Paweł Kaczyński
Andrzej Siadkowski
Wioletta Klimczak

Projekt z przedmiotu Podstawy Sztucznej Inteligencji

Projekt „Sieci Neuronowe”
ZS.SN.2

Treść zadania:

W laboratorium badane jest czy białka reagują z pewnym enzymem. Każde białko opisane
jest przy pomocy trzech parametrów. Napisać aplikację sygnalizującą czy dane białko będzie
reagować z danym enzymem czy też nie.

Realizacja projektu:

Sieć neuronowa została napisana w języku programistycznym C++.
Środowisko, z którego korzystaliśmy przy tworzeniu aplikacji to „Visual Studio 2013”.
Dodatkowo korzystaliśmy z repozytorium „GitHub”.
Do organizacji pracy zostało wykorzystane również narzędzie „Trello”.

Działanie sieci neuronowej:

Sieć neuronowa składa się z 3 warstw. Warstwy wejściowej, jednej warstwy ukrytej oraz warstwy wyjściowej. Warstwa wejściowa ma 3 neurony, zaś warstwa wyjściowa 1 neuron. Warstwa ukryta składa się w chwili obecnej z 2 neuronów, zaś nasza sieć była testowana również dla innej ilości neuronów w warstwie ukrytej (więcej na ten temat w rozdziale „Testowanie”).

Działanie pierwszej warstwy, warstwy wejściowej:
Warstwa ta pobiera dane na wejściu, tak że każdy neuron zbiera jedną zmienną spośród danych. Nasze dane dla których mieliśmy przeprowadzić symulację składają się z 3 danych, zawierających zakodowane parametry białka (ARGP820102, CHAM830104, QIAN880116). W związku z powyższym faktem warstwa wejściowa składa się z 3 neuronów wejściowych.

Działanie warstwy ukrytej:
Każdy neutron warstwy ukrytej pobiera wartości z poprzedzającej ją warstwy, a następnie sumuje te wartości, przemnażając je odpowiednio przez różne, odpowiednie wagi, oraz dodatkowo dodaje współczynnik wagowy niezależny od poprzedniej warstwy. W kolejnym kroku wyniki z sumowania podawane są funkcji tangensa hiperbolicznego, który przemnożony jest przez współczynnik Beta.

Działanie warstwy wyjściowej:
Warstwa wyjściowa pobiera wartości z poprzedniej warstwy i po prostu je sumuje, dodając podobnie jak warstwy ukryte współczynnik wagowy niezależny od poprzedniej warstwy. Jest to warstwa liniowa.

W momencie, gdy mamy już wyniki na każdej warstwie, idziemy od tyłu i obliczmy wartość błędu dla ostatniej warstwy. Robimy to w następujący sposób:
Błąd na ostatniej warstwie jest to spodziewana wartość (pobierana z pliku) – wynik obliczeń sieci. Zgodnie z metodą propagacji wstecznej błąd ten powinien zostać przemnożony przez pochodną funkcji aktywującej, jednak jako że jest to warstwa liniowa, pochodna tej funkcji wynosi 1.
Gdy mamy już obliczony ten błąd zaczynamy propagację wsteczną do poprzedniej warstwy. Robione jest to w następujący sposób:

1) Liczona jest suma błędów neuronów warstwy następnej przemnożonych przez wartość wagi, z jaką neuron dla którego teraz liczony jest błąd wchodzi w skład sumy neuronu warstwy następnej.
2) Policzona przed chwilą suma jest mnożona przez pochodną funkcji aktywacji. Argumentem tej pochodnej jest suma obliczona podczas obliczania wyników sieci neuronowej, tj. wejście warstwy poprzedniej przemnożona przez odpowiednią wagę i skorygowana wagą wolną

Mając obliczony błąd, korygujemy wagi. Cel ten realizowany jest w następujący sposób:

*Waga += błąd danego neuronu * współczynnik eta (współczynnik uczenia się) * pobrane wejście neuronu, dla którego korygowana jest waga + współczynnik alfa (współczynnik momentum) * (waga w tej iteracji - waga w iteracji poprzedniej)*

Waga wolna (ta której nie jest przypisany żaden neuron warstwy poprzedniej) jest korygowana podobnie, pominięte jest jedynie wejście neuronu, zatem wzór wygląda następująco

*Waga += błąd danego neuronu * współczynnik eta + współczynnik alfa * (waga w tej iteracji - waga w iteracji poprzedniej)*

Klasy Sieci Neuronowej:

Klasa Network:
Główna klasa programu agregująca pozostałe moduły. W głównej metodzie run()wywoływane są wszystkie najważniejsze metody innych klas. Przetrzymuje ona też informacje o warstwach i sieciach.

Klasa Interface:
Klasa zajmująca się interpretowaniem i zapisywaniem danych wejściowych do pliku (parametry). Na podstawie zdefiniowanych kluczy w odpowiednich mapach, przetwarza podane w konstruktorze argumenty programu i zapisuje informacje o nich lub zwraca błąd. Klucze odwołują się do wartości logicznych, kiedy parametr wystąpił lub nie albo do napisów, takich jak np. nazwa pliku wejściowego. Udostępnia pobieranie informacji o parametrach za pomocą metod typu get(). Oprócz tego zawiera też powszechnie znane wiadomości „help” oraz „usage” wyświetlane po podaniu odpowiednich parametrów lub gdy parametry są błędne, bądź nie ma ich wcale.

