Wstęp:

Sieci neuronowe są pojęciem bardzo szerokim, tak iż powstało wiele różnych architektur inspirowanych biologicznymi sieciami neuronowymi [161]. Łączy je jednak wspólny element jakim jest model podstawowej jednostki obliczeniowej zwany neuronem, który odpowiednio powielony tworzy architekturę sieci. Pojedyncze neuron implementowany jest w postaci funkcji zwanej funkcją aktywacji $y = f(\mathbf{x},\mathbf{w})$, która odwzorowuje wektor $\mathbf{x}$; $8x\ 2\ <$ na wartość $y\ 2\ <$, gdzie $\mathbf{w}$ jest zbiorem parametrów poddawanych adaptacji w trakcie procesu uczenia [78]. W problemach predykcyjnych typu klasyfikacja i regresja najpopularniejszym typem sieci jest perceptron wielowarstwowy (MLP, ang. Multi Layer Perceptron), lub jego pochodne [113]. Dla sieci MLP funkcja aktywacji neuronu przyjmuje postać modelu liniowego z dodatkową nieliniową funkcją wyjściową $g(\cdot)$, co można zapisać jako:

$$y = f(\mathbf{x}|\mathbf{w}) = g\left(\sum_{i=1}^{m} w_i x_i + w_0\right)$$

Nieliniowa funkcja $g(\cdot)$ w pierwszych wersjach algorytmu przybierała postać funkcji binarnej

$$y = g(z) = \begin{cases} 1 \text{ dla } z \geq 0 \\ -1 \text{ dla } z < 0 \end{cases}$$

jednak z uwagi na problem z nieciągłością binarną funkcję $g(\cdot)$ zastąpiono funkcją sigmoidalną typu

$$y = g(z) = \frac{1}{1 + \exp(-\beta z)}$$

a obecnie coraz popularniejsze są funkcje wywodzące się ze skorygowanej funkcji liniowej (ReLU, ang. Rectified Linear Unit)

$$y = g(z) = max(0, z)$$

która ogranicza problem tak zwanego zanikającego gradientu [146].

Architektura sieci MLP sprowadza się do sekwencyjnego połączenia grup neuronów zwanych warstwami, przykładową postać sieci MLP przedstawiono na Rys.2.4. W sieci tej transmisja sygnału odbywa się w jednym kierunku tj. od tak zwanej warstwy wejściowej (pierwszej grupy neuronów) odpowiadającej wartościom wektora $\mathbf{x}$ do warstwy wyjściowej, z której dekodowana jest odpowiedź sieci np. w problemach klasyfikacyjnych odpowiedzi sieci zamieniane są na etykietę klasy.

Uczenie sieci polega na adaptacji parametrów $\mathbf{w}$ każdego z neuronów, a proces ten odbywa się w oparciu o algorytm gradientowy [204], tak że

$$\mathbf{w}^{\{k+1\}} = \mathbf{w}^{\{k\}} + \Delta\mathbf{w}^{\{k\}}$$

gdzie

$$\Delta\mathbf{w}^{\{k\}} = -\alpha\frac{\partial E}{\partial\mathbf{w}^{\{k\}}}$$

gdzie E jest funkcją kosztu (zwykle $E = (y - y')^2$), dla której y – to rzeczywista wartość wyjściowa, natomiast y' to wartość oszacowana przez sieć oraz $\alpha > 0$ jest współczynnikiem uczenia. Indeks górny k oznacza wartość wagi $\mathbf{w}$ w k' tej iteracji. Wartość gradientu dla poszczególnych warstw wyznaczana jest zgodnie z tak zwanym algorytmem wstecznej propagacji. Dodatkowo często stosowaną poprawką klasycznego algorytmu wstecznej propoagacji jest wykorzystanie tak zwanego czynnika momentum [220], co sprowadza się do aktualizacji wag neuronu w oparciu o zależność:

$$\Delta\mathbf{w}^{\{k\}} = -\alpha\frac{\partial E}{\partial\mathbf{w}^{\{k\}}} + \beta\Delta\mathbf{w}^{\{k-1\}}$$

