Metody Sztucznej Inteligencji w Sterowaniu

Ćwiczenie 2

Sztuczne siei neuronowe - wprowadzenie

Przygotował: mgr inż. Marcin Pelic
Instytut Technologii Mechanicznej
Politechnika Poznańska

Poznań, 2011

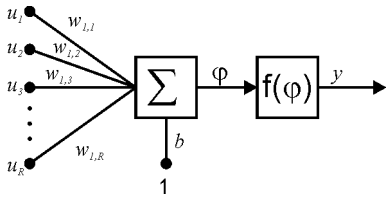
Wstęp teoretyczny

1. Model sztucznego neuronu

Podobnie jak w przypadku neuronowych sieci biologicznych, podstawowymi elementami z których buduje się sztuczne sieci neuronowe są sztuczne neurony. Sztuczny neuron jest elementem, którego własności odpowiadają wybranym własnościom neuronu biologicznego. Z założenia nie jest więc on jego wierną kopią, lecz elementem, który powinien spełniać określone funkcje w sztucznej sieci neuronowej.

Ogólnie sztuczny neuron można rozpatrywać jako specyficzny przetwornik sygnałów działający według następującej zasady: na wejście przetwornika doprowadzone są sygnały wejściowe, które następnie są mnożone przez odpowiednie współczynniki wag, ważone sygnały wejściowe są następnie sumowane i na tej podstawie wyznacza się aktywność neuronu.

Na rys. 1 przedstawiono model sztucznego neuronu. Składa się on z dwóch bloków: bloku sumowania $\sum$ i bloku aktywacji f(φ).



Rys. 1. Model neuronu

W bloku sumowania wykonywane jest algebraiczne sumowanie ważonych sygnałów wejściowych, oraz generowany jest sygnał wyjściowy φ:

$$\varphi = \sum_{i=1}^{R} w_{1,i} u_i + b = \mathbf{w}^T \mathbf{u} + b \tag{1}$$

gdzie: w – wektor współczynników wag $w_{1,i}$, u – wektor sygnałów wejściowych u_i, R – liczba wejść neuronu, b – próg (bias).

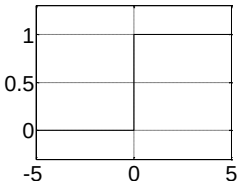
Sygnał φ poddawany jest przetwarzaniu przez blok aktywacji $f(\varphi)$ realizujący zależność $y = f(\varphi)$. Ostatecznie sygnał wyjściowy ma postać:

$$y = f(\varphi) = f\left(\sum_{i=1}^{R} w_{1,i} u_i + b \right) = f\left(\mathbf{w}^T \mathbf{u} + b \right) \tag{2}$$

Funkcja aktywacji, w zależności od konkretnego celu, jakiemu służy neuron, może przyjmować różne postacie. Niektóre z nich to:

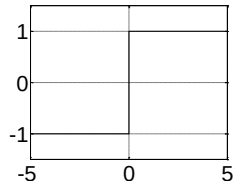
- funkcja skokowa unipolarna (funkcja Heaviside'a)

$$f(x) = 1(x) = \begin{cases} 1 & \text{jeśli } x > 0 \\ 0 & \text{jeśli } x \le 0 \end{cases} \tag{3}$$



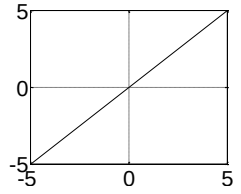
- funkcja skokowa bipolarna

$$f(x) = 1(x) = \begin{cases} 1 & \text{jeśli } x > 0 \\ -1 & \text{jeśli } x \le 0 \end{cases} \tag{4}$$



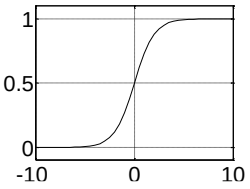
- funkcja liniowa

$$f(x) = ax \tag{5}$$



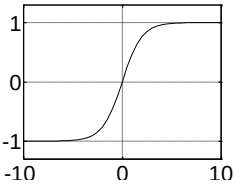
- funkcja sigmoidalna unipolarna

$$f(x)=\frac{1}{1+e^{-\beta x}} \tag{6}$$



- funkcja sigmoidalna bipolarna (tangensoidalna)

$$f(x)=\tanh(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \quad \text{lub} \quad f(x)=\tanh\left(\frac{x}{2}\right)=\frac{1-e^{-x}}{1+e^{-x}}$$

