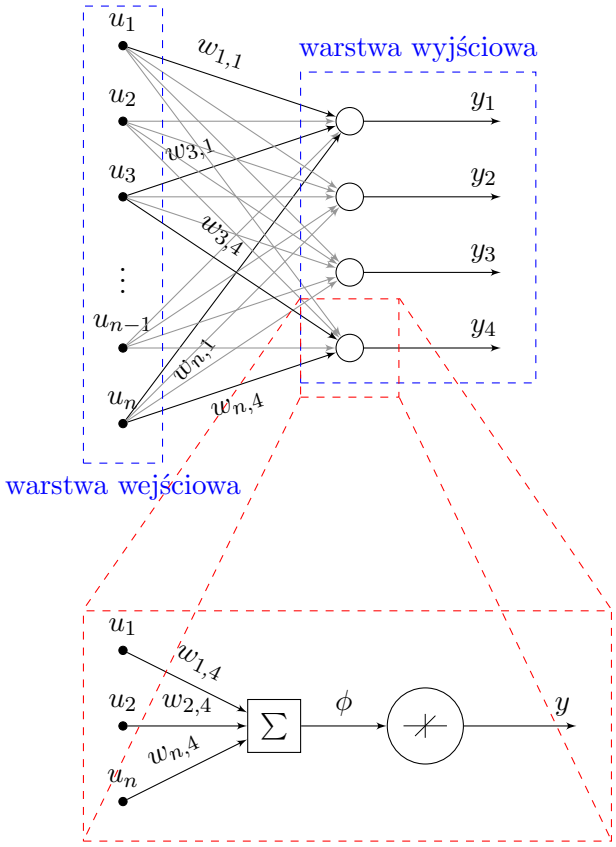
ESI: Sieci liniowe i nieliniowe sieci wielowarstwowe

[Matlab_1.1] Matlab2016b i nowsze

15 kwietnia 2019

1. **Cel ćwiczeń:** Celem ćwiczeń jest zapoznanie się studentów z zagadnieniami z zakresu sztucznych sieci neuronowych związanymi z sieciami liniowymi oraz wielowarstwowymi sieciami nieliniowymi.

2. **Struktury sieci neuronowych:** Pojedyncze neurony można łączyć w sieci jedno- lub wielowarstwowe. Podstawową, najprostszą siecią jest **liniowa sieć neuronowa**. Rys. 1 przedstawia liniową sieć jednowarstwową. Są one zbudowane z równolegle połączonych neuronów liniowych. Najczęściej



Rysunek 1: Jednowarstwowa liniowa sztuczna sieć neuronowa

wszystkie wejścia sieci połączone są z każdym neuronem. Sieci liniowe mogą być wykorzystywane w

wielu aplikacjach, np. jako wielowymiarowe aproksymatory liniowe, klasyfikatory czy filtry liniowe. W ogólnej postaci, sieć liniowa jest opisana następująco

$$y = \sum_{i=1}^{n} \Big(u_i w_i + w_{n+1} \Big) = WU, \tag{1}$$

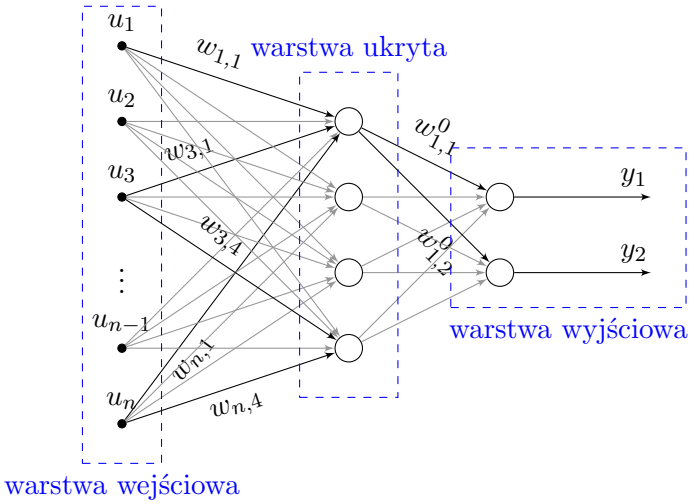
gdzie W jest macierzą wag a U wektorem danych wejściowych. Strukturą bardziej złożoną, zdolną również do rozwiązywania problemów nieliniowych jest **perceptron wielowarstwowy** (MLP, ang. Multi-Layer Perceptron), przedstawiony na Rys. 2. W ogólnej postaci (dwuwarstwowej) opisany jest on następująco

$$y = \sigma\left(\sum_{i=1}^{n_0} w_{0,i} \sigma\Big(\sum_{j=1}^{n} w_{1,j} u_j + w_{1,n+1} \Big) + w_{0,n+1} \right), \tag{2}$$