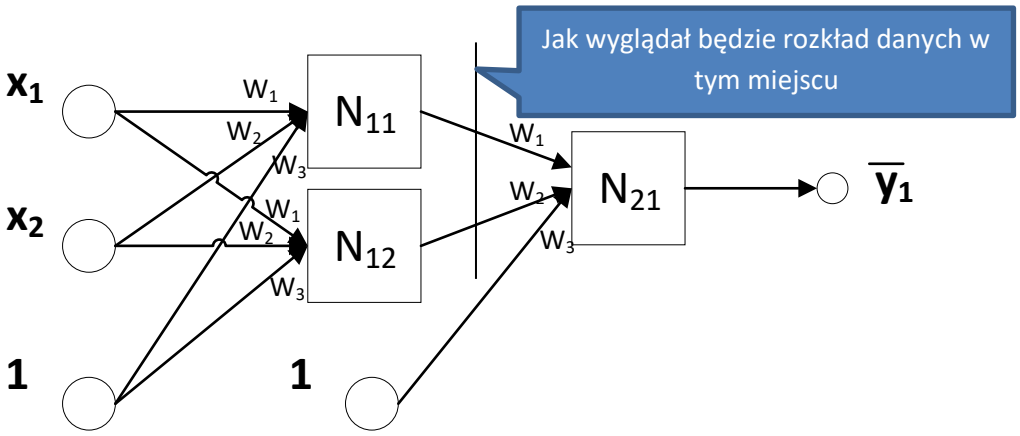
gdzie β określa wagę czynnika momentum, czyli wpływ poprzedniej wielkości aktualizacji wag. W efekcie wykorzystania wstecznej propagacji z czynnikiem momentum uzyskujemy szybszą zbieżność algorytmu. Jednym z problemów związanych z rozwojem sieci neuronowych był problem zanikającego gradientu w

przypadku sieci o większej liczbie warstw niż 3, który objawiał się zmniejszeniem wartości $\triangle \mathbf{w}$ wraz z analizą głębszych warstw sieci. Jedną z metod rozwiązania tego problemu jest zastąpienie sigmoidalnej funkcji nieliniowej neuronu poprzez funkcję ReLU. Powoduje to ograniczenie tempa spadku gradientu, jednak czyni sieć bardziej czułą na wartości odstające [102]. W przedstawionych poniżej badaniach wykorzystywano sieci z jedną lub dwoma warstwami ukrytymi z wykorzystaniem sigmoidalnej funkcji $g(\cdot)$, a do uczenia wykorzystano algorytm wstecznej propagacji z czynnikiem momentum.

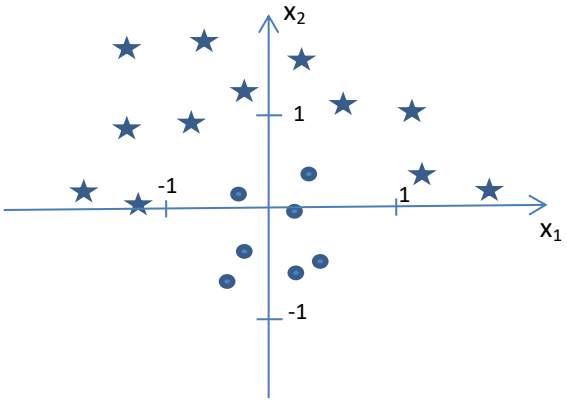
Zadania

1. Znaczenie funkcji nieliniowej w neuronie
   Co się stanie jeśli dla sieci z zadania 2 neurony będą miały liniową funkcję aktywacji f(**x**)=w1*x1+w2*x2+w3. Jaki kształt będzie miała granica decyzji
2. Przyjmując, że poniższa sieć neuronowa ma neurony z bipolarną funkcją aktywacji
   $f(x) = \begin{cases} 1, x > 0 \\ -1, x \leq 0 \end{cases}$
   Nanieś na wykres przedstawiający dane wejściowe położenie neuronów, oraz pokaż jak będzie wyglądał rozkład danych na wyjściu warstwy ukrytej