Klasa OutputManager:
Klasa odpowiedzialna za przetwarzanie danych wyjściowych i zapisy do różnych plików. Posiada zdefiniowane domyślne wartości dla ścieżek do plików rezultatów, szczegółowych statystyk dla danego uruchomienia i uśrednionych statystyk globalnych. Użytkownik może oczywiście narzucić swoje parametry dla pliku wyników i statystyk „lokalnych”. Statystyki globalne posiadają wersje mniej i bardziej szczegółowe (patrz niżej: współczynniki jakości walidacji) oraz wersje sformatowane oraz nie (sformatowane do podglądu tekstowego, niesformatowane do importu w programach typu Excel).

Przez podane w argumencie odpowiednich metod zapisujących kontenery danych, klasa ta wyciąga informacje o warstwach i iteracjach w procesie testowania i formatując (lub nie) odpowiednio linijki tekstu, zapisuje je na dysku. Przygotowując statystyki lokalne, wykorzystywany jest specjalny szablon w pliku tekstowym ze słowami kluczowymi, które są zamieniane na odpowiednie wartości i kopiowane do odpowiedniej lokalizacji.

Klasa IterationInfo:
Przetrzymuje informacje statystyczne o danej iteracji. Podstawowe dane to wartości klasyfikacji TP, TN, FP oraz FN. Oprócz tego zawiera szereg współczynników jakości walidacji, np. Accuracy, Specificity, Precision czy F1 score (wzory w pliku nagłówkowym). Pliki okrojone zawierają najbardziej adekwatny parametr, czyli Accuracy, a szczegółowe także wszystkie pozostałe. Klasa udostępnia też operatory dodawania dwóch obiektów typu IterationInfo oraz dzielenia przez liczbę (w celu uśrednienia). Klasa jest klasą szablonową, ponieważ raz chcemy mieć ilości TP, TN itd. jako zmienne całkowite, a raz jako rzeczywiste, jeśli liczymy średnią.

Klasa CrossValidator:
Klasa zawiera kontener przechowujący wiele obiektów typu IterationInfo. Zawiera informacje o liczbie iteracji i przez odpowiednie metody zwraca informacje o podziałach zbioru testowego na grupy do klasy Network. Zdefiniowany jest w niej typ TestResults, który przetrzymuje wektor par <bool, bool>, które mogą być interpretowane jako <wartość oczekiwana, wartość uzyskana> i na ich postawie wyliczane są TP, TN itd.

Inne klasy:

Klasa Utilities:
Statyczna klasa zawierająca konwersję pomiędzy std::string i innymi typami oraz metodę zamieniającą słowo kluczowe w tekście na podany łańcuch znaków.

Klasa XMLParser:
Klasa przetwarzająca plik XML. Potrzebna była do wygodnego parametryzowania właściwości sieci, liczby warstw i neuronów. Po przetworzeniu pliku tekstowego, struktura znaczników i atrybutów XML jest przechowywana w formie drzewa. Klasa udostępnia metody wyciągający informacje o znacznikach i atrybutach.

Struktura plików:

Katalog projektu

- Pliki źródłowe i nagłówkowe
- _results
    - Pliki z wynikami (aczkolwiek mało przydatne dla takich skomplikowanych danych)
- _stats
    - Pliki ze statystykami dla każdego uruchomienia.
- Data
    - headers
        - Nagłówki plików statystyk do tworzenia w razie potrzeby nowych.
        - Plik szablonowy statystyk per uruchomienie.
    - Pliki z danymi testowymi
    - Plik konfiguracyjny sieci
    - Pliki z globalnymi statystykami w różnych konfiguracjach.

Testowanie:

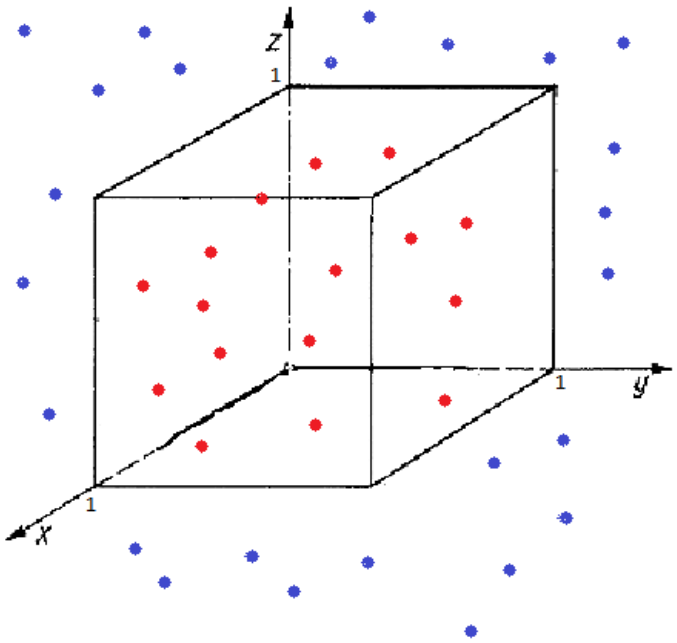
Wywołanie programu:

Program wywoływany jest z następującymi argumentami (podany końcowy plik jest przykładowy):

-f ./Data/testData.txt

Dane wczytywane przez naszą aplikację powinny być następującej postaci:
[spacja] [spacja / -] [liczba][e][znak][liczba] [spacja][spacja albo znak] [następna liczba itd...]

