Warszawa, 14.05.2021

Politechnika Warszawska

Wydział Elektroniki i Technik Informacyjnych

Sieci neuronowe w zastosowaniach biomedycznych (SNB)

Projekt: (35) Diagnostyka raka piersi w badaniach mammograficznych za pomocą sieci MLP

ETAP II

Prowadzący: dr inż. Paweł Mazurek

Wykonawca:

Zespół **4**

Jagoda Adamczyk, Aleksandra Krakowiak

Spis treści

1 Wstęp 3
2 Model sieci 3
2.1 Struktura sieci 3
2.2 Przetwarzanie danych 4
2.2.1 Standaryzacja 4
2.2.2 Podział danych na dane treningowe i testowe 5
2.3 Funkcje aktywacji 5
2.4 Algorytm uczenia 6
2.5 Proces uczenia 8
3 Wyniki procesu uczenia sieci 10
3.1 Wykres błędu w zależności od liczby iteracji 10
3.2 Testy opisujące użyteczność sieci 10
3.2.1 Czułość 11
3.2.2 Specyficzność 11
3.2.3 Podsumowanie wyników 12
4 Bibliografia 12
5 Oświadczenie 12

1 Wstęp

Celem wykonywanego projektu jest stworzenie sieci neuronowej MLP. Zadaniem stworzonej sieci, będzie zaklasyfikowanie kobiet do jednej z dwóch grup – łagodnej zmiany mammograficznej lub złośliwej. Klasyfikacja ta umożliwiłaby zmniejszenie liczby wykonywanych niepotrzebnych biopsji. Powyższa klasyfikacja opiera się na zebranych wcześniej informacjach mammograficznych. W projekcie wykorzystywana będzie baza – *Mammogtaphic Mass Data Set*[1] , która została opublikowana w październiku 2007 roku. Baza zawiera dane od 516 kobiet o łagodnej zmianie mammograficznej oraz 445 – złośliwej zmianie. Zestaw danych wejściowych ma charakter wielowymiarowy.

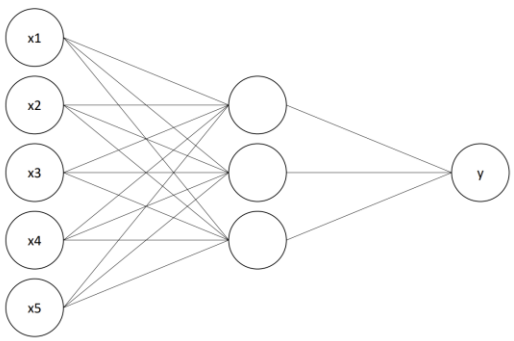
Celem drugiego etapu projektu jest zaimplementowanie algorytmu uczenia sieci oraz przeprowadzenie wstępnych testów. Algorytm uczenia sieci został zaimplementowany w środowisku MATLAB w oparciu o założenia z punktu 2. (opis w 2. Model sieci). Przeprowadzone wstępne testy opierały się na wyznaczeniu wartości błędu w funkcji iteracji oraz wartości wskaźników czułości i specyficzności, które pozwalają na określenie użyteczności stworzonej klasyfikacji.

Projekt będzie wykonywany przy wykorzystaniu oprogramowania MATLAB R2021a przy wykorzystaniu Matlab Driver. Matlab Driver umożliwia współdzielenie pliku, dzięki czemu możliwa jest praca na jednym pliku.

2 Model sieci

2.1 Struktura sieci

Zaprojektowana przez nas sieć to sieć dwuwarstwowa perceptronowa jednokierunkowa (Rysunek 1). Sieć składa się z pięciu neuronów wejściowych, które odpowiadają kolejnym atrybutom w bazie danych. Na etapie dokumentacji wstępnej założono, że ilość neuronów w warstwie ukrytej to 3, z możliwością zmiany tej wartości ze względu na optymalizację działania sieci. Powyższe zagadnienie ma charakter binarny przez to zastosowano jeden neuron wyjściowy. Wszystkie neurony wejściowe są połączone ze wszystkimi neuronami warstwy ukrytej. Neurony warstwy ukrytej są połączone z neuronem wyjściowym (połączenie bezpośrednie). Po każdym kolejnym cyklu uczenia sieci zostają zaktualizowane wszystkie wagi połączeń na podstawie całego zbioru danych treningowych.



