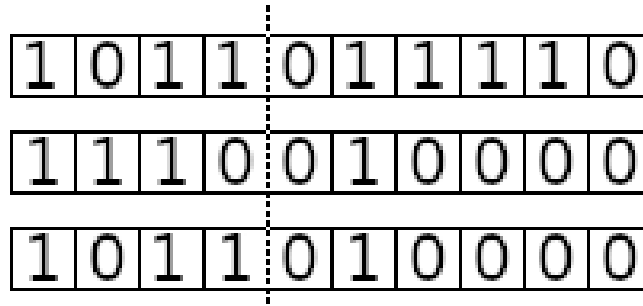
2.1.3 Krzyżowanie

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

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

- **Krzyżowanie jednopunktowe:**

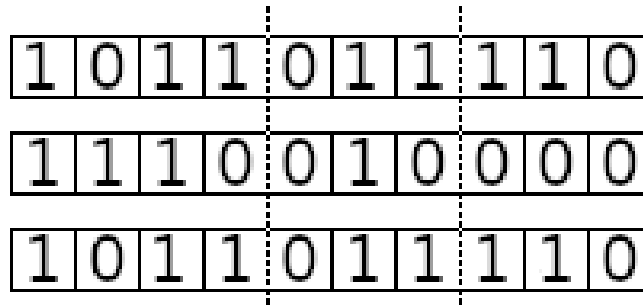
W tej odmianie potomka tworzymy z dwóch wybranych przez selekcję rodziców a następnie losujemy punkt l z przedziału $[0, n_x)$. Potomek wartości na pozycjach $[0, l)$ przyjmuje wartości z pierwszego rodzica, natomiast na pozycjach $[l, n_x)$ z drugiego.



Rysunek 2: Krzyżowanie jednopunktowe, gdzie $n_x = 10$ i $l = 4$.

- **Krzyżowanie dwupunktowe:**

Sposób ten jest zbliżony do opisanego wyżej lecz zamiast jednego punktu losujemy dwa $l_1, l_2 \in [0, n_x)$, gdzie $l_1 < l_2$. Potomek natomiast przyjmuje wartości z pierwszego rodzica poza elementami na pozycjach $[l_1, l_2)$ gdzie wstawiamy elementy z drugiego.



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

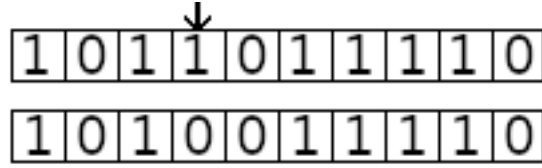
Krzyżowania tego rodzaju można uogólnić na operacje gdzie losujemy k punktów a potomek naprzemiennie pobiera wartości z rodziców.

- **Krzyżowanie równomierne:**

Potomek każdy element jest wybierany z obu rodziców z takim równym prawdopodobieństwem. (Można również wybrać inny stosunek prawdopodobieństw).

2.1.4 Mutacja

Operator ten w klasycznej wersji algorytmu polega na zamienieniu wartości znajdującej się na losowo wybranej pozycji z bardzo małym prawdopodobieństwem σ_m (np. 0.2%).



Rysunek 4: Podstawowa mutacja dla genomu o $n_x = 10$ i wylosowanej pozycji 3.

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. Zostały one 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 z 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 ważoną sumę sygnałów wejściowych:

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

gdzie:

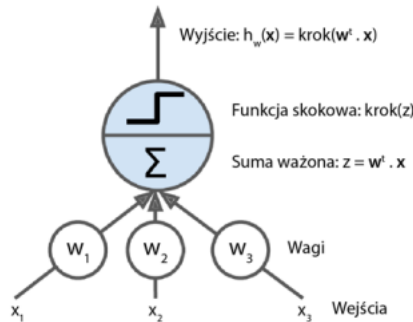
w_{ij} - waga połączenia pomiędzy i -tym neuronem wejściowym a j -tym wyjściowym,

x_i - i -ta wartość wejściowa.

Następnie wynik otrzymany w (1) 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

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 jest aktualizowana według wzoru:

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

gdzie:

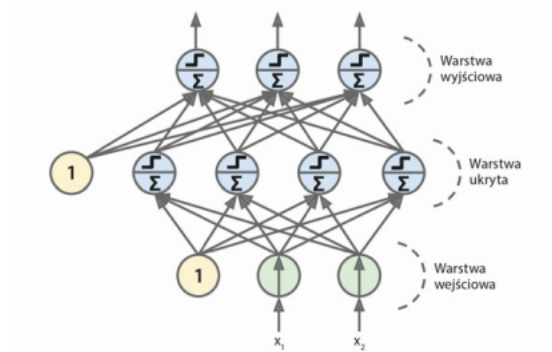
\hat{y}_i - otrzymany wynik na j -tym wyjściu,

y_j - docelowy wynik j -tego neuronu,

η - współczynnik uczenia. Perceptron jednak nie nadaje się do skomplikowanych zadań ponieważ przez swoją prostą budowę jest jedynie zdolny do klasyfikowania danych na zbiory, 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**. Dodatkowa w każdej warstwie znajdują się **neuron obciążający** jego 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 pisanymi 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).



