4. DEFINIOWANIE ARCHITEKTURY SIECI – FUNKCJA network

Biblioteka *Neural Network Toolbox* udostępnia wiele różnych metod tworzenia sieci neuronowych. Dostępne są specjalizowane funkcje do tworzenia konkretnych rodzajów sieci, można również korzystać z polecenia **network** – umożliwiającego wygenerowanie sieci o dowolnej strukturze.

Polecenie **network** można wywołać bez parametrów:

```
>> net = network

net =
    Neural Network object:

    architecture:
                numInputs: 0
                numLayers: 0
              biasConnect: []
             inputConnect: []
             layerConnect: []
            outputConnect: []
            targetConnect: []
               numOutputs: 0    (read-only)
               numTargets: 0    (read-only)
          numInputDelays: 0    (read-only)
          numLayerDelays: 0    (read-only)
    subobject structures:
                  inputs: {0x1 cell}    of inputs
                  layers: {0x1 cell}    of layers
                 outputs: {1x0 cell}    containing no outputs
                 targets: {1x0 cell}    containing no targets
                  biases: {0x1 cell}    containing no biases
            inputWeights: {0x0 cell}    containing no input weights
            layerWeights: {0x0 cell}    containing no layer weights
    functions:
                adaptFcn: (none)
                initFcn: (none)
             performFcn: (none)
                trainFcn: (none)
    parameters:
              adaptParam: (none)
              initParam: (none)
           performParam: (none)
              trainParam: (none)
    weight and bias values:
                      IW: {0x0 cell}    containing no input weight matrices
                      LW: {0x0 cell}    containing no layer weight matrices
                       b: {0x1 cell}    containing no bias vectors
    other:
            userdata: (user stuff)
```

Wykonanie powyższego polecenia powoduje utworzenie zmiennej o nazwie **net**, która reprezentuje sieć o niezdefiniowanej jeszcze architekturze. Architekturę tą można zdefiniować odwołując się do odpowiednich elementów składowych zmiennej **net**.

Polecenie **network** może być również wywołane z parametrami – znaczenie i kolejność parametrów omówiono w podanym poniżej przykładzie.

4.1. Składowe zmiennej net

4.1.1. Znaczenie składników opisujących architekturę sieci

(podpunkt **architecture** widoczny podczas wyświetlania wartości zmiennej reprezentującej sieć neuronową)

net.numInputs – liczba definiująca ilość źródeł wejściowych (a nie rozmiar wektora wejściowego), w przypadku prostych sieci parametr ten wynosi zawsze 1,

net.numLayers – liczba określająca ilość warstw sieci,

net.biasConnect – wektor o rozmiarze równym ilości warstw sieci, element wektora o indeksie *i* definiuje czy neurony *i*-tej warstwy posiadają bias, elementy wektora mogą przyjmować wartości : 1 (neurony posiadają bias) i 0 (brak biasu),

net.inputConnect – macierz o rozmiarze **net.numLayers** x **net.numInputs**, element (*i*, *j*) macierzy definiuje czy *i*-ta warstwa jest połączona z *j*-tym źródłem wejściowym, elementy macierzy mogą przyjmować wartości: 1 (jest połączenie) i 0 (brak połączenia). Jeżeli *i*-ta warstwa otrzymuje na wejście sygnały z *j*-tego źródła, to macierz wag neuronów tej warstwy jest przechowywana w elemencie **net.inputWeights{i,j}**,

net.layerConnect – macierz o rozmiarze **net.numLayers** x **net.numLayers**, element (*i*, *j*) macierzy definiuje czy *i*-ta warstwa przetwarza sygnały wyjściowe *j*-tej warstwy, elementy macierzy mogą przyjmować wartości: 1 (przetwarza) i 0 (nie przetwarza). Jeżeli warstwa *i* otrzymuje sygnały z warstwy *j*, to macierz wag neuronów tej warstwy jest przechowywana w elemencie **net.layerWeights{i,j}**,

net.outputConnect – wektor o rozmiarze równym ilości warstw sieci, element wektora o indeksie *i* definiuje czy sygnały wyjściowe tej warstwy są jednocześnie sygnałami wyjściowymi sieci, elementy wektora mogą przyjmować wartości: 1 (sygnały są sygnałami wyjściowymi) i 0 (sygnały nie są sygnałami wyjściowymi),

net.targetConnect – jest wektorem o rozmiarze odpowiadającym ilości warstw sieci, informacje w nim zawarte wykorzystywane są w procesie uczenia, element o indeksie *i* opisuje czy sygnały wyjściowe z *i*- tej warstwy mają być wykorzystywane do wyznaczenia błędu uczenia (poprzez porównanie z zadanymi

wartościami docelowymi), w przypadku prostych sieci wektor ten ma takie same wartości **jak net.outputConnect**,

net.numOutputs – liczba opisująca ilość źródeł wyjściowych sieci, jej wartość jest związana z wartością wektora **net.outputConnect**, **net.numOutputs** jest równa ilości niezerowych elementów wektora **net.outputConnect**,

net.numTargets – liczba równa ilości niezerowych elementów wektora **net.targetConnect**.

Przykład 1.

Należy utworzyć zmienną reprezentującą sieć jednokierunkową z:

- jednym źródłem sygnałów wejściowych,
- dwoma warstwami (sygnały wyjściowe z warstwy pierwszej przekazywane są na wejścia warstwy drugiej).

Dodatkowo, sygnały wyjściowe drugiej warstwy powinny być sygnałami wyjściowymi sieci i mają być wykorzystywane w trakcie jej uczenia. Wszystkie neurony sieci powinny posiadać bias.