Rysunek 1. Schemat sieci

2.2 Przetwarzanie danych

2.2.1 Standaryzacja

Wartości danych wejściowych, które umożliwiają trenowanie sieci są cechami jakościowymi (*BI-RADS, Shape, Margin i Density*) oraz ilościowymi (*Age*) przez co mają różne zakresy w reprezentacji liczbowej. Różne zakresy wartości uniemożliwiają włączenie w równym stopniu wszystkich cech do wyznaczenia wartości wyjściowej. Aby uniknąć takiej sytuacji przeprowadzono standaryzację, która umożliwi ujednolicenie danych. Standaryzację dokonano według poniższych wzorów.

$$\overline{x}_i = \frac{1}{n}\sum_{j=1}^{n} x_j^i \qquad (2.1)$$

$$\sigma_i(x) = \sqrt{\frac{\sum_{j=1}^{n}(x_j^i - \overline{x}_i)^2}{n-1}} \qquad (2.2)$$

$$z_i = \frac{x_i - \overline{x}_i}{\sigma_i(x)} \qquad (2.3)$$

gdzie:

$x_i - i - ta\ cecha\ wektora\ wejściowego$
$\overline{x}_i - wartość\ średnia\ i - tej\ cechy$
$x_j^i - wartość\ i - tej\ cechy\ j - tego\ elemnetu\ wektora\ wejściowego$
$\sigma_i(x) - odchylenie\ standardowe\ i - tej\ cechy$
$z_i - przeskalowana\ wartość\ cechy\ x_i$

Dzięki powyższej transformacji otrzymujemy cechy o wartości średniej równej 0 ($\overline{x}_i = 0$) oraz odchyleniu standardowym równym 1 ($\sigma_i = 1$). Poniżej przedstawiono fragment kodu, który prezentuje wyżej opisaną metodę przetwarzania danych (Rysunek 2).

```
77      %standaryzacja
78 -    srednia = mean(N);
79 -    odchylenie = std(N);
80
81 -    pomoc1 = size(N);
82 -    daneStand = zeros(pomoc1);
83
84 -    for i = 1:pomoc1
85 -        daneStand(i,1) = (N(i,1)-srednia(1,1))./odchylenie(1,1);
86 -        daneStand(i,2) = (N(i,2)-srednia(1,2))./odchylenie(1,2);
87 -        daneStand(i,3) = (N(i,3)-srednia(1,3))./odchylenie(1,3);
88 -        daneStand(i,4) = (N(i,4)-srednia(1,4))./odchylenie(1,4);
89 -        daneStand(i,5) = (N(i,5)-srednia(1,5))./odchylenie(1,5);
90 -        daneStand(i,6) = N(i, 6);
91 -    end
```

Rysunek 2. Fragment kodu źródłowego - przeprowadzenie standaryzacji danych wejściowych

Przetwarzaniu zostały poddane wszystkie dostępne dane (przed podziałem na dane treningowe i testowe (2.2.2)). Kolumny 1-5 zawierają wartości pięciu cech, natomiast ostatnia kolumna zawiera etykiety klasyfikacji danego elementu (0 – łagodna zmiana, 1 – złośliwa zmiana).

2.2.2 Podział danych na dane treningowe i testowe

Dostępne dane zostały podzielone na dane treningowe i testowe. Dane testowe to losowo wybrany zbiór danych z udostępnionych danych (założyłyśmy, że będzie to 15% całego zbioru). Podczas wykonywania podziału na dane testowe i treningowe wykorzystano funkcje *cvpartition* z Statistics and Machine Learning Toolbox, która umożliwiła nam powyższy podział. Rysunek 3. przedstawia zaimplementowany podział.

```
92    %podział na dane testowe i treningowe
93 -  cv = cvpartition(size(daneStand,1),'HoldOut',0.15);
94 -  idx = cv.test;
95 -  dataTrain = daneStand(~idx,:);
96 -  dataTest  = daneStand(idx,:);
97
98 -  dataTrain_test = dataTrain; %pełen zbiór treningowy
99 -  dataTrain_ocz = (dataTrain(:,6))'; %wartosci oczekiwane zbioru treningowego (wyjscie)
100 -  dataTrain(:,6) = [];  %zbiór treningowy bez oczekiwanych wartości (wejscie)
101
102 -  dataTest_test = dataTest; %pełen zbiór testowy
103 -  dataTest_ocz = (dataTest(:,6))'; %wartosci oczekiwane zbioru testowego (wyjscie)
104 -  dataTest(:,6) = []; %zbiór testowy bez oczekiwanych wartości (wejscie)
```