Rysunek 6: Perceptron wielowarstwowy z jedną warstwą ukrytą.

Lecz ponieważ nasza sieć składa się z wielu warstw metoda uczenia używana dla pojedynczego perceptronu nie spiszę się w tym przypadku. Najpopularniejszą metodą uczenia sieci neuronowych jest **wsteczna propagacja** (alternatywnie do uczenia SSN można użyć np. Algorytmów Genetycznych). Cały proces możemy przedstawić za pomocą kroków:[[stronka_z_ssn](#)]:

Algorithm 2: Procedura uczenia wielowarstwowej SSN.

- 1 Wybór parametrów sieci (liczba warstw, liczba neuronów w warstwach itd.)
 - 2 Wagi inicjujemy losowo.
 - 3 Błąd 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.

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

- **funkcja tangensa hiperbolicznego:**

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

$$\tanh(z) = 2\sigma(2z) - 1 \quad (6)$$

- **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 .

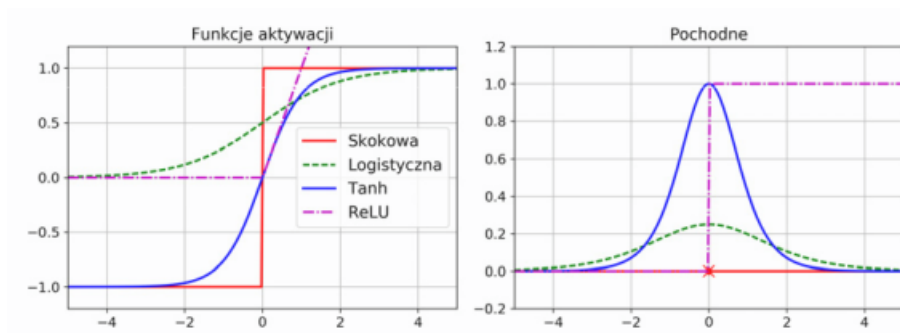
Po wyliczeniu $s_k(\mathbf{x})$ dla każdej klasy, obliczane jest odpowiednio znormalizowane prawdopodobieństwo przynależności danej próbki do klasy k :

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

gdzie: K - liczba klas.

Model prognozuje klasę o najwyższym prawdopodobieństwie:

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

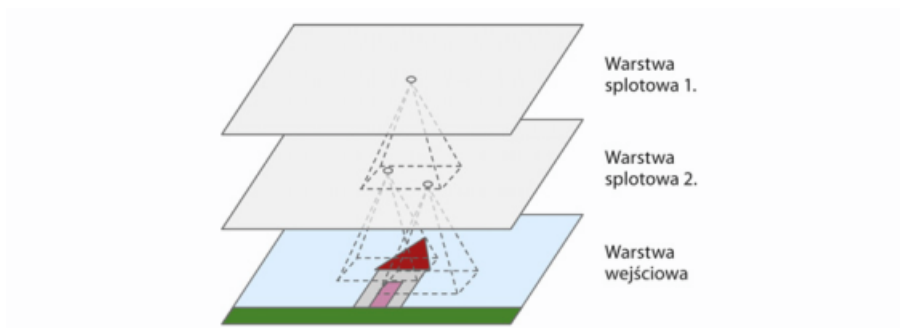


Rysunek 7: Funkcje aktywacyjne i ich pochodne.

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 innych 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 połączony z każdym pikselem obrazu wejściowego, lecz wyłącznie z pikselami znajdującymi się w ich polu recepcyjnym.

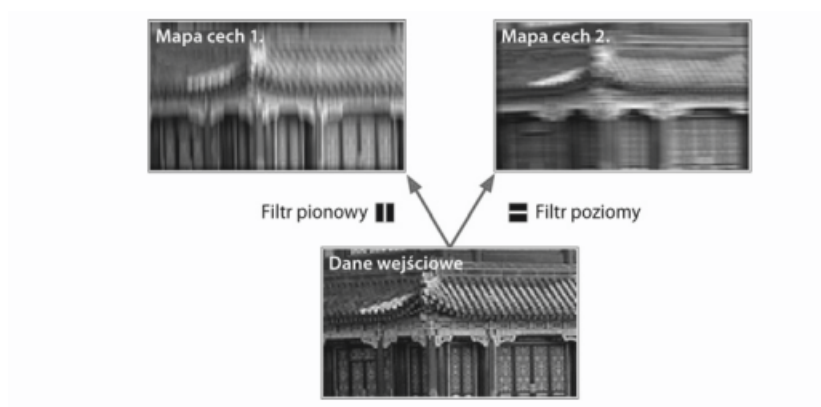


Rysunek 8: Warstwy CNN z prostokątnymi lokalnymi polami recepcyjnymi.