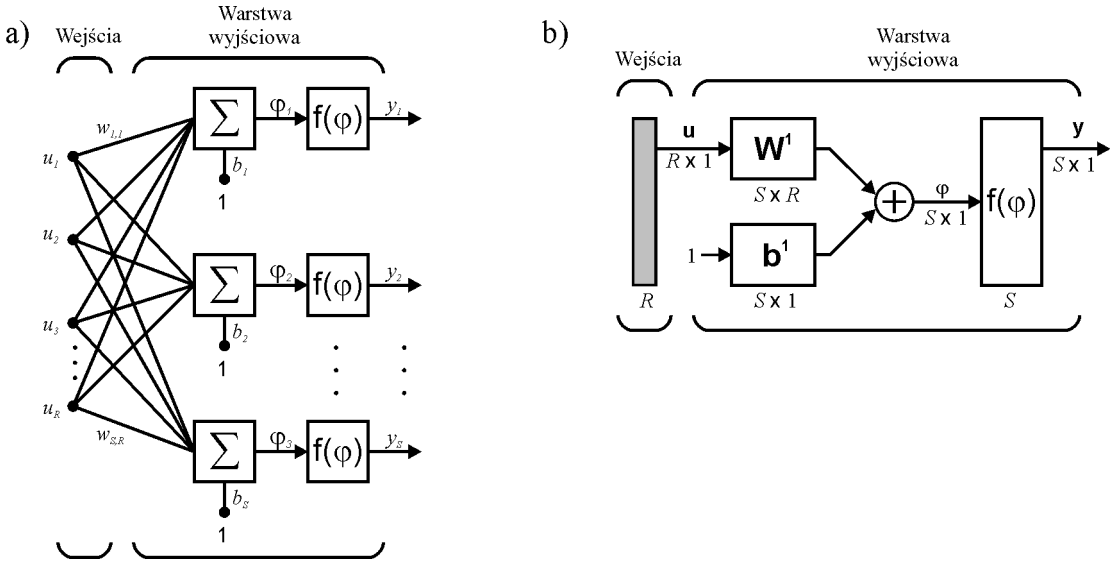


2. Sieci jednokierunkowe

Neurony połączone między sobą tworzą układ nazywany sztuczną siecią neuronową (w skrócie siecią neuronową). W zależności od sposobu połączenia neuronów można wyróżnić sieci jednokierunkowe lub rekurencyjne (ze sprzężeniem zwrotnym).

Sieć neuronowa jednokierunkowa jest złożona z neuronów ułożonych w warstwy o jednym kierunku przepływy sygnałów. Połączenia międzywarstwowe występują jedynie między sąsiednimi warstwami.

Najprostszą siecią neuronową jest sieć jednowarstwowa. Tworzą ją neurony ułożone w jednej warstwie (rys. 2a, 2b). Każdy neuron posiada próg (bias) b_i oraz wiele wag w_{ij} prowadzonych do sygnałów wejściowych u_j. Neurony ułożone w pojedynczej warstwie działają niezależnie od siebie, stąd możliwości takiej sieci są ograniczone do możliwości pojedynczych neuronów.



Rys. 2. Jednowarstwowa sieć neuronowa o R wejściach i S wyjściach: a) schemat pełny, b) schemat uproszczony

Każdy neuron realizuje odwzorowanie funkcyjne:

$$y_i = f\left(\sum_{j=1}^{R} w_{ij}u_j + b_i\right) \tag{8}$$

gdzie: R – liczba wejść, y_i – i-te wyjście, w_{ij} – waga dla i-tego neuronu i j-tego wejścia.

Powyższe równanie można zapisać również w zwięzłej postaci macierzowej:

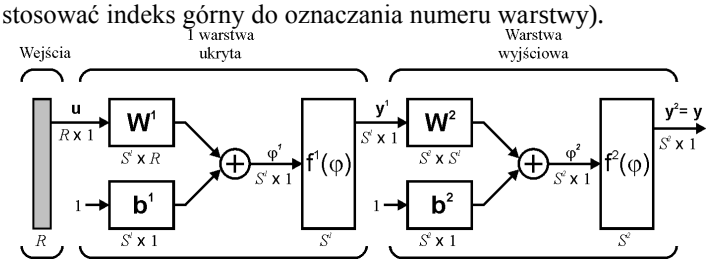
$$\mathbf{y} = f\left(\mathbf{W}\mathbf{u} + \mathbf{b}\right) \tag{9}$$

gdzie: u – wektor wejścia, y – wektor wyjścia, W – macierz wag.