Rysunek 3. Fragment kodu źródłowego - podział na dane testowe i treningowe

2.3 Funkcje aktywacji

W projekcie funkcją aktywacji warstwy ukrytej i warstwy wyjściowej będzie funkcja tangensa hiperbolicznego opisana poniższym wzorem:

$$f(x) = \tanh(x) \tag{2.4}$$

Założone funkcje aktywacji nie dały nam zadawalających wyników. Co spowodowało zmianę funkcji w obu przypadkach.

Poniżej została przedstawiona postać funkcji aktywacji tangensa hiperbolicznego (*tanh)* oraz jej pochodna w kodzie źródłowym (Rysunek 4).

```
182    %implementacja funkcji aktywacji wartstwy wyjściowej i ukrytej
183    %funkcja tangens hiperboliczny
184    function [y] = tangh(v)
185 -      y = tanh(v);
186 -  end
187
188    %pochodna
189    function [dtang] = difftanh(v)
190 -      dtang = 1 -tanh(v).*tanh(v);
191 -  end
```

Rysunek 4. Fragment kodu źródłowego – implementacja własna funkcji tanh oraz jej pochodnej

Przyjęte oznaczenia:
v – sygnał pobudzenia neuronowej warstwy ukrytej określony jako v^k *(2.5)* oraz warstwy wyjściowej v_m^k *(2.7).*
y – stan wyjściowy neuronów warstwy ukrytej określony jako ξ_l^k *(2.6)* oraz warstwy wyjściowejy_m^k *(2.8).*

Pochodna funkcji aktywacji warstwy ukrytej będzie wykorzystywana przy wyznaczeniu błędu sygnału $\delta_l^{(h)k}$*(2.10)* warstwy ukrytej oraz błędu sygnału $\delta_m^{(o)k}$ *(2.9)* warstwy wyjściowej.

Próg decyzyjności to 0.5. Przy innych wartościach progu sieci nie dawała zadawalających wyników. Oznacza to, że wartości wyższe lub równe 0.5 określają złośliwą zmianę (*y = 1)*, natomiast wartości mniejsze niż 0.5 – łagodna zmiana (*y = 0*).

2.4 Algorytm uczenia

Algorytmem wykorzystywanym do uczenia sieci będzie algorytm wstecznej propagacji błędów. Metoda ta umożliwia uczenie sieci wielowarstwowych poprzez propagację różnicy pomiędzy pożądanym a otrzymanym sygnałem na wyjściu sieci.

Algorytm:

1. Przyjęcie małych, losowych wartości wag $w_{l,n}$ oraz $\omega_{m,l}$.
2. Podanie na wejście sieci losowy wektor uczący $x^k = \left[x_1^k \;\; x_2^k \;\; \ldots \;\; x_N^k\right]^T$.
3. Obliczenie pobudzenia neuronów warstwy ukrytej, przy pomocy wzoru:

$$v^k = \sum_{n=1}^{N} w_{l,n} \cdot x_n^k \qquad (2.5)$$

4. Obliczenie stanu wyjść neuronów warstwy ukrytej:

$$\xi_l^k = f\left(v_l^k\right) \qquad (2.6)$$

5. Obliczenie pobudzenia neuronów warstwy wyjściowej, przy pomocy wzoru:

$$v_m^k = \sum_{l=1}^{L} \omega_{m,l} * \xi_l^k \qquad (2.7)$$

6. Obliczenie stanu wyjść neuronów warstwy wyjściowej:

$$y_m^k = f\left(v_m^k\right) \qquad (2.8)$$

7. Obliczenie sygnału błędy dla warstwy wyjściowej:

$$\delta_m^{(o)k} = \left(d_m^k - y_m^k\right) * f'(v_m^k) \qquad (2.9)$$

8. Obliczenie sygnału błędy dla warstwy ukrytej:

$$\delta_l^{(h)k} = f'(v_l^k) * \sum_{m=1}^{M} \left(\omega_{m,l} * \delta_m^{(0)k}\right) \tag{2.10}$$