Początkowe testowanie:
Początkowo aplikacja była testowana dla własnych, specjalnie wygenerowanych danych.
Dane te były generowane przy pomocy Excela i dostępnej tam funkcji los().
Następnie były odpowiednio konwertowane, tak aby wczytanie ich przez naszą aplikację było możliwe.
Generacja danych opierała się na koncepcji związanej z sześcianem (*Rys. 1*).
Wartości znajdujące się wewnątrz sześcianu (*ozn. czerwone kropeczki*) były oznaczane przez wartość '1' (jako te 'reagujące'), z kolei dane będące na zewnątrz sześcianu (*ozn. niebieskie kropeczki*) zostały oznaczone wartością '-1' (jako te 'nie reagujące').



Rys. 1

Fragment wygenerowanych przez nas losowo danych widoczny jest poniżej (*Rys. 2*).

8.8993453e-001	3.1870925e-001	7.2641574e-001	1.0000000e+000
9.5272923e-001	5.4638929e-001	6.5962144e-001	1.0000000e+000
6.9980379e-001	9.3546681e-001	4.1523353e-001	1.0000000e+000
6.9667084e-001	2.4092192e-001	3.1430771e-001	1.0000000e+000
6.8082843e-001	8.5247783e-001	7.2836316e-001	1.0000000e+000
9.6938223e-001	4.8129572e-001	7.0452068e-001	1.0000000e+000
8.0989533e-001	9.1809456e-001	7.8597650e-001	1.0000000e+000
3.0341730e-001	5.3904923e-001	2.9772099e-001	1.0000000e+000
6.9367920e-001	3.5766265e-001	4.2229548e-001	1.0000000e+000
9.9966763e-001	9.5895380e-001	5.1095073e-001	1.0000000e+000
8.9638630e-001	4.7953515e-001	3.6184619e-001	1.0000000e+000
6.8974901e-001	7.0345149e-001	5.7900646e-001	1.0000000e+000
3.2637037e+000	2.0821121e+000	1.2776893e+000	-1.0000000e+000
1.3728246e+000	2.5582823e+000	1.6658936e+000	-1.0000000e+000
1.7254626e+000	1.8457423e+000	3.3910934e+000	-1.0000000e+000
1.3334960e+000	2.4305781e+000	1.5853706e+000	-1.0000000e+000
2.7237427e+000	2.5752468e+000	2.6377571e+000	-1.0000000e+000
1.6890013e+000	1.0127856e+000	2.7029558e+000	-1.0000000e+000

Rys. 2

Oznaczenia:
Nasze wyniki oraz trafienia sieci neuronowej były oznaczane w następujący sposób:
TP – trafiliśmy i celowaliśmy w ‘1’ (reagujące)
TN – trafiliśmy i celowaliśmy w ‘-1’ (nie reagujące)
FP – nie trafiliśmy i celowaliśmy w ‘-1’ (nie reaguje)
FN – nie trafiliśmy i celowaliśmy w ‘1’ (reaguje)

Pomocniczo przedstawione jest to na *Rys. 3*

		Actual	
		True	False
Classifier	True	True Positive	False Positive
	False	False Negative	True Negative

Rys. 3

Test 1:

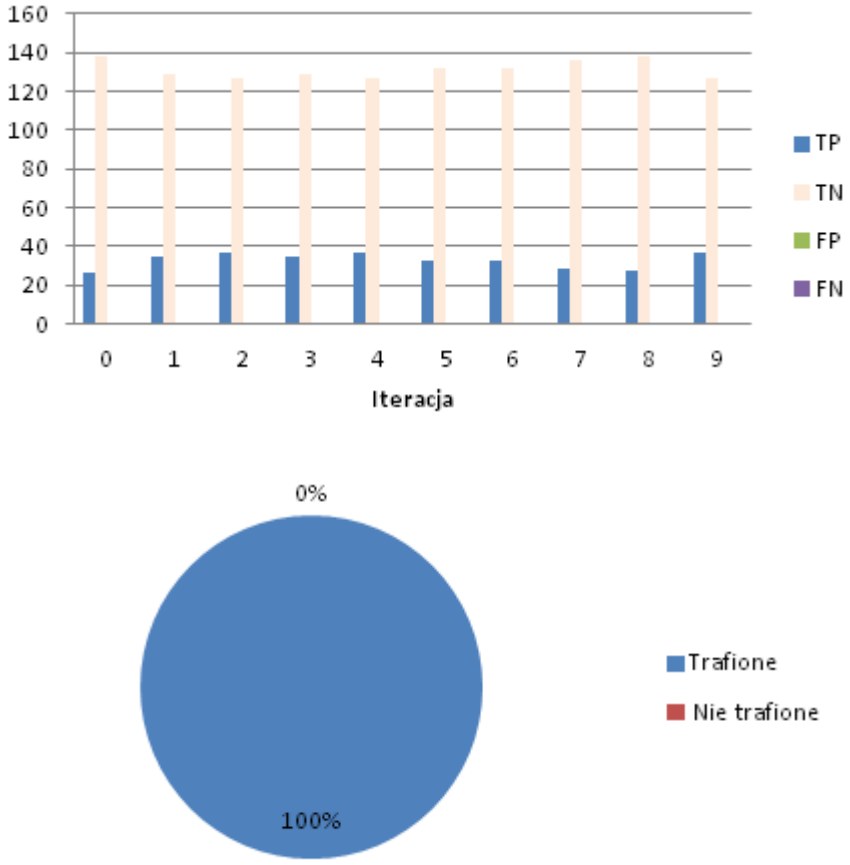
Dla tak wygenerowanych danych i dla 2 neuronów warstwy ukrytej otrzymaliśmy następujące statystyki (*Rys. 4*):

```
        >>> LAYERS INFO <<<
INPUT      | 3
HIDDEN     | 2
OUTPUT     | 1

ITERATION | TP      | TN      | FP      | FN
=================================================
0         | 26      | 138     | 0       | 0
1         | 35      | 129     | 0       | 0
2         | 37      | 127     | 0       | 0
3         | 35      | 129     | 0       | 0
4         | 37      | 127     | 0       | 0
5         | 32      | 132     | 0       | 0
6         | 32      | 132     | 0       | 0
7         | 28      | 136     | 0       | 0
8         | 26      | 138     | 0       | 0
9         | 37      | 127     | 0       | 0
```