jednakże znacznie częściej stosuje się MLP w następującej postaci

$$y = W_0 \sigma\Big(\sum_{i=1}^{n} u_i w_i + w_{n+1} \Big), \tag{3}$$

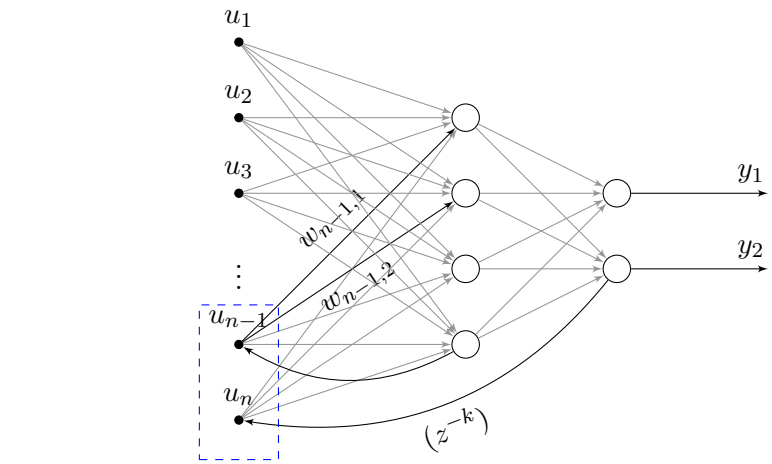
co oznacza, iż warstwa wyjściowa jest warstwą liniową podczas gdy warstwa ukryta złożona jest z neuronów nieliniowych (najczęściej o bipolarnej, sigmoidalnej funkcji aktywacji). Zazwyczaj neurony z warstwy ukrytej są połączone z warstwą wyjściową w relacji **każdy z każdym**, tak samo jak wejścia z neuronami w warstwie ukrytej. W przypadku tradycyjnych (niegłębokich) sieci neuronowych rzadko stosuje się architekturę o większej ilości warstw. Powodem tego są problemy z uczeniem takich sieci związane z efektem **zanikania gradientu** [6, 2] oraz dużą złożonością pamięciowo–obliczeniową sieci o dużej liczbie neuronów. Ponadto, podłączając wyjście sieci do wejścia (z opóźnieniem) powstaje **sieć**



Rysunek 2: Sztuczna sieć neuronowa typu MLP

rekurencyjna (ang. RNN, Recurent Neural Network). Dzięki rekurencji sieci tego typu posiadają **pamięć**, wykorzystywaną w sieciach LSTM (ang. Long-Short Term Memory) [4], ESN (ang. Echo State Network) [7] czy w sieciach Hopfielda [3]. Dzięki sprzężeniu zwrotnemu sieci rekurencyjne typu

NAR (ang. nonlinear autoregression) i NARX (ang. nonlinear autoregression with exogenous input) nadają się również np. do przewidywania szeregów czasowych lub symulacji układów dynamicznych. Również w sieciach rekurencyjnych występuje zjawisko zanikania gradientu, co wymusza stosowanie metod uczenia zapobiegających temu zjawisku. Sieci rekurencyjne mogą posiadać sprzężenie zwrotne z warstwy ukrytej lub z warstwy wyjściowej. Rys. 3 przedstawia sieć ze sprzężeniem od warstwy ukrytej oraz wyjściowej z opóźnieniem o k próbek. W sieciach płytkich wejścia opóźnione najczęściej opóźniane są o $k \in \{1, 2\}$ próbek.



Rysunek 3: Przykładowa rekurencyjna sieć neuronowa

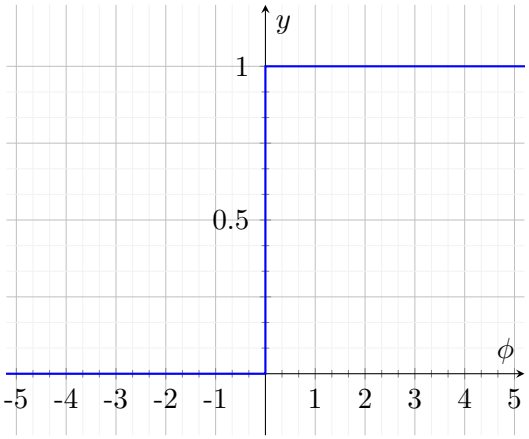
3. **Nieliniowe funkcje aktywacji:** Poniżej przedstawiono najpopularniejsze funkcje aktywacji wykorzystywane w sieciach płytkich

   funkcja unipolarna skoku jednostkowego $y = \begin{cases} 0, & \phi \leqslant 0 \\ 1, & \phi > 0 \end{cases}$, funkcja przedziałami różniczkowalna, monotoniczna, wykorzystywana w klasyfikatorach binarnych. Rys. 4 przedstawia wykres funkcji skoku jednostkowego.