9. Modyfikacja wagi warstwy wyjściowej:

$$\omega_{m,l}(t+1) = \omega_{m,l}(t) + \eta * \delta_m^{(o)k} * \xi_l^k \tag{2.11}$$

10. Modyfikacja wagi warstwy ukrytej:

$$w_{l,n}(t+1) = w_{l,n}(t) + \eta * \delta_l^{(h)k} * x_n^k \tag{2.12}$$

Wykonywany algorytm zostaje zatrzymany w wyniku osiągnięcia określonej ilości iteracji. Ilość iteracji zależy od ilości obserwacji zmian wartości błędu:

$$E = \frac{1}{2} \cdot \sum_{m=1}^{M} \left(d_m^k - y_m^k\right)^2 \tag{2.13}$$

W tym przypadku jest to *M*, które opisuje ilość wektorów uczących

```
194
195     %algorytm uczenia - propagacja wsteczna
196     function [W_ww, W_wu, E] = backProp(W_ww, W_wu, X, D)
197         alfa = 0.001; %współczynnik szybkości uczenia
198         T = size(X,1);
199         E = 0;
200         for i = 1:T
201             x = X(i,:)';
202             d = D(i);
203             v1 = W_ww*x; %obliczenie pobudzenia warstwy ukrytej
204             y1 = tangh(v1); %obliczenie stanu wyjścia wartswy ukrytej
205             v = W_wu*y1 ; %obliczenie pobudzenia warstwy wyjściowej
206             y = tanh(v); %obliczenie stanu wyjścia warstwy wyjściowej
207 
208             e = d - y; %różnica między wartością oczekiwaną, a uzyskaną y
209             delta = difftanh(v)*e;
210             e1 = W_wu'*delta;
211             delta1 = difftanh(v1).*e1;
212             dW_ww = alfa.*delta1*x';
213             W_ww = W_ww + dW_ww;
214             dW_wu = alfa*delta*y1';
215             W_wu = W_wu + dW_wu;
216             E = E + 0.5*e^2;
217         end
218         E = E/T;
219     end
```

Rysunek 5. Fragment kodu źródłowego - algorytm uczenia sieci

Przyjęte oznaczenia:
W_ww – macierz wag połączeń warstwy wejściowej
W_wu - macierz wag połączeń warstwy ukrytej
E – błąd określony wzorem (*2.13)*.
Wszystkie kolejne linijki powyższego kodu zostały napisane na podstawie wzorów *(2.5 – 2.12)*

2.5 Proces uczenia

Proces uczenia polega na wykorzystaniu funkcji propagacji wstecznej błędu (w kodzie źródłowym – funkcja *backProp*) w pętli, która zatrzymuje się w momencie osiągniecia określonej liczby iteracji. Im większa ilość powtórzeń tym lepiej. Należy mieć na względzie, że może dojść do przeuczenia sieci, dlatego ilość powtórzeń powinna być dobrana odpowiednio. Przed przystąpienie do procesu uczenia zainicjalizowano w losowy sposób wartości wektorów wag. Następnie wykonywana jest pętla z funkcją propagacji wstecznej. Po wykonaniu każdej iteracji zapisywana jest wartość błędu na wyjściu (2.13). Rysunek 7 pokazuje procesu uczenia, czyli implementacja wektorów wagowych oraz pętlę, która realizuje „trenowanie" sieci.

```
107 -    W_ww = 2*rand(3,5)-1;
108 -    W_wu = 2*rand(1,3)-1;
109 -    C = 100000;
110 -    blad_mse_train = 0;
111 -    Err = zeros(C,3);
112 -    Tre = size(dataTrain, 1);
113
114 -  for cykl = 1:C
115 -        [W_ww, W_wu, E] = backProp(W_ww, W_wu, dataTrain, dataTrain_ocz);
116 -        Err(cykl, 1) = cykl;
117 -        Err(cykl, 2) = E;
118 -        blad_mse_train = E*2/Tre;
119 -        Err(cykl, 3) = blad_mse_train;
120 -  end
```

Rysunek 6. Fragment kodu źródłowego - proces uczenia sieci

Kolejnym etapem jest sprawdzenie wytrenowanej sieci na danych testowych. Na wejściu podawane są dane testowe a na wyjściu otrzymujemy błąd sieci. Kod opiera się na progu decyzyjnym (opis w 2.3.2). Testowanie polega na wyznaczeniu kolejnych pobudzeń i stanów warstw ukrytych i wejściowych z pominięciem wyznaczania błędu warstwy ukrytej i aktualizacji wag.