Rys. 4

Co przedstawia się następująco (*Rys. 5*)



Rys. 5

Test 2:

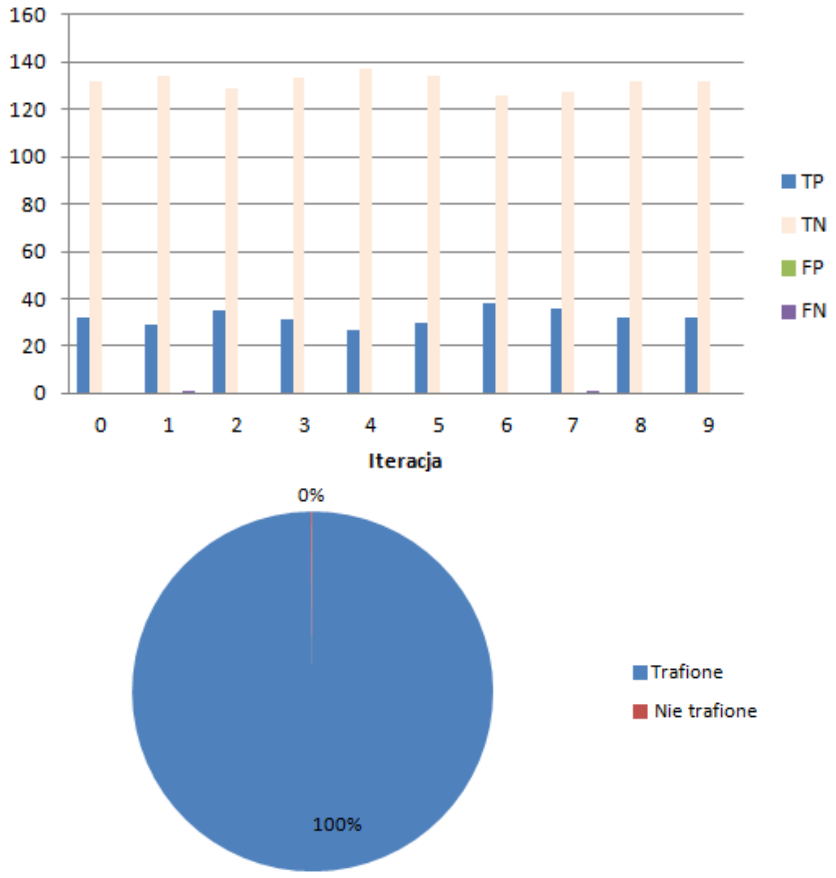
Dla tak wygenerowanych danych i dla 4 neuronów warstwy ukrytej otrzymaliśmy następujące statystyki (*Rys. 6*):

```
        >>> LAYERS INFO <<<
INPUT      | 3
HIDDEN     | 4
OUTPUT     | 1

ITERATION | TP      | TN      | FP      | FN
=================================================
0         | 32      | 132     | 0       | 0
1         | 29      | 134     | 0       | 1
2         | 35      | 129     | 0       | 0
3         | 31      | 133     | 0       | 0
4         | 27      | 137     | 0       | 0
5         | 30      | 134     | 0       | 0
6         | 38      | 126     | 0       | 0
7         | 36      | 127     | 0       | 1
8         | 32      | 132     | 0       | 0
9         | 32      | 132     | 0       | 0
```

Rys. 6

Co przedstawia się następująco (*Rys. 7*)



Rys. 7

Test 3:

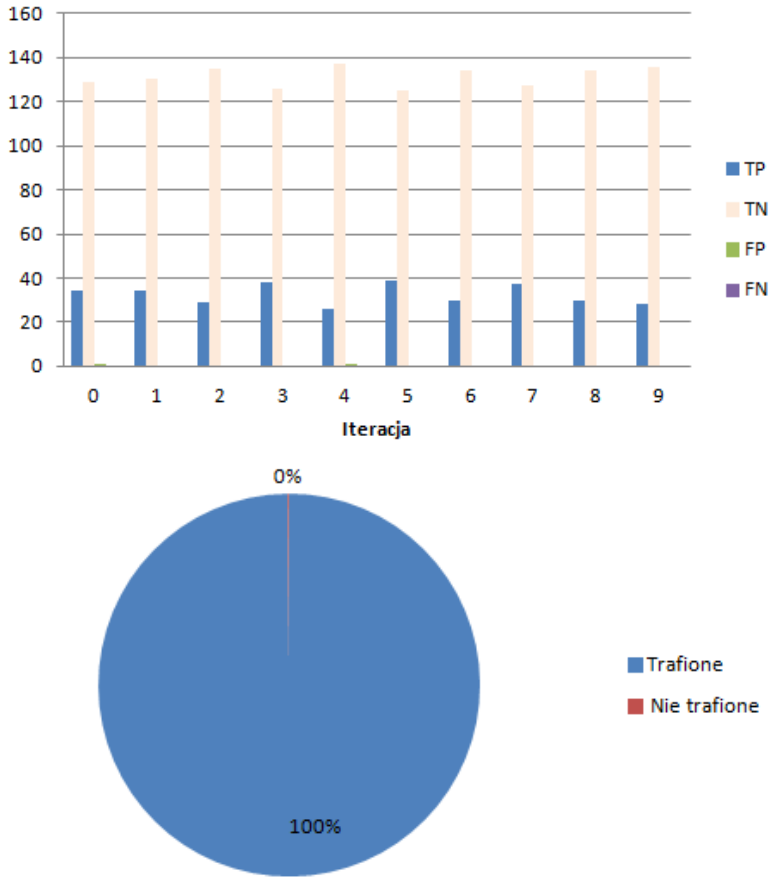
Dla tak wygenerowanych danych i dla 8 neuronów warstwy ukrytej otrzymaliśmy następujące statystyki (*Rys. 8*):

```
        >>> LAYERS INFO <<<
INPUT      | 3
HIDDEN     | 8
OUTPUT     | 1

ITERATION | TP      | TN      | FP      | FN
=============================================
0         | 34      | 129     | 1       | 0
1         | 34      | 130     | 0       | 0
2         | 29      | 135     | 0       | 0
3         | 38      | 126     | 0       | 0
4         | 26      | 137     | 1       | 0
5         | 39      | 125     | 0       | 0
6         | 30      | 134     | 0       | 0
7         | 37      | 127     | 0       | 0
8         | 30      | 134     | 0       | 0
9         | 28      | 136     | 0       | 0
```