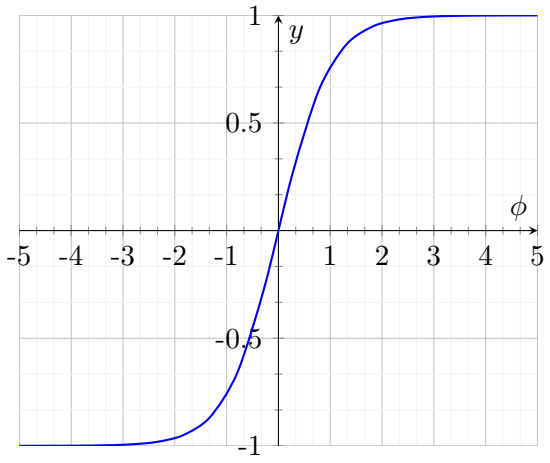
   tangens hiperboliczny $y = tanh(\phi)$, funkcja różniczkowalna, gładka i monotoniczna, wykorzystywana najczęściej w nieliniowych perceptronach wielowarstwowych (MLP) lub w sieciach rekurencyjnych. Rys. 5 przedstawia wykres funkcji sigmoidalnej – tangens hiperboliczny.

   funkcja Gaussa $y = ae^{-\frac{(x-b)^2}{2c^2}}$, funkcja różniczkowalna, gładka, niemonotoniczna, wykorzystywana w sieciach RBF, także rekurencyjnych. W reprezentacji rozkładu gęstości prawdopodobieństwa przyjmuje się $b = \mu, c = \sigma, a = \frac{1}{\sigma\sqrt{2\pi}}$, zatem dla rozkładu normalnego o parametrach $\mathcal{N}(0, 1)$, $y = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$. Rysunek 6 przedstawia krzywą Gaussa dla standardowego rozkładu normalnego.

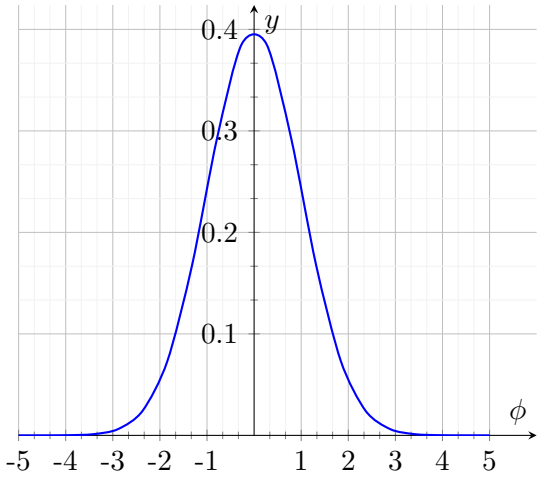
4. **Uczenie sieci wielowarstwowych:** Uczenie sieci z nauczycielem polega na takim dostosowywaniu wag neuronów aby wartość na wyjściu sieci była możliwie bliska wartości oczekiwanej. Istnieje



Rysunek 4: Wykres funkcji skoku jednostkowego



Rysunek 5: Wykres funkcji tangens hiperboliczny

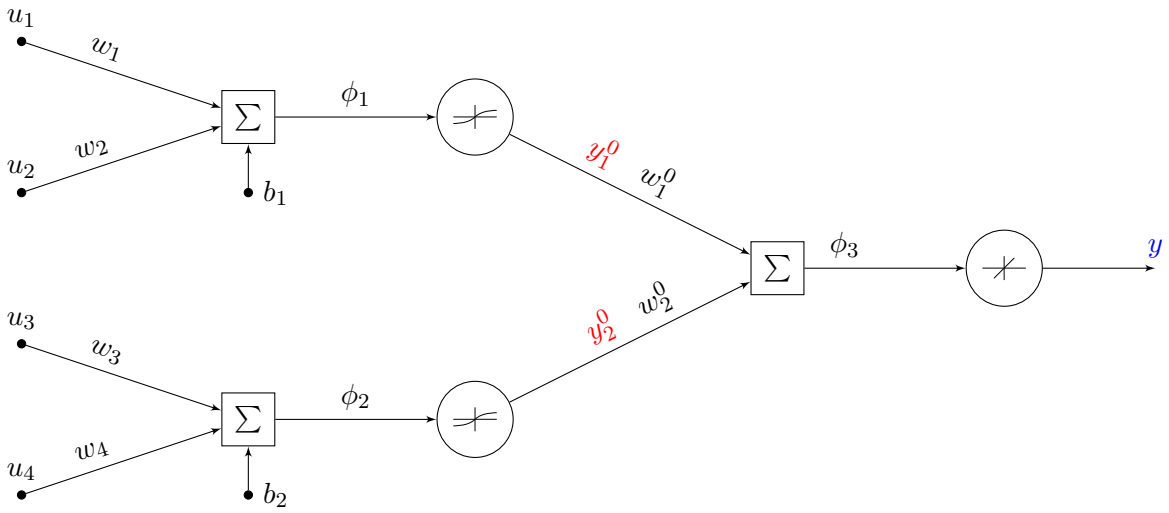


Rysunek 6: Wykres funkcji Gaussa o parametrach $\mathcal{N}(0,1)$

wiele metod dostosowywania wag aby spełnić ten warunek. W przypadku sieci nieliniowych wielowarstwowych najpopularniejsze są metody gradientowe. Pozwalają one na wyznaczenie pochodnych cząstkowych funkcji aktywacji w formie gradientu, przez co możliwe jest iteracyjne dążenie do minimum funkcji celu, tj. minimum odległości między wartościami wyjściowymi i oczekiwanymi. Jedną ze stosowanych funkcji celu jest błąd średniokwadratowy (MSE) dany wzorem