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**

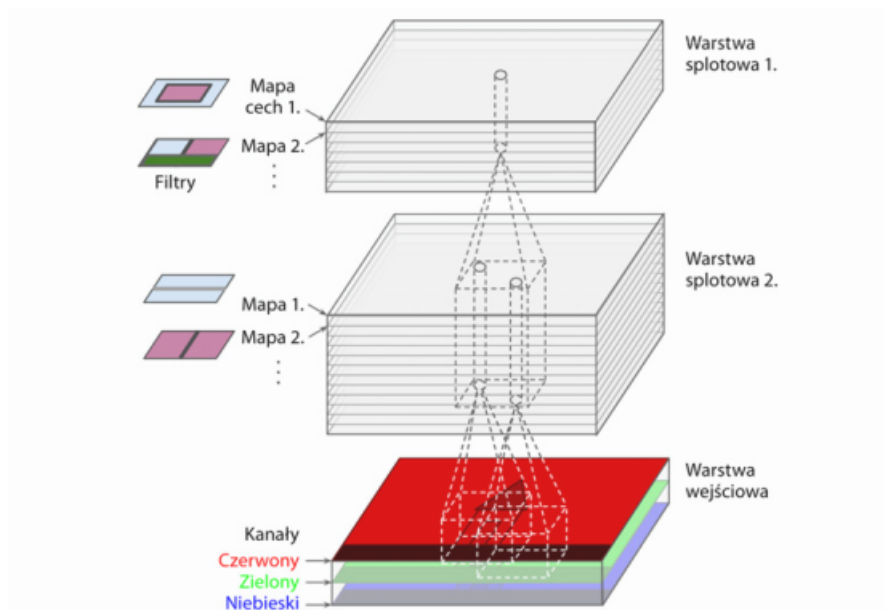
odpowiednio w kolumnach i rzędach. **Krokiem** nazywamy odległość pomiędzy dwoma kolejnymi polami recepcyjnymi. Aby uzyskać takie same rozmiary wymiary dla każdej warstwy najczęściej dodawane są zera wokół wejść.

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.



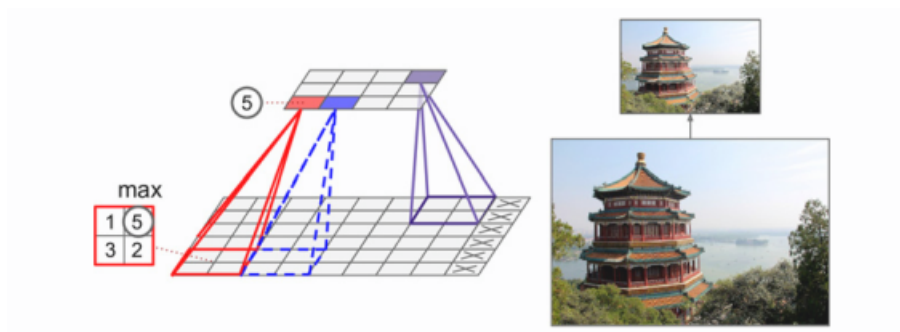
Rysunek 9: Dwie mapy cech otrzymane przy pomocy dwóch różnych filtrów.

Warstwa wypełniona neuronami o takim samym filtrze nazywamy **mapą cech** która pomaga nam wyszczególnić elementy przypominające użyty filtr. Warstwa spłotowa składa się z kilku map cech o identycznych rozmiarach. Każda mapa ma swoje wartości parametrów, dzięki czemu stosując przez stosowanie różnych filtrów warstwa spłotowa 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 spłotowa zawierająca wiele cech, oraz zdjęcie z trzema kanałami barw.

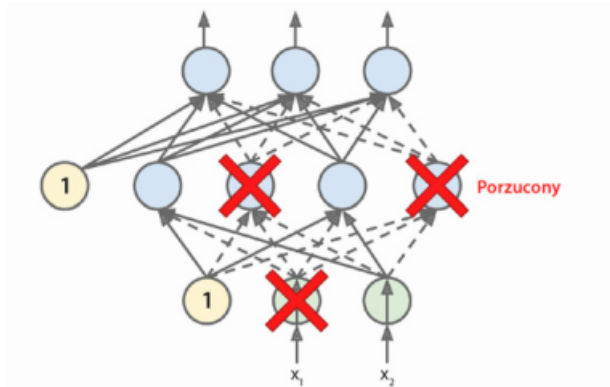
- **warstwa łącząca:** Jej celem jest zmniejszenie obrazu wejściowego w celu zredukowania obciążenia obliczeniowego, wykorzystania pamięci i liczby parametrów. Tak samo jak w warstwach spłotowych neurony łączą się z wyjściami określonej liczby neuronów poprzedniej warstwy, które mieszczą się w obszarze 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.

- **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.

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 ϕ^*

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	Liniowa

Dodanie innych typów warstw (np. rekurencyjnych) i funkcji nie powinno być problemem, jedynie trzeba pamiętać o dodaniu odpowiednich zasad budowania dla nich. Przykładowy genom 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]]

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

3.2 Funkcja kosztu i selekcja

3.3 Krzyżowanie

3.4 Mutacja

3.5 Warunek końca

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] David E. Goldberg. *Algorytmy genetyczne i ich zastosowania*. 1989.