Rys. 8

Co przedstawia się następująco (*Rys. 9*)



Rys. 9

Wniosek:
Patrząc na wyniki powyższych testów można zauważyć, iż nasza Sieć Neuronowa działa poprawnie. Widać również, że optymalne są 2 neurony w warstwie ukrytej.

Testy dla danych białkowych – właściwych:

Fragment właściwych danych z zapisanymi informacjami o białkach widoczny jest poniżej (*Rys. 10*).

-5.7309540e-001	3.5413246e-001	1.0489510e-001	-1.0000000e+000
-9.8352780e-001	1.0000000e+000	-3.8461538e-001	-1.0000000e+000
-4.8798902e-001	-1.0000000e+000	1.0000000e+000	-1.0000000e+000
-4.8798902e-001	-1.0000000e+000	1.0000000e+000	-1.0000000e+000
-4.3582704e-001	-3.3552271e-001	2.4475524e-001	-1.0000000e+000
-8.7508579e-001	1.9102354e-001	6.2937063e-002	-1.0000000e+000
-9.1077557e-001	3.1198686e-002	7.6223776e-001	-1.0000000e+000
1.0000000e+000	9.7044335e-001	-6.2237762e-001	-1.0000000e+000
1.0000000e+000	9.7044335e-001	-6.2237762e-001	-1.0000000e+000
-9.1077557e-001	3.1198686e-002	7.6223776e-001	-1.0000000e+000
-3.0885381e-002	9.7044335e-001	-8.6013986e-001	-1.0000000e+000
-8.0645161e-001	5.3585112e-001	7.4825175e-001	-1.0000000e+000
-8.6684969e-001	2.0963328e-001	-3.2867133e-001	-1.0000000e+000
1.4619080e-001	9.8905309e-001	-8.0419580e-001	-1.0000000e+000
-4.8798902e-001	-1.0000000e+000	1.0000000e+000	-1.0000000e+000
-8.7508579e-001	-9.3596059e-002	6.7832168e-001	-1.0000000e+000
1.0000000e+000	9.7044335e-001	-6.2237762e-001	1.0000000e+000
1.0000000e+000	9.7044335e-001	-6.2237762e-001	1.0000000e+000
1.0000000e+000	9.7044335e-001	-6.2237762e-001	1.0000000e+000
1.0000000e+000	9.7044335e-001	-6.2237762e-001	1.0000000e+000
1.0000000e+000	9.7044335e-001	-6.2237762e-001	1.0000000e+000
1.0000000e+000	9.7044335e-001	-6.2237762e-001	1.0000000e+000
1.0000000e+000	9.7044335e-001	-6.2237762e-001	1.0000000e+000
1.0000000e+000	9.7044335e-001	-6.2237762e-001	1.0000000e+000
1.0000000e+000	9.7044335e-001	-6.2237762e-001	1.0000000e+000
1.0000000e+000	9.7044335e-001	-6.2237762e-001	1.0000000e+000

Rys. 10

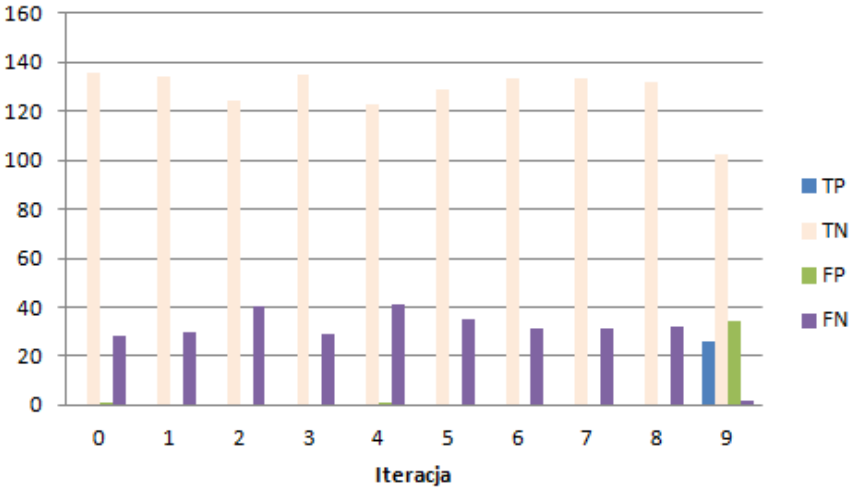
Test właściwy 1:
Dla właściwych danych białkowych i dla 2 neuronów warstwy ukrytej otrzymaliśmy następujące statystyki (*Rys. 11*):