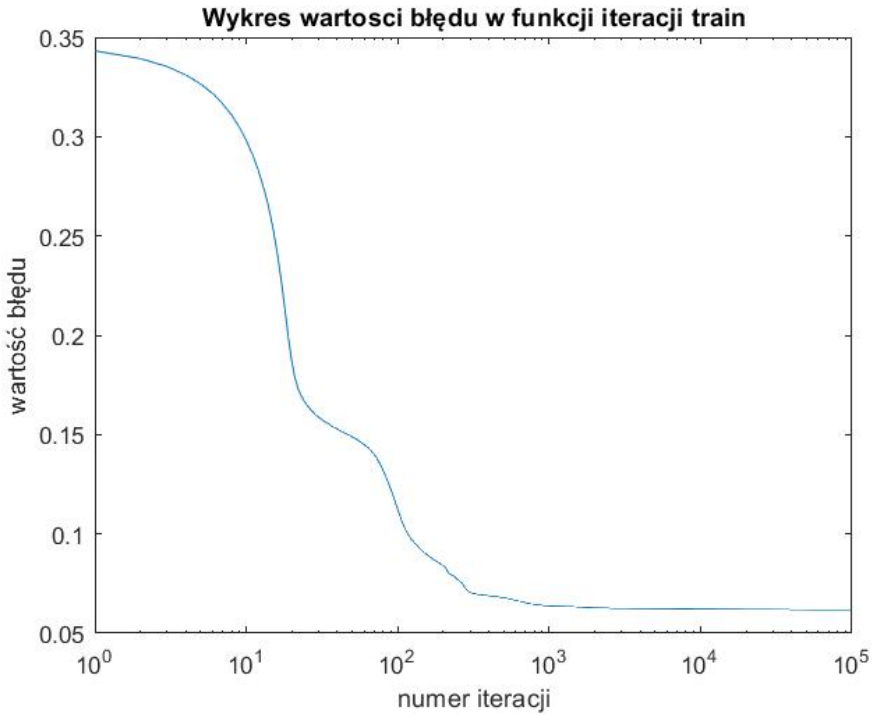
```
129 -   Test = size(dataTest_test,1);
130 -   blad_test = 0;
131 -     for q = 1:Test
132 -         dataTest1 = dataTest(q,:)';
133 -         x_test = W_ww*dataTest1;
134 -         y_test = tangh(x_test);
135 -         x_test1 = W_wu*y_test;
136 -         y_test1 = tangh(x_test1);
137
138 -         if y_test1 >= 0.5
139
140 -               y_test1 = 1;
141 -         else
142 -             y_test1 = 0;
143 -         end
144
145 -         Y_test(q,1) = y_test1;
146 -         e_test = dataTest_ocz(1,q)-y_test1;
147 -         blad_test = blad_test+0.5*e_test^2;
148 -     end
149
150 -   blad_mse_test1 = blad_test*2/Test;
151
```

Rysunek 7. Fragment kodu źródłowego - testowanie wytrenowanej sieci

3 Wyniki procesu uczenia sieci

3.1 Wykres błędu w zależności od liczby iteracji

Do wykreślenia wykresu błędu w zależności od liczby iteracji należało po każdym cyklu wyznaczyć błąd sieci wg wzoru 2.14. Dzięki temu mogłyśmy otrzymać poniższy wykres.



Rysunek 8. Przedstawienie zależności wartości błędu w funkcji iteracji

Na wykresie wartości błędu, wraz ze wzrostem liczby iteracji błąd znacząco spada. W okolicach 1000 iteracji wartość błędu stabilizuje się. W zależności od wylosowanego zbioru treningowego i testowego, przebieg wykresu może się nieznacznie różnić - błąd może się stabilizować nieco później np. w okolicach 10000 iteracji.

3.2 Testy opisujące użyteczność sieci

Do przeprowadzenia oceny użyteczności zaprojektowanej sieci wyznaczone został wskaźniki czułości (ang. *sensitivity*) i specyficzność (ang. *specificity*) metody diagnostycznej. Wymienione wskaźniki zostały opisane w kolejnych podpunktach.