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \bar{y}_i)^2. \tag{4}$$

Jeżeli $MSE < \epsilon$ jest mniejsze od pewnej małej wielkości zadanej oznacza to, iż proces uczenia można uznać za zakończony. W przypadku sieci jednowarstwowych cała procedura jest nieskomplikowana, jako iż wielkość MSE bezpośrednio wpływa na wagi sieci. Trudniejszym w realizacji jest algorytm uczenia sieci wielowarstwowej, jako iż dla warstwy wyjściowej MSE jest łatwe do określenia, o tyle odchyłka na warstwach ukrytych musi być wyznaczona w sposób pośredni. Ilustruje to Rysunek 7, gdzie na niebiesko zaznaczono parametry których oczekiwane wartości są znane, a na czerwono nie są znane. **Algorytm wstecznej propagacji błędów** powstał aby rozwiązać ten problem. W ujęciu



Rysunek 7: Fragment perceptronu wielowarstwowego

intuicyjnym polega on na propagacji błędu od wyjścia sieci do wejść. Graficznie, przepływ wsteczny gradientu może być przedstawiony w formie grafu skierowanego, w którym węzły są bramkami (funkcjami). Rysunek 8 przedstawia ideę algorytmu wstecznej propagacji. Poszczególne bramki oznaczają funkcje: dodawania $\sum$, mnożenia $\prod$, odwrotną $\frac{1}{x}$ oraz wykładniczą e^x. Na niebiesko oznaczono kolejne pochodne cząstkowe zgodnie z tzw. regułą łańcuchową. Reguła łańcuchowa oparta jest o **twierdzenie o pochodnej funkcji złożonej**. W notacji Leibnitza określa się ją następująco

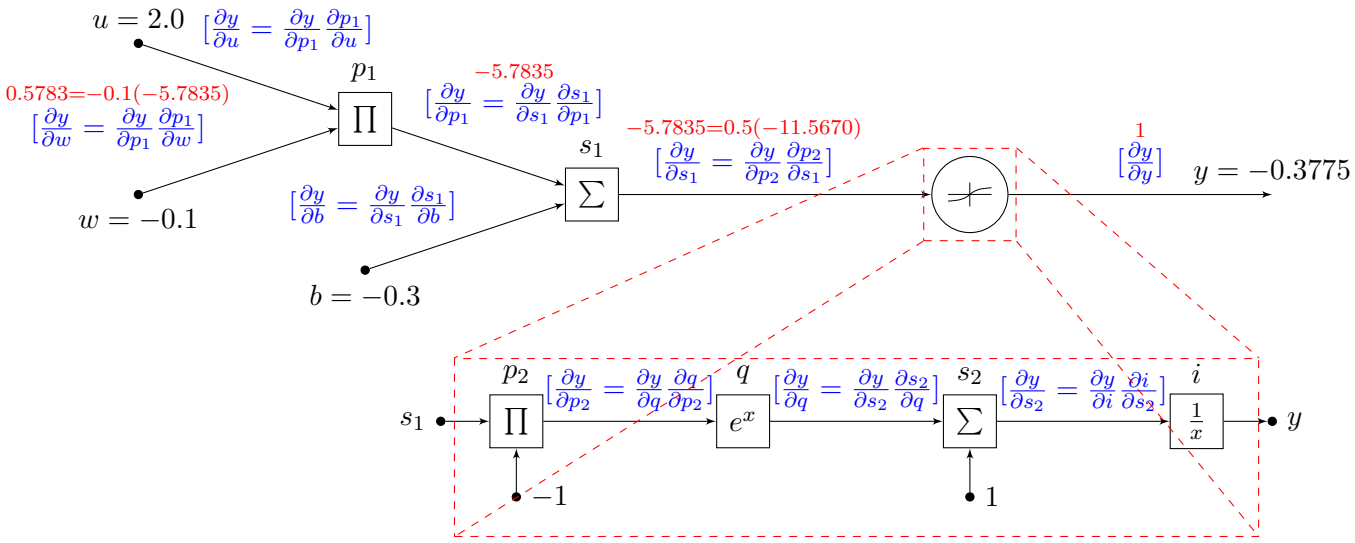
$$\frac{\partial y}{\partial x} = \frac{\partial t}{\partial x}\frac{\partial y}{\partial t}, \tag{5}$$

gdzie $y = f(g(x))$, natomiast $t = g(x)$. Na czerwono oznaczono wartości pochodnych cząstkowych.

Analizując strukturę wszystkich kolejnych pochodnych wzdłuż kierunku wstecznej propagacji należy zauważyć, iż jednym z czynników iloczynu jest pochodna *lokalna* a drugim *zwrotna*. W ten sposób wartości są propagowane wstecz, od wyjścia do wejścia (do wewnątrz funkcji złożonej jaką jest neuron a w konsekwencji sieć neuronowa). Końcowa wartość pochodnej cząstkowej $\frac{\partial y}{\partial w}$ uwzględniana jest w algorytmie aktualizacji wag, np. dla *algorytmu największego spadku*

$$w(k+1) = w(k) + \eta \frac{\partial y}{\partial w} = -0.1 - 0.5783 = -0.6783, \tag{6}$$

przy $\eta = 1$.