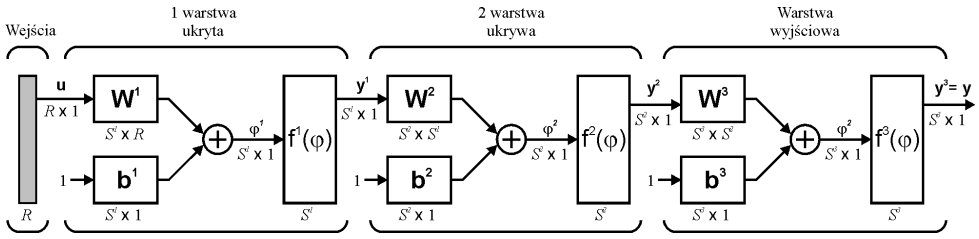
Sieć jednowarstwowa ma niewielkie znaczenie praktyczne, jakkolwiek stosuje się ją nadal tam, gdzie istnienie jednej warstwy jest wystarczające do rozwiązania określonego problemu.

Sieć wielowarstwową tworzą neurony ułożone w wielu warstwach, przy czym oprócz wejść i warstwy wyjściowej istnieje co najmniej jedna warstwa ukryta.

Na rys. 3 przedstawiono sieć o jednej warstwie ukrytej, a na rys. 4 sieć o dwóch warstwach ukrytych (w oznaczeniach przyjęto stosować indeks górny do oznaczania numeru warstwy).



Rys. 3. Sieć dwuwarstwowa



Rys. 4. Sieć trójwarstwowa

Sieć dwuwarstwowa realizuje następujące odwzorowanie wektora wejściowego u na wektor wyjściowy y:

$$\mathbf{y} = f^2\left(\mathbf{W}^2\mathbf{y}^1 + \mathbf{b}^2\right) = f^2\left(\mathbf{W}^2 f^1\left(\mathbf{W}^1\mathbf{u} + \mathbf{b}^1\right) + \mathbf{b}^2\right), \tag{10}$$

lub dla k-tego wyjścia:

$$y_k = f^2\left(\sum_{i=1}^{S^1} w_{ki}^2 y_i^1 + b_k\right) = f^2\left(\sum_{i=1}^{S^1} w_{ki}^2 f^1\left(\sum_{j=1}^{R} w_{ij}^1 f\left(u_j\right) + b_i\right) + b_k\right) \tag{11}$$

gdzie: R – liczba wejść, S^1 – liczba neuronów w 1-szej warstwie ukrytej, y_k – k-te wyjście, w_{ij}, w_{ki} – wagi, b_k, b_i – progi.

Sieci neuronowe wykorzystujące ciągłe funkcje aktywacji mają ciągłe charakterystyki. Pozwala to na bezpośrednie zastosowanie algorytmów gradientowych do uczenia takich sieci (uczenie polega na doborze wartości wag według określonego algorytmu, które umożliwi dostosowanie działania sieci do warunków środowiskowych określonych w postaci określonych wymagań co do odwzorowania danych wejściowych na wyjściowe).

Sieci z funkcjami liniowymi mają nieograniczony zakres wartości wyjściowej, ale realizują tylko odwzorowanie liniowe. Z kolei sieci zawierające funkcje sigmoidalne mogą tworzyć dowolne odwzorowanie nieliniowe o ograniczonym zakresie wyjściowym. Aby połączyć zalety obu tych sieci – zdolność realizowania nieliniowych odwzorowań i nieograniczoność zakresu wyjściowego – należy w warstwach ukrytych zastosować sigmoidalne funkcje aktywacji, natomiast w warstwie wyjściowej – liniowe.

Skokowe funkcje aktywacji przyjmuje się w tego typu systemach, gdzie sygnał wyjściowy powinien przyjmować jedną z dwóch wartości dyskretnych. W tym przypadku algorytmy gradientowe, uznawane za najskuteczniejsze w uczeniu, nie mogą mieć zastosowania, gdyż podstawowe wymaganie dotyczące funkcji celu nie jest spełnione.