Do wyznaczenia powyższych wskaźników należy określić współczynniki TN (*true negative* – prawdziwie negatywny, negatywny wynik testu u osoby zdrowej), TP (*true positive* – prawdziwie pozytywny, pozytywny wynik testu u osoby chorej), FN (*false negative* – fałszywie negatywny, negatywny wynik testu u osoby faktycznie chorej), FP (*false positive* – fałszywie pozytywny, pozytywny wynik testu u osoby chorej). Powyższe współczynniki zostały uwzględnione w kodzie (Rysunek 9)

```
151
152 -   TN = 0;
153 -   TP = 0;
154 -   FP = 0;
155 -   FN = 0;
156
157 -   for m=1:Test
158
159 -       if Y_test(m,1) == 0 && dataTest_ocz(m) == 0
160 -           TN = TN + 1;
161 -       elseif Y_test(m,1) == 1 && dataTest_ocz(m) == 1
162 -           TP = TP + 1;
163 -       elseif Y_test(m,1) == 0 && dataTest_ocz(m) == 1
164 -           FN = FN + 1;
165 -       else
166 -           FP = FP + 1;
167 -       end
168 -   end
169
170     %czułość
171
172 -   TPR = TP/(TP+FN);
173
174     %specyficzność
175
176 -   TNR = TN/(FP+TN);
```

Rysunek 9. Fragment kodu źródłowego - wyznaczenie czułości i specyficzności

3.2.1 Czułość

Czułość (ang. *sensitivity – true positive rate* TPR) to wskaźnik opisujący stosunek wyników prawdziwie pozytywnych (TP – *true positive)* do sumy wyników prawdziwych pozytywnych (TP) i fałszywych negatywnych (FN – *false negative*).

$$TPR = \frac{TP}{TP+FN} \tag{3.1}$$

Inaczej jest to prawdopodobieństwo uzyskania wyniku pozytywnego w teście rozpoznającym chorobę u prawdziwie chorej osoby.

3.2.2 Specyficzność

Specyficzność (ang. *specificity – true negative rate* TNR) opisuje stosunek wyników prawdziwie negatywnych (TN – *true negative)* do sumy wyników prawdziwie negatywnych (TN) i fałszywie pozytywnych (FP – *false positive*).

$$TNR = \frac{TN}{TN+FP} \tag{3.2}$$

Inaczej jest prawdopodobieństwo otrzymania negatywnego wyniku w teście rozpoznającym chorobę przez osobę prawdziwe zdrową.

3.2.3 Podsumowanie wyników

W poniższej tabeli zostały zestawione wyniki kolejnych 8 prób trenowania i testowania sieci. Za każdym razem wybierany został nowy, losowy zbiór treningowy i testowy. Na podstawie wyników zostały wyznaczone wartości średnie (Tab. 1).

Tab. 1 Przedstawienie wartości czułości i specyficzności

Numer próby	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Średnia**
Specyficzność	0.8627	0.9400	0.8125	0.8125	0.8696	0.9206	0.9483	0.7826	0.8686
Czułość	0.8667	0.7826	0.7500	0.8125	0.8000	0.7273	0.8421	0.8400	0.8027

Analiza uzyskanych wyników pozwala stwierdzić, iż czułość i specyficzność osiągają zadowalającą wartość. System diagnostyczny oparty na utworzonej sieci neuronowej jest użyteczny. Wyniki przez niego osiągane nie są najwyższe, niemniej jednak są zadowalające. Zdecydowanie lepiej system klasyfikuje rozpoznanie osoby z łagodną zmianą (86,86%), natomiast gorzej klasyfikuje osoby ze złośliwą zmianą (80,27%).

4 Bibliografia

1. UCI Machine Learning Repository: Mammographic Mass Data Set
2. prezentacje do wykładu SNB - Sieci neuronowe w zastosowaniach biomedycznych

5 Oświadczenie

Oświadczam, że niniejsza praca stanowiąca podstawę do uznania osiągnięcia efektów uczenia się z przedmiotu *Sieci neuronowe w zastosowaniach biomedycznych (SNB)* została wykonana przeze mnie samodzielnie.

Adamczyk Jagoda, 289214
Krakowiak Aleksandra, 290292