Rysunek 8: Przepływ gradientu

5. **Normalizacja danych wejściowych i wyjściowych:** Analizując Rys. 5 można szybko zauważyć, iż funkcja tangens hiperboliczny szybko przyjmuje wartości graniczne.

$$\lim_{\phi \to \infty} tanh(\phi) = 1, \quad \lim_{\phi \to -\infty} tanh(\phi) = -1, \tag{7}$$

innymi słowy oznacza to szybkie nasycenie (dla $y(\phi > 3) \approx 1$ oraz $y(\phi < -3) \approx -1$). Nieliniowe właściwości sieci mogą zostać osłabione ze względu na liniowy (oraz o słabym nachyleniu) charakter funkcji sigmoidalnych w pewnych przedziałach swojej dziedziny, tj.

$$|\phi| > 3 \Rightarrow \frac{dy}{d\phi} \approx 0, \tag{8}$$

zatem zmiana warości parametru funkcji nie prowadzi do dalszej zmiany wartości funkcji. Do podobnych wniosków można dojść analizując dzwon Gaussa z Rys. 6, z tym że

$$\lim_{\phi \to \infty} tanh(\phi) = 0, \quad \lim_{\phi \to -\infty} tanh(\phi) = 0 \tag{9}$$

a

$$\phi > 5 \Rightarrow y = \frac{1}{\sqrt{2\pi}} e^{-\frac{\phi^2}{2}} < 1.5e-6 \tag{10}$$

oraz

$$|\phi| > 5 \Rightarrow \frac{dy}{d\phi} \approx 0. \tag{11}$$

Stąd, aby uniknąć zerowania pochodnych prowadzącego do zaniku nieliniowych właściwości sieci należy wprowadzić normalizację danych wejściowych.

Normalizacja poprzez skalowanie liniowe (normalizacja min-max). Pozwala na zmianę zakresu danych wejściowych, tj. element największy wektora danych wejściowych (x_{max}) skalowany jest do nowej wartości największej (b), element najmniejszy (x_{min}) do nowego elementu najmniejszego (a), natomiast wartości pośrednie liniowo mapowane do nowej dziedziny

$$f(x)_{ab} = \frac{(b-a)(x-x_{min})}{x_{max}-x_{min}} + a. \tag{12}$$

Normalizacja poprzez standaryzację. Polega na sprowadzenie parametrów danych wejściowych do $\mathcal{N}(0,1)$

$$f(x)_{\mu\sigma} = \frac{x-\mu}{\sigma}, \tag{13}$$

gdzie estymatorem wartości oczekiwanej μ jest średnia arytmetyczna a estymatorem odchylenia standardowego pierwiastek średniej arytmetycznej kwadratów odchyleń, zatem $\sigma = \sqrt{MSE}$.

Denormalizacja W celu zapewnienia poprawnego działania sieci z normalizacją $f_n(u)$ na wejściu należy na wyjściu zastosować funkcję odwrotną $f^{-1}(y)$.

6. **Predefiniowane struktury sztucznych sieci neuronowych w środowisku Matlab:** W środowisku Matlab istnieje możliwość prostej i szybkiej implementacji wielu różnych struktur sieci neuronowych. Użytkownik ma do wyboru wiele predefiniowanych struktur lub może stworzyć sieć o zdefiniowanej przez siebie architekturze [1]. Poniżej przedstawiono kilka popularnych struktur.

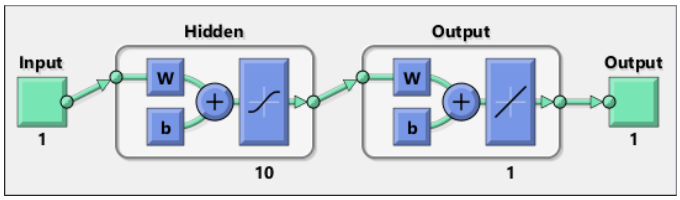
MLP Perceptron wielowarstwowy jest podstawową sztuczną siecią neuronową predefiniowaną w środowisku Matlab. Do zainicjalizowania struktury sieci MLP wykorzystuje się polecenie *feedforwardnet*, którego podstawowym parametrem jest liczba neuronów w warstwie ukrytej. Domyślnie

```
1 net = feedforwardnet(10);    %MLP z 10 neuronami w warstwie ukrytej
```

Listing 1: Inicjalizacja sieci typu MLP z 10 neuronami w warstwie ukrytej.

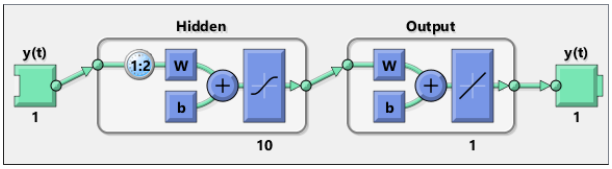
sieć ta jest inicjalizowana z dwoma warstwami, z sigmoidalnymi funkcjami aktywacji w wartwie ukrytej (tangens hiperboliczny, *tansig*) w warstwie ukrytej oraz liniowymi (*purelin*) w warstwie wyjściowej. Ilość wejść oraz wyjść jest ustawiana automatycznie po wywołaniu funkcji uczącej *train* za rzecz zdefiniowanej sieci, na podstawie danych uczących. Domyślną metodą uczenia

jest metoda Levenberga–Marquardta a domyślną funkcją celu MSE. Rysunek 2 Przedstawia przykładowy perceptron wielowarstwowy zainicjalizowany poleceniem z Listingu 1.