Opis zastosowanych funkcji

newp – tworzy nową sieć perceptronową

> ***NET = NEWP(PR,S,TF,LF)***
> ***PR*** – macierz określająca zakres wartości wejść sieci (liczba wejść jest określana na podstawie rozmiaru tej macierzy)
> ***S*** – liczba neuronów
> ***TF*** – Transfer Function - funkcja wyjścia, domyślnie '***hardlim***'
> ***LF*** – Learning Function - funkcja ucząca, domyślnie '***learnp***'
> Funkcja zwraca: nowy perceptron o nazwie ***NET***
> Więcej informacji***: help newp***
>
> Przykład tworzenie perceptronu z dwoma wejściami z których pierwsze przyjmuje sygnały z zakresu <-15; 15> natomiast drugie wejście przyjmuje parametry z zakresu <1;10>
>
> >> ***net = newp([-15 15; 1 10], 1);***

plotpv – zaznacza na prostej, płaszczyźnie lub w przestrzeni współrzędne z wektorów uczących dla perceptronu dla sieci perceptronowej z jednym, dwoma i trzema wejściami. Punkty o wartości 1 zaznacza „x", punkty o wartości 0 zaznacza w postaci „o".

> ***plotpv (P, T, V)***
> ***P*** – wektor wartości wejściowych, gdzie ilość wierszy wektora jest równa ilości wejść sieci perceptronowej
> ***T*** – wektor odpowiedzi w postaci 0 lub 1 (prawda, fałsz)
> ***V*** – limit osi wyświetlanej przestrzeni w postaci wektora [x_min x_max y_min y_max z_min z_max] – parametr nie jest wymagany
> ***UWAGA! Długość wektorów P i T musi być taka sama!***
> Więcej informacji***: help plotpv***
>
> Przykład wykorzystania funkcji *plotpv* dla współrzędnych punktów leżących na płaszczyźnie P1(1,1), P2(1,5), P3(13, -4), P4(-12,3), oraz wartości przypisanych dla punktów: P1 prawda, P2 fałsz, P3 fałsz, P4 prawda.
>
> >> ***wejścia = [1 1 13 -13; 1 5 -4 3]***
> >> ***wyjście = [1 0 0 1]***
> >> ***plotpv (wejścia, wyjście)***

plotpc – zaznacza punkt, prostą lub płaszczyznę wyznaczoną na podstawie wag oraz biasów sieci perceptronowej. Dla prawidłowo nauczonej sieci punkt, prosta lub płaszczyzna rozdziela zbiór rozwiązań na dwie części – pierwsza gdzie wyniki są fałszem (0), druga, gdzie wyniki są prawdą (1) .

> ***plotpc (W, b)***
> ***W*** – wektor wag wejściowych sieci perceptronowej (patrz wyświetlanie wag)
> ***b*** – wektor biasów sieci perceptronowej (patrz wyświetlanie biasów)
> ***UWAGA! plotpc stosuje się po wcześniejszym wywołaniu funkcji plotpv***
> Więcej informacji***: help plotpc***
>
> Przykład zaznaczenia płaszczyzny wyznaczonej przez wektory wag i biasów trójwejściowej sieci perceptronowej o nazwie „SiecPerceptronowa" (zastosowanie jest analogiczne dla sieci dwu i jednowejściowej)
>
> >>***plotpc (SiecPerceptronowa.IW{1}, SiecPerceptronowa.b{1})***

newff – tworzy sieć neuronową propagacji wstecznej

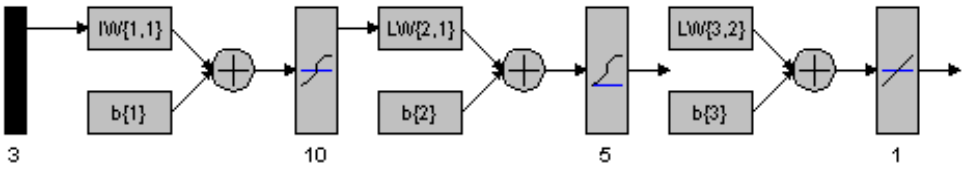
> ***NET = NEWFF(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)***
> ***PR*** - macierz określająca zakres wartości wejść sieci (liczba wejść jest określana na podstawie rozmiaru tej macierzy),