	N12	N22	N21
W1	1	1	−1
W2	−1	1	1
W3	1	−1	−1

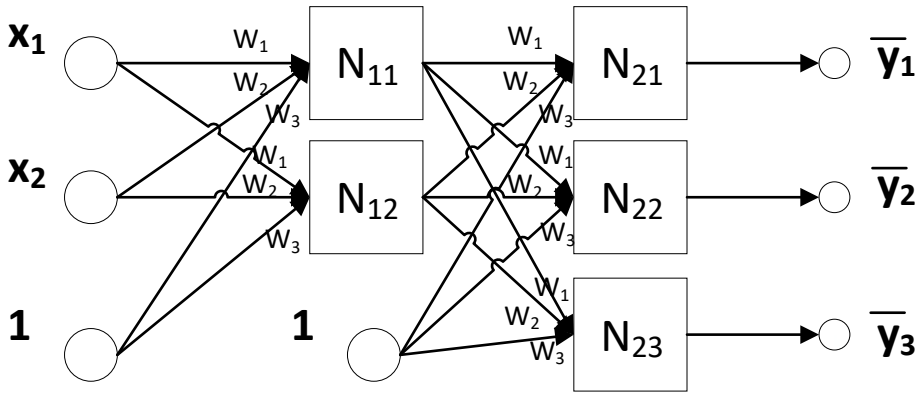


Rozkład danych przedstawiono na rys. poniżej.



3. Dokonywanie predykcji przez sieć

Na rys. poniżej przedstawiono schemat sieci neuronowej. Wartości wag dla poszczególnych neuronów podano w tabeli. Dokonaj klasyfikacji obiektów, których zmienne x1 i x2 podano w tabeli 2, wiedząc że należą one do jednej z trzech klas K1, K2, K3 wiedząc że każdej z klas odpowiada jeden neuron wyjściowy. Przyjmuj że funkcja aktywacji neuronów ma postać funkcji sigmoidalnej o postaci $f(x) = \frac{1}{1+\exp(-x)}$



Wagi między neuronami	N11	N12	N21	N22	N23
W1	5.466	−4.199	−4.298	−6.465	6.753
W2	5.564	−3.902	6.344	−6.442	−4.544
W3	−3.292	−3.758	−2.922	3.102	−3.231

X1	X2	Y	Nr neuronu
0.542	0.583		
0.085	0		

4. Ile neuronów ma sieć z zadania 3, ile neuronów znajduje się w warstwie/warstwach ukrytych, ile neuronów znajduje się w warstwie wyjściowej