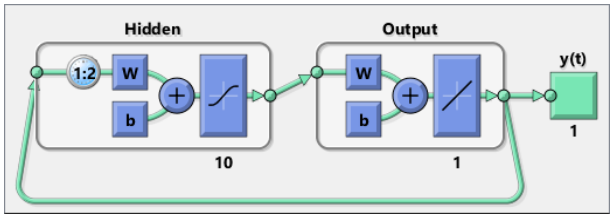


Rysunek 9: Perceptron wielowarstwowy

NAR Nonlinear autoregressive - to najprostsza sieć rekurencyjna. w trybie uczenia pętla sprzężenia zwrotnego jest otwarta (Rys. 10a) natomiast w trybie odtwarzania zamknięta (Rys. 10b). Domyślną metodą uczenia jest metoda Levenberga–Marquardta a domyślną funkcją celu MSE. Sieci typu NAR stosować można do przewidywania szeregów czasowych. W środowisku Matlab



(a) otwarta pętla



(b) zamknięta pętla

Rysunek 10: Sieć typu NAR

do inicjalizacji sieci NAR służy funkcja *narnet* natomiast w celu przygotowania danych (przesunięcie próbek) stosuje się funkcję *preparets* jako pokazano na Listingu 2. Do zamknięcia pętli

```
net = narnet(1:2,10); % 10 neuronow opoznienia o 1 i 2 probki
[Xs,Xi,Ai,Ts] = preparets(net,{},{},T);
```

Listing 2: Inicjalizacja sieci typu NAR z 10 neuronami w warstwie ukrytej i przygotowanie zbioru danych uczących

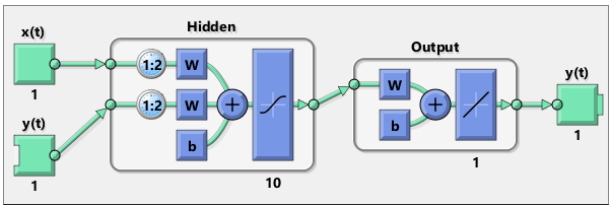
sprzężenia zwrotnego służy polecenie *closeloop* (patrz. Listing 3) Sieć typu NAR po zamknięciu pętli nie posiada wejść, w każdej iteracji generuje nowy element szeregu bazując jedynie na elementach poprzednich (w zależności od opóźnionych wejść).

```
[netc,Xic,Aic] = closeloop(net,Xf,Af);
```

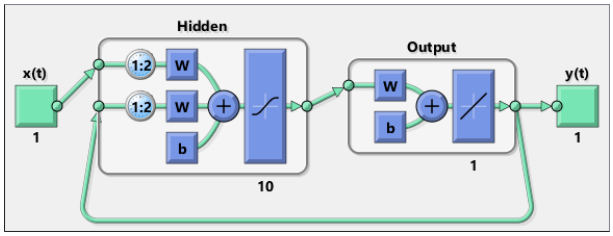
Listing 3: Zamykanie pętli sprzężenia zwrotnego

NARX nonlinear autoregressive with exogenous input – sieć rekurencyjna o wejściach bez sprzężenia zwrotnego. W trybie uczenia, tak jak w sieciach typu NAR uczenie odbywa się z otwartą

pętlą (Rys. 11a). Polecenie *narxnet* służ do inicjalizacji struktury sieci i przyjmuje takie same parametry jak funkcja *narnet*. Podobnie jak dla innych struktur funkcja *train* służy do uczenia sieci. Po nauczeniu sieci należy zamknąć pętlę (Rys. 11b) poleceniem *closeloop* (Listing 3). Tak przygotowana sieć może odtwarzać zachowanie układów dynamicznych lub służyć do predykcji szeregów czasowych.