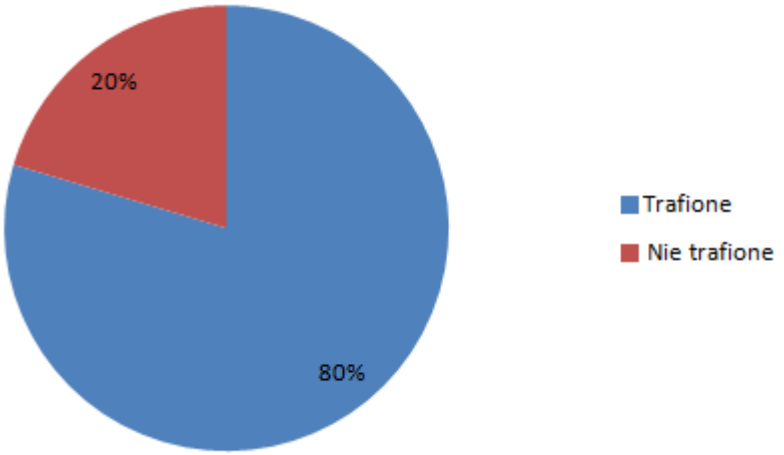
```
        >>> LAYERS INFO <<<
INPUT      | 3
HIDDEN     | 2
OUTPUT     | 1

ITERATION | TP      | TN      | FP      | FN
=============================================
0         | 0       | 136     | 0       | 28
1         | 0       | 134     | 0       | 30
2         | 0       | 124     | 0       | 40
3         | 0       | 135     | 0       | 29
4         | 0       | 123     | 0       | 41
5         | 0       | 129     | 0       | 35
6         | 0       | 133     | 0       | 31
7         | 0       | 133     | 0       | 31
8         | 0       | 132     | 0       | 32
9         | 26      | 102     | 34      | 2
```

Rys. 11

Co przedstawia się następująco (*Rys. 12*)





Rys. 12

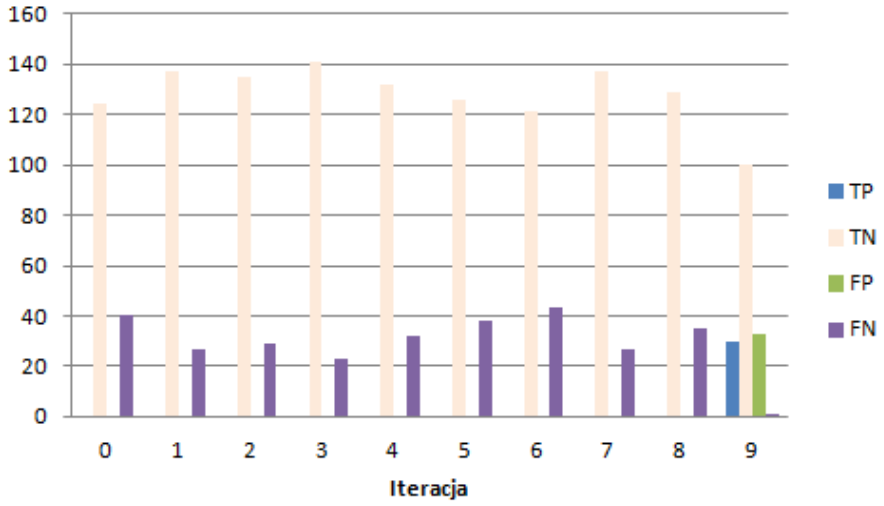
Test właściwy 2:
Dla właściwych danych białkowych i dla 4 neuronów warstwy ukrytej otrzymaliśmy następujące statystyki (*Rys. 13*):

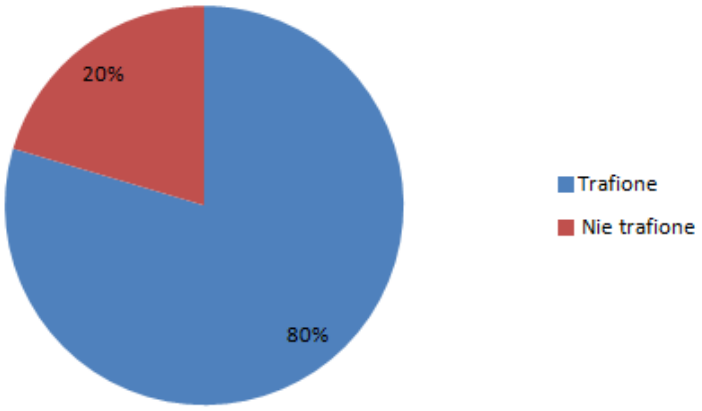
```
        >>> LAYERS INFO <<<
INPUT      | 3
HIDDEN     | 4
OUTPUT     | 1

ITERATION | TP      | TN      | FP      | FN
=============================================
0         | 0       | 124     | 0       | 40
1         | 0       | 137     | 0       | 27
2         | 0       | 135     | 0       | 29
3         | 0       | 141     | 0       | 23
4         | 0       | 132     | 0       | 32
5         | 0       | 126     | 0       | 38
6         | 0       | 121     | 0       | 43
7         | 0       | 137     | 0       | 27
8         | 0       | 129     | 0       | 35
9         | 30      | 100     | 33      | 1
```