Si - liczba neuronów w i-tej warstwie, N1 – liczba warstw,
TFi – funkcja wyjścia neuronów w i-tej warstwie, domyślnie 'tansig'.
BTF – funkcja ucząca sieć, domyślnie 'trainlm'.
BLF – funkcja ucząca wagi/bias, domyślnie 'learngdm'.
PF - funkcja błędu, domyślnie 'mse'.
Zwraca N warstwową sieć neuronową propagacji wstecznej o nazwie ***NET.***
Więcej informacji: ***help newff.***

Przykład tworzenia nowej sieci neuronowej o nazwie „NowaSiec" propagacji wstecznej z trzema wejściami o zakresach X1∈<-10;10>, X2∈<-5;2>, X3∈<0;1> i złożoną z 3 warstw:
- warstwa 1 – 10 neuronów funkcja aktywacji '*tansig*',
- warstwa 2 – 5 neuronów funkcja aktywacji '*logsig*',
- warstwa 3 – 1 neuron funkcja aktywacji '*purelin*'.

>> ***NowaSiec = newff([-10 10; -5 2; 0 1],[10 5 1],{ 'tansig' 'logsig' 'purelin'})***

Struktura sieci została przedstawiona na rysunku 5



Rys. 5. Struktura sieci neuronowej z przykładu tworzenia sieci za pomocą polecenia newff

init – inicjacja lub reinicjacja (ponowne przywrócenie wartości początkowej) sieci neuronowej – zerowanie wag i biasów wszystkich warstw

NET =init(NET)
NET – Nazwa sieci neuronowej do zainicjowania
Więcej informacji: ***help network/init.***

Przykład inicjowania sieci neuronowej o nazwie „*MojaSiec*"

>> ***MojaSiec=init(MojaSiec)***

train – rozpoczęcie procedury treningu sieci neuronowej zgodnie z parametrami treningu

[net,tr,Y,E,Pf,Af] = train(NET,P,T,Pi,Ai,VV,TV)
NET – nazwa sieci wejściowej
P – wektor danych wejściowych
T – wektor danych wyjściowych
Pi, Ai , VV, TV – dane nieobowiązkowe
net – nowa, nauczona sieć
TR – wynik treningu
Y – eektor wyjść sieci
E – błędy sieci
Pf , Af – dane nieobowiązkowe
Więcej informacji ***help network/train***.

Przykład polecenia rozpoczynającego trening sieci „SiecNeuronowa" z wektorami uczącymi „StanyWejsc" i „StanyWyjsc"

>>***SiecNeuronowa = train (SiecNeuronowa, StanyWejsc, StanyWyjsc)***

epochs – zmienna pozwalająca ustawić ilość epok podczas nauki sieci

NET.trainParam.epochs=X
NET – Nazwa sieci neuronowej w której zmieniana będzie liczba epok
X – nowa liczba epok dla sieci ***NET***

Przykład zmiany liczby epok sieci „*NowaSiec*" na 100

>> ***NowaSiec.trainParam.epochs=100;***

sim – oblicza odpowiedź sieci neuronowej na dany wektor wejściowy

[Y,Pf,Af,E,perf] = SIM(NET,P,Pi,Ai,T)
Parametry:
NET – Nazwa sieci neuronowej
P – Wektor wejściowy.
Pi, Ai,T – parametry nieobowiązkowe
Funkcja zwraca:
Y – Wektor odpowiedzi sieci.
Więcej informacji: ***help network/sim.m.***

Przykład symulacji odpowiedzi sieci neuronowej „MojaNowaSiec" na wektor T

>> ***Y = SIM (MojaNowaSiec, T)***

Wyświetlanie wektorów wag wejść, wag warstw oraz biasów

NET.IW{N} wyświetlanie wektora wag wejściowych

NET – nazwa sieci neuronowej
IW – Nazwa wektora wag warstwy wejściowej (*Input Weight*)
N – numer komórki wektora wag

Przykład wyświetlenia wektora wag sieci o nazwie „NowaSiecNeuronowa"

>> ***NowaSiecNeuronowa.IW{1}***

NET.LW{N} wyświetlanie wektora wag wejściowych

NET – nazwa sieci neuronowej
LW – Nazwa wektora wag warstwy wejściowej (*Layer Weight*)
N – numer lub adres komórki wektora wag

Przykład wyświetlenia wektora wag warstwy 2 sieci o nazwie „NowaSiecNeuronowa"

>> ***NowaSiecNeuronowa.LW{2,1}***

NET.B{N} wyświetlanie wektora wag wejściowych

NET – nazwa sieci neuronowej
IW – Nazwa wektora wag warstwy wejściowej (*Input Weight*)
B – numer komórki wektora biasów