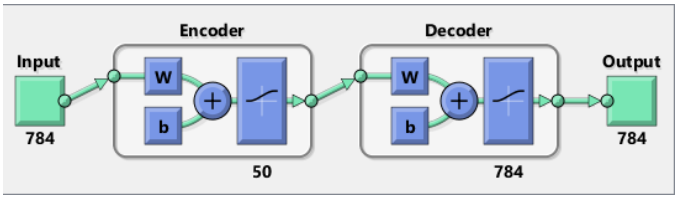
(a) otwarta pętla



(b) zamknięta pętla

Rysunek 11: Sieć typu NARX

Autoencoder Jest to specyficzny rodzaj sieci neuronowej, który można wykorzystać do kompresji i dekompresji danych lub jako element klasyfikatora. Autoencoder uczony jest odwzorowania funkcji tożsamościowej, tj. przekształcania zbioru samego w siebie. W literaturze klasyfikowany jest jako przykład sieci uczonej bez nauczyciela, mimo iż w istocie oczekiwana wartość wyjściowa sieci tożsama jest z wejściem. Autoencoder składa się z dwóch części **encodera** i **decodera**, natomiast każda z nich posiada min. jedną warstwę. Co do zasady część encoderowa jest ekstraktorem cech i ma za zadanie zmniejszyć wymiarowość problemu reprezentowanego przez dane wejściowe. Decoder z kolei pełni rolę odwrotną – na podstawie cech dostarczanych przez encoder powinien on odtwarzać dane wejściowe. Budowę przykładowego autoecodera przedstawia Rys. 12. Należy również zwrócić uwagę, iż ze względu charakter sieci tego typu, jako funkcja ak-



Rysunek 12: Przykładowy autoencoder

tywacji wykorzystywana jest sigmoidalna funkcja unipolarna *logsig* a jako funkcję błędu stosuje się zmodyfikowane MSE.

7. **Zadania:**

7.1. Wykorzystać liniową sieć neuronową (inicjalizacja i uczenie poleceniem *newlind*) w celu wyznaczenia parametrów równania regresji liniowej $y = ax + bz + c$. Wykorzystać następujące dane

uczące

$$U = \begin{bmatrix} x_1, \ldots, x_n \\ z_1, \ldots, z_n \end{bmatrix} = \begin{bmatrix} 1 & 3 & 4 & 5 & 6 & 8 & 9 & 11 & 14 & 16 \\ -1 & 4 & 4 & 6 & 8 & 8 & 10 & 12 & 15 & 18 \end{bmatrix},$$
$$Y = \begin{bmatrix} y_1, \ldots, y_n \end{bmatrix} = \begin{bmatrix} 14 & 10 & 13 & 12 & 11 & 17 & 16 & 18 & 21 & 21 \end{bmatrix}.$$

Podejrzeć strukturę sieci poleceniem *view*. Wagi sieci znajdują się w strukturze (np. *net*) zwróconej przez polecenie *newlind* w polach $IW\{1\}$ oraz b (np. $net.IW\{1\}$).

7.2. Zdefiniować macierz jak na Listingu 4. Zastosować funkcję *mapminmax* do normalizacji wartości elementów macierzy A do przedziału $\langle -1, 0 \rangle$. Zastosować funkcję *mapstd* do normalizacji wartości elementów macierzy A do rozkładu o parametrach $\mu = 2, \sigma = 4$. Wyniki zapisać odpowiednio do macierzy B i C. Sprawdzić poprawność normalizacji.

```
A = randn(2,20)*100; % macierz 2x20 losowych wartosci o rozkladzie normlanym
```

Listing 4: Macierz do zadania 7.2.

.

7.3. Wykorzystując polecenia *feedforwardnet* oraz *train* zainicjalizować oraz wytrenować sieć o nazwie *net*, która będzie realizować następujące równanie $y = x^2$. Za dane uczące przyjąć wektor liczb rzeczywistych z przedziału $\langle -4, 4 \rangle$ (z interwałem co 0.1). Za ilość neuronów w warstwie ukrytej przyjąć kolejno $n \in \{1, 4, 8, 20, 100\}$. Strukturę sieci podejrzeć używając polecenia *view*. Sprawdzić jakość odwzorowania dla liczb z przedziału $\langle -6, 6 \rangle$.

7.4. Sprawdzić jakie funkcje normalizacyjne/filtrujące są wykorzystywane w sieci z zad 2. Wykorzystać polecenia z Listingu 5.

```
net.input.processFcns       % funkcje przetwarzajace wejscie sieci
net.input.processParams     % parametry funkcji przetwarzajacych wejscie
net.input.processSettings   % wlasciwosci funkcji przetwarzajacych wejscie
net.output.processFcns      % funkcje przetwarzajace wyjscie
net.output.processParams    % parametry funkcji przetwarzajacych wyjscie
net.output.processSettings  % wlasciwosci funkcji przetwarzajacych wyjscie
```

Listing 5: Polecenia do zad. 7.4.

.

7.5. Sprowadzić macierze B i C do macierzy A odwracając proces normalizacji z zad. 7.2.

7.6. Sprawdzić jak zachowuje się rekurencyjna sieć neuronowa typu NAR dla przykładowego szeregu czasowego w zależności od wejść opóźnionych o $k \in \{(1, 2), (1, 2, 3), (1, 3)\}$ oraz w zależności od liczby neuronów w warstwie ukrytej $n \in \{5, 10, 15, 50\}$. W badaniach wykorzystać skrypt z Listingu 6