Rys. 13

Co przedstawia się następująco (*Rys. 14*)





Rys. 14

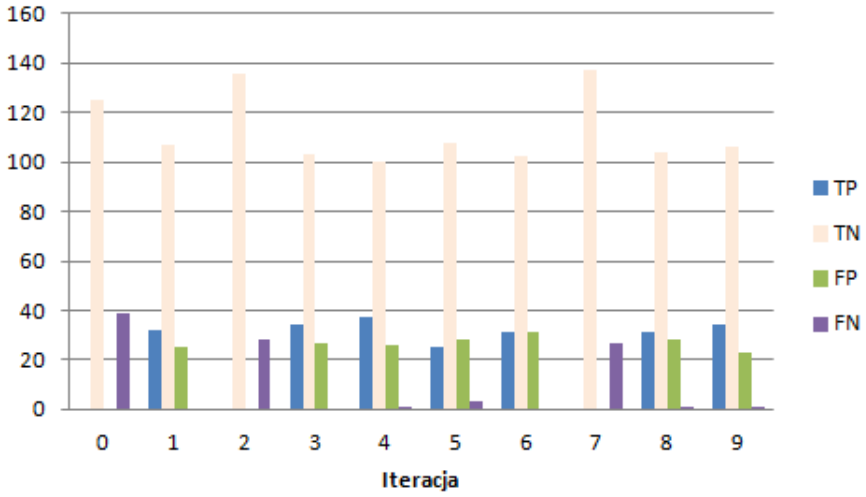
Test właściwy 3:
Dla właściwych danych białkowych i dla 6 neuronów warstwy ukrytej otrzymaliśmy następujące statystyki (*Rys. 15*):

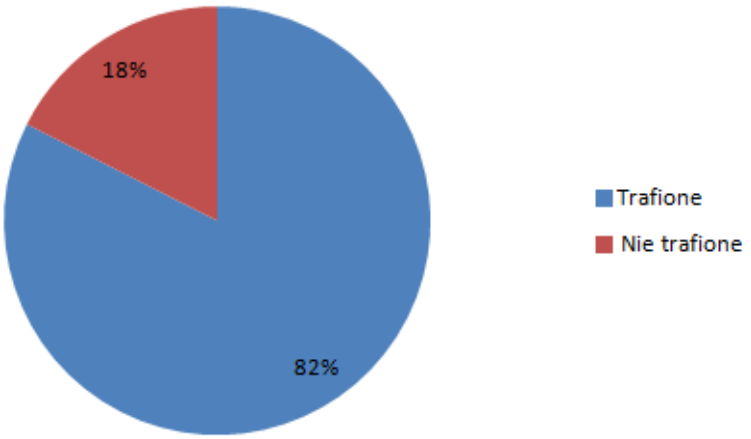
```
        >>> LAYERS INFO <<<
INPUT      | 3
HIDDEN     | 6
OUTPUT     | 1

ITERATION | TP      | TN      | FP      | FN
=============================================
0         | 0       | 125     | 0       | 39
1         | 32      | 107     | 25      | 0
2         | 0       | 136     | 0       | 28
3         | 34      | 103     | 27      | 0
4         | 37      | 100     | 26      | 1
5         | 25      | 108     | 28      | 3
6         | 31      | 102     | 31      | 0
7         | 0       | 137     | 0       | 27
8         | 31      | 104     | 28      | 1
9         | 34      | 106     | 23      | 1
```

Rys. 15

Co przedstawia się następująco (*Rys. 16*)





Rys. 16

Test właściwy 4:

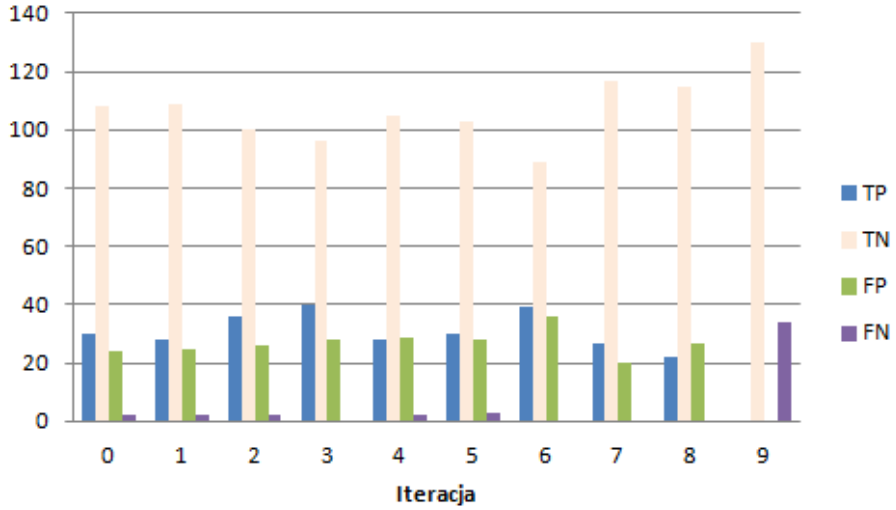
Dla właściwych danych białkowych i dla 8 neuronów warstwy ukrytej otrzymaliśmy następujące statystyki (*Rys. 17*):