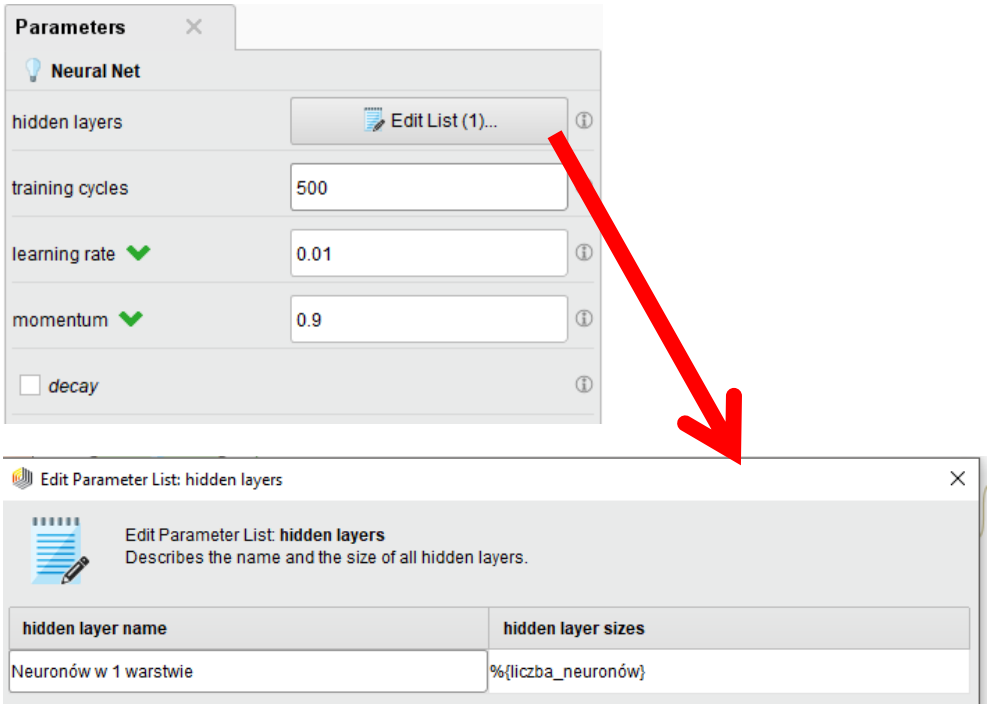
W RapidMiner

5. Wczytaj zbiór danych data_q_train.csv. Jak myślisz ile neuronów będzie potrzebne do poprawnej klasyfikacji tego zbioru.
6. Sprawdź podaną liczbę stosując do oceny nauczonej sieci dane *data_q_test.csv* oraz mierząc dokładność predykcji za pomocą operatora *Performance*. Uwaga, jako sieć neuronową wykorzystaj operator *Neural Net* i ustaw w nim liczbę iteracji na 500
7. Spróbuj zwiększać liczbę neuronów co 2 do 10 i pokaż zależność liczba neuronów w funkcji dokładności. Jaką liczbę neuronów użyłbyś w praktyce. Uzasadnij wybór.

8. Powtórz operacje wykorzystując zbiór *dataTrain_GL.csv i dataTest_GL.csv* jednak zmieniaj liczbę neuronów co 2 do 20 neuronów oraz ustaw liczbę cykli uczenia na 1500. Pokaż zależność liczby neuronów w funkcji dokładności.
9. Wybierz najlepszą liczbę neuronów z zadania 8 i sprawdź jak wpływa liczba iteracji na dokładność sieci. Sprawdzenia dokonaj w zakresie od 100 do 6000 iteracji (wartości zmieniaj w postępie logarytmicznym)
   Na wykresie pokaż zależność dokładność predykcji w funkcji liczby iteracji oraz czas uczenia w funkcji liczby iteracji.
   Przy jakiej liczbie iteracji sieć uzyskuje dobre wyniki?

Uwagi

Dla ułatwienia realizacji zadań wykorzystaj operator Loop Parameters oraz Log. W operatorze Loop Parameters zmieniaj odpowiednio liczbę cykli uczenia oraz liczbę neuronów. Do zmiany liczby neuronów wykorzystaj zmienne czyli *makra*. W tym celu wewnątrz operatora Loop Parameters dodaj operator *Set Macro*, w którym zostanie stworzone makro np. o nazwie „liczba_neuronów". Wartość makra ustawiaj za pomocą Loop Parameters modyfikując „value" dla operatora Set Macro. Zmienną liczba neuronów wykorzystaj do ustawienia liczby neuronów w operatorze Neural Net jak pokazano na rys.



Jeśli nie zmieniasz liczby neuronów to ustaw porządaną wartość w operatorze *Set Macro*

Podczas obliczeń loguj liczbę iteracji, liczbę neuronów, dokładność oraz czasu uczenia i czas predykcji