Przykład wyświetlenia wektora biasów drugiej warstwy sieci o nazwie „NowaSiecNeuronowa"

>> ***NowaSiecNeuronowa.B{2}***

Procedura tworzenia i wykorzystania sieci neuronowych

1) Przygotowanie wektorów uczących (wektor wejść i wektor wyjść)
2) Utworzenie sieci neuronowe o odpowiedniej strukturze
3) Ustawienie parametrów uczenia się sieci neuronowej (maksymalnej liczby epok, akceptowalnego błędu końcowego etc.)
4) Uczenie sieci za pomocą wcześniej przygotowanych wektorów uczących
5) Testowanie wiedzy sieci
6) Przygotowanie danych wejściowych do symulacji z nauczoną siecią neuronową (wektor wejściowy nie musi być tożsamy z wektorem uczącym!)
7) Symulacja wyników z wykorzystaniem sieci neuronowej

Przykład I)

Utworzenie perceptronu działającego jak trójwejściowa bramka logiczna AND o nazwie „*ANDnn*" Wartości wejściowe X1, X2 i X3 do bramki zapisano w postaci wektora o nazwie „*Wejścia*" gdzie każdy wiersz reprezentuje kolejne wejście. Wektor „Wyjście" reprezentuje wyjście z bramki i został zapisany w postaci wektora o jednym wierszu (ad. 1).

```
>> Wejscia= [ 0 0 0 0 1 1 1 1;
             0 0 1 1 0 0 1 1;
             0 1 0 1 0 1 0 1]
>> Wyjscie=[ 0 0 0 0 0 0 0 1]
```

Stworzenie nowej sieci perceptronowej, trójwejściowej z wejściami z zakresu <0;1> o nazwie „*ANDnn*" (ad. 2).

```
>> ANDnn= newp([0 1; 0 1; 0 1],1)
```

Ustawienie maksymalnej ilości epok =10 (ad. 3).

```
>> ANDnn.TrainParam.Epochs=10
```

Uczenie sieci (ad. 4).

```
>> ANDnn=train(ANDnn, Wejscia, Wyjscia)
```

Testowanie sieci (ad. 5).

```
>> plotpv (Wejscia, Wyjscia)
>> plotpc (ANDnn.IW{1}, ANDnn.b{1})
```

Do symulacji zostanie wykorzystany wektor „*Wejscia*" ponieważ zawiera wszystkie kombinacje logiczne wejść. (ad. 6). Symulacja z wykorzystaniem polecenia ***sim*** (ad. 7).

```
>> Y = sim(ANDnn, Wejscia)
```

Przykład II)

Nauczyć sieć neuronową o nazwie „SIEC" naśladować funkcję $y=2x^2+3x-30$ w zakresie <-10; 10>

Tworzenie wektorów uczących (ad. 1)

```
>> X= [-10:10]
>> Y = 2*X.^2+3*X-30
```

Tworzenie sieci neuronowej o nazwie SIEC, zakresie wejść <-10;10>, dziesięciu neuronach w warstwie wejściowej i jeden w warstwie wyjściowej z funkcjami aktywacji odpowiednio 'tansig' oraz 'purelin' (ad. 2)

>> ***SIEC=newff([-10 10], [10 1],{'tansig' 'purelin'})***

Ustawienie 100 epok dla sieci „SIEC" (ad. 3)

>>***SIEC.trainparam.epochs=100***

Trenowanie sieci (ad. 4)

>>***SIEC=train(SIEC, X, Y)***

Testowanie sieci (ad. 5) – sprawdzenie wartości funkcji w punkcie np. x=0

>> ***wynikTestu=sim(SIEC, 0)***

Symulacja z wykorzystaniem funkcji *sim* i nowego wektora wejściowego *nowyX* (ad. 6 i 7). Wynik zostanie pokazany na wykresie w postaci dwóch krzywych – czerwona –funkcja obliczona algebraicznie, niebieska – funkcja jako odpowiedź z sieci neuronowej na nowy wektor wejściowy *nowyX*

>> ***nowyX=-10:0.01:10***
>> ***nowyY= 2*X.^2+3*X-30***
>> ***SymulacjaY=sim(SIEC, nowyX)***
>> ***plot (nowyX, nowY,'r'); grid; hold; plot(nowyX, SymulacjaY, 'b')***