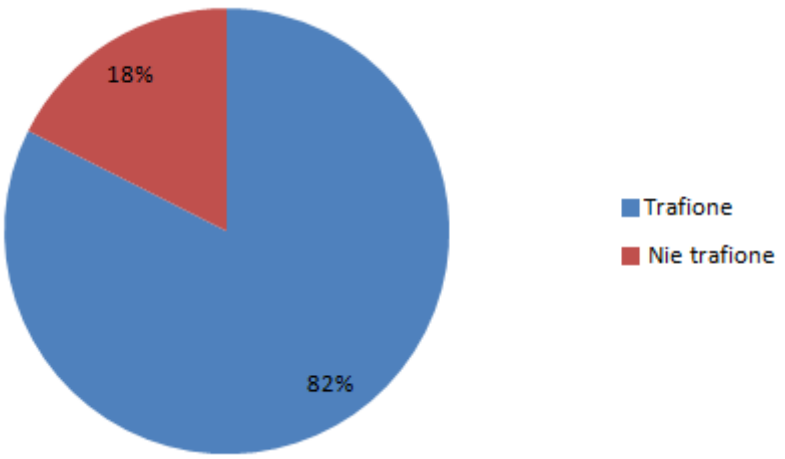
```
        >>> LAYERS INFO <<<
INPUT      | 3
HIDDEN     | 8
OUTPUT     | 1

ITERATION | TP      | TN      | FP      | FN
=============================================
0         | 30      | 108     | 24      | 2
1         | 28      | 109     | 25      | 2
2         | 36      | 100     | 26      | 2
3         | 40      | 96      | 28      | 0
4         | 28      | 105     | 29      | 2
5         | 30      | 103     | 28      | 3
6         | 39      | 89      | 36      | 0
7         | 27      | 117     | 20      | 0
8         | 22      | 115     | 27      | 0
9         | 0       | 130     | 0       | 34
```

Rys. 17

Co przedstawia się następująco (Rys. *18*)





Rys. 18

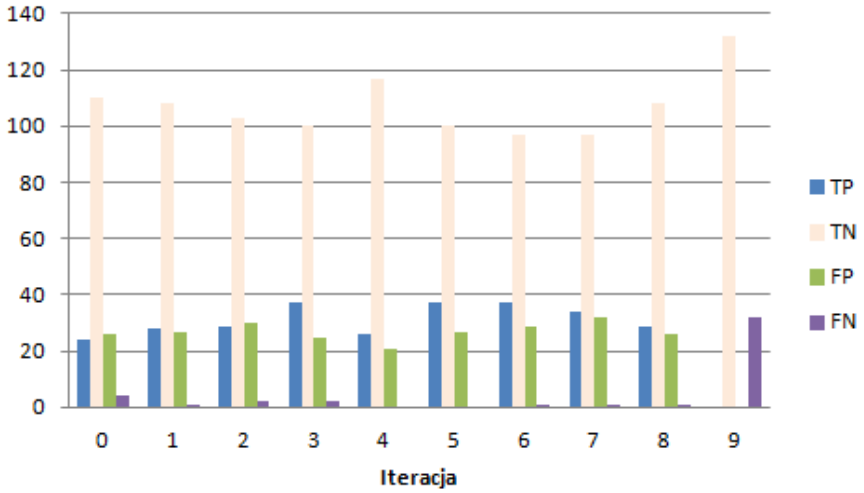
Test właściwy 5:
Dla właściwych danych białkowych i dla 16 neuronów warstwy ukrytej otrzymaliśmy następujące statystyki (*Rys. 19*):

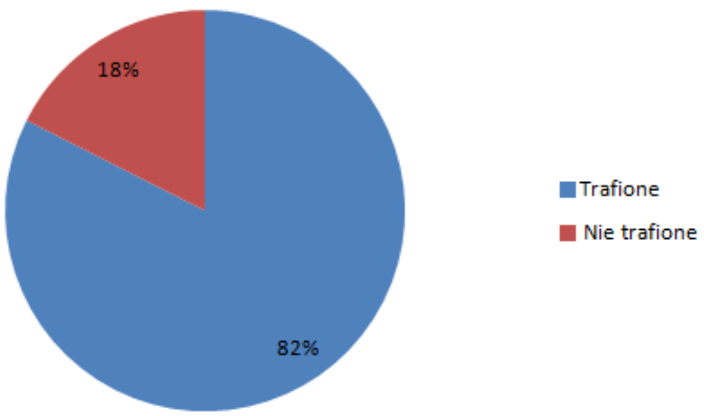
```
        >>> LAYERS INFO <<<
INPUT      | 3
HIDDEN     | 16
OUTPUT     | 1

ITERATION | TP      | TN      | FP      | FN
=============================================
0         | 24      | 110     | 26      | 4
1         | 28      | 108     | 27      | 1
2         | 29      | 103     | 30      | 2
3         | 37      | 100     | 25      | 2
4         | 26      | 117     | 21      | 0
5         | 37      | 100     | 27      | 0
6         | 37      | 97      | 29      | 1
7         | 34      | 97      | 32      | 1
8         | 29      | 108     | 26      | 1
9         | 0       | 132     | 0       | 32
```

Rys. 19

Co przedstawia się następująco (*Rys. 20*)





Rys. 20

Wnioski końcowe:
Dla tak zdefiniowanych danych wejściowych dotyczących białek, skuteczność naszej Sieci Neuronowej jest na poziomie $\geq$ 80%. Dla 2 neuronów w warstwie ukrytej osiągamy skuteczność ok. 80 %. Dla większej ilości neuronów w warstwie ukrytej (np. 6) możemy osiągnąć skuteczność rzędu 82 %.
Fakt, iż skuteczność ta nie jest rzędu 100 % może wynikać np. z faktu, iż zbiór danych wejściowych może być zbiorem rozmytym, a co za tym idzie nie da osiągnąć się 100 % poprawności.