7.7. Wykorzystując polecenie z Listingu 7 wczytać przykład zastosowania autoencodera do odtwarzania pisma ręcznego. Sprawdzić jakość odtwarzania dla różnych parametrów sieci

```
clear;clc;close all hidden;
nnet.guis.closeAllViews();  % zamkniecie okien narzedzia sieci neuronowych
Tfull = simplenar_dataset;  % wczytanie danych demonstracyjnych
T = Tfull(1:50);
net = narnet(1:2,5);        % zdefiniowanie struktury sieci NAR
[Xs,Xi,Ai,Ts] = preparets(net,{},{},T); % przygotowanie zbioru uczacego
net = train(net,Xs,Ts,Xi,Ai);
view(net);  % siec z otwarta petla sprzezenia zwrotnego
[Y,Xf,Af] = net(Xs,Xi,Ai);
perf = perform(net,Ts,Y)
[netc,Xic,Aic] = closeloop(net,Xf,Af); % zamkniecie petli
view(netc)
y2 = netc(cell(0,50),Xic,Aic)
% wyswietlenie wynikow
hold on
    plot(cell2mat(Tfull(51:100)),'r-');
    plot(cell2mat(y2),'b--');
hold off;
```

Listing 6: Skrypt do zadania 7.6..

.

```
openExample('nnet/ReconstructHandwrittenDigitImagesUsingSparseAutoencoderExample')
```

Listing 7: Skrypt do zadania 7.7.

.

- ilości neuronów na wyjściu endodera *hiddenSize* $\in \{3, 4, 10, 25, 50\}$,
- współczynnika regularyzacji (*'L2WeightRegularization'*) w przedziale $\langle 0.000, 0.01 \rangle$
- współczynnika dywergencji [5] (*'SparsityRegularization'*) w przedziale $\langle 3, 5 \rangle$

7.8. Zmodyfikować sieć z zadania 5. do predykcji dochodu narodowego Polski, z wykorzystaniem danych historycznych. Wczytać dane jak na Listingu 8.

```
rawdata_gni = csvread('UIFS-NA_GNI_POL.csv',1,1,[1,1,30,1]);
data_gni = fliplr(rawdata_gni'/10^10);
T_feedback = num2cell(data_gni);
```

Listing 8: Dane do zadania 7.8.

.

7.9. Zaimplementować sieć typu NARX do predykcji dochodu narodowego Polski z wykorzystaniem danych archiwalnych dochodu narodowego oraz wartości eksportu. Wykorzystać skrypt z Listingu 9. Porównać wyniki ze skryptem z zad 7.8., wykonać oba skrypty wielokrotnie.

7.10. Wykorzystując polecenie z Listingu 10 sprawdzić działanie sieci RBF jako aproksymatora. Zbudować sieć MLP o takiej samej liczbie neuronów i porównać wyniki. Do wyświetlania struktury sieci użyć polecenia *view*. Zbadać jak zmienia się zachowanie procesu uczenia oraz samej sie-

```
clear;clc;close all hidden;
nnet.guis.closeAllViews();
rawdata_gni = csvread('UIFS-NA_GNI_POL.csv',1,1,[1,1,30,1]);
rawdata_exp = csvread('UIFS-BP_EVA_POL.csv',1,1,[1,1,30,1]);
rawdata_exp = rawdata_exp(:,1);
data_gni = fliplr(rawdata_gni'/10^10);
data_exp = fliplr(rawdata_exp'/10^10);
T_nonfeedback = num2cell(data_exp);
T_feedback = num2cell(data_gni);
net = narxnet(1,1:2,7);
[Xs,Xi,Ai,Ts] = preparets(net,T_nonfeedback,{},T_feedback);
net = train(net,Xs,Ts,Xi,Ai);
[Y,Xf,Af] = net(Xs,Xi,Ai);
perf = perform(net,Ts,Y)
[netc,Xic,Aic] = closeloop(net,Xf,Af);
netp = net;%removedelay(net);
[Xs,Xi,Ai,Ts] = preparets(netp,T_nonfeedback,{},T_feedback);
y2 = netp(Xs,Xi,Ai);
view(netp);
hold on
    plot(data_gni,'r--');
    plot(cell2mat(y2),'b');
hold off;
```

Listing 9: Skrypt do zadania 7.9.

.

```
openExample('nnet/demorb1')
```

Listing 10: Skrypt do zadania 7.10.

.

ci RBF dla różnych parametrów *eg* (maksymalny MSE) i *ec* (maksymalna rozpiętość krzywej dzwonowej)

Literatura

[1] Shallow Neural Network Architectures. https://www.mathworks.com/help/deeplearning/define-neural-network-architectures.html.

[2] Sepp Hochreiter. The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 06(02):107–116, 2003.

[3] J J Hopfield. Computational Abilities. *Biophysics*, 79(April):2554–2558, 1982.

[4] Xiaolei Ma, Zhimin Tao, Yinhai Wang, Haiyang Yu, and Yunpeng Wang. Long short-term memory neural network for traffic speed prediction using remote microwave sensor data. *Transportation Research Part C: Emerging Technologies*, 54:187–197, 2015.

[5] Bruno A Olshausen and David J Fieldt. Sparse Coding with an Overcomplete Basis Set: A Strategy Employed by V1 ? Coding V1 Gaborwavelet Natural images. *Vision Res*, 37(23):33113325, 1997.

[6] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. BT - Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013. (2):1310–1318, 2013.

[7] Mark D Skowronski and John G Harris. Automatic speech recognition using a predictive echo state network classifier. *Neural networks*, 20(3):414–423, 2007.