>> **net = network;**	zmienna **net** reprezentuje sieć o niezdefiniowanej strukturze
>> **net.numInputs = 1;**	sieć ma jedno źródło sygnałów wejściowych
>> **net.numLayers = 2;**	sieć ma dwie warstwy neuronów
>> **net.biasConnect = [1; 1];**	neurony warstwy pierwszej i drugiej posiadają bias
>> **net.inputConnect = [1; 0];**	sygnały wejściowe są podawane na wejścia pierwszej warstwy
>>**net.layerConnect = [0 0; 1 0];**	warstwa druga otrzymuje na wejściu sygnały wyjść warstwy pierwszej
>>**net.outputConnect = [0 1];**	sygnałami wyjściowymi sieci są sygnały wyjściowe drugiej warstwy
>>**net.targetConnect = [0 1];**	sygnały wyjściowe drugiej warstwy będą wykorzystywane w trakcie uczenia

Szczegóły dotyczące architektury tworzonej sieci można podać już w samym poleceniu **network**, kolejność parametrów musi odpowiadać podanym powyżej przypisaniom, tzn:

>> **net = network(1, 2, [1;1], [1;0], [0 0; 1 0], [0 1], [0 1]);**

4.1.2 Znaczenie składników opisujących wejścia, wyjścia oraz wagi sieci:
(podpunkt **subobject structures** widoczny podczas wyświetlania wartości zmiennej reprezentującej sieć neuronową)

net.inputs – definiuje właściwości poszczególnych źródeł wejściowych sieci, jest tablicą blokową[1] o wymiarze **net.numInputs**. Dostęp do opisu właściwości *i*-tego źródła jest możliwy poprzez odwołanie: **net.inputs{i}**. Na definicję każdego ze źródeł składają się:

-rozmiar źródła (**size**),

-zakres wartości każdego z sygnałów wejściowych, tzn. jego minimalna i maksymalna wartość (**range**), składnik ten jest tablicą o ilości wierszy odpowiadającej liczbie sygnałów wejściowych źródła i liczbie kolumn równej dwa (w pierwszej kolumnie powinna znajdować się minimalna wartość sygnału a w drugiej jego maksymalna wartość),

net.layers – definiuje właściwości poszczególnych warstw sieci, jest tablicą blokową o wymiarze **net.numLayers**. Dostęp do opisu właściwości *i*-tej warstwy jest możliwy poprzez odwołanie: **net.layers{i}.** Na definicję każdej z warstw składają się np.:

-ilość neuronów warstwy (**size**),

-typ funkcji aktywacji neuronów (**transferFcn**),

net.outputs – opisuje właściwości wyjść sieci, jest tablicą blokową o wymiarze **net.numLayers** – jeżeli sygnały wyjściowe *i*-tej warstwy sieci są jednocześnie sygnałami wyjściowymi sieci to odwołanie: **net.outputs{i}** daje dostęp do informacji o tym wyjściu (np. o ilości neuronów leżących na tej warstwie sieci – składowa **size**),

net.targets – opisuje właściwości wyjść sieci, które są wykorzystywane w procesie uczenia (do wyznaczenia błędu uczenia), jest tablicą blokową o wymiarze **net.numLayers** – jeżeli sygnały wyjściowe *i*-tej warstwy sieci są wykorzystywane do wyznaczania błędu uczenia to odwołanie: **net.targets{i}** daje dostęp do informacji o tym wyjściu (np. o ilości neuronów leżących na tej warstwie sieci – składowa **size**),

---

[1]Tablica blokowa – (ang. cell array), elementami tablicy blokowej są bloki, w których mogą być przechowywane wartości różnych typów: teksty, macierze. Sposób tworzenia zmiennych tego typu jest podobny do tworzenia zwykłych macierzy, jedyną różnicą jest konieczność użycia nawiasów klamrowych zamiast nawiasów kwadratowych.
Załóżmy, że zostały zdefiniowane macierze **A**, **B**, **C**, **D**. Należy zdefiniować tablicę blokową (o nazwie np. **x**) o wymiarach 2x2 której elementami będą te macierze. Zmienną można utworzyć wykonując polecenie:
>> **x = {A B ; C D}**
Żądany element tablicy blokowej wskazuje się w podobny sposób jak element macierzy, indeksy elementu należy wskazać w nawiasach klamrowych. Aby odwołać się do elementu w pierwszym wierszu i pierwszej kolumnie należy wykonać polecenie:
>> **x{1,2} % wyświetlona zostanie macierz A**

net.biases – definiuje właściwości biasu neuronów sieci, jest tablicą blokową o wymiarze **net.numLayers** – jeżeli neurony *i*-tej warstwy sieci posiadają bias to odwołanie: **net.biases{i}** daje dostęp do informacji np.: o ilość neuronów (**size**), możliwe jest również ustalenie sposobu inicjalizacji wartości bias-ów (**initFcn**) czy metod wykorzystywanych do zmiany wartości bias-ów w trakcie uczenia (**learn**, **learnFcn**),

net.inputWeights – definiuje właściwości wejść sieci, jest tablicą blokową o wymiarze **net.numLayers** x **net.numInputs** – jeżeli warstwa *i*-ta otrzymuje na wejście sygnały z *j*-tego źródła wejściowego to odwołanie: **net.inputWeights{i,j}** daje dostęp do informacji np.: o rozmiarze macierzy wag (**size**), możliwe jest również ustalenie sposobu inicjalizacji wartości wag (**initFcn**) czy metod wykorzystywanych do zmiany wartości wag w trakcie uczenia (**learn**, **learnFcn**),

net.layerWeights – jest tablicą blokową o wymiarze **net.numLayers** x **net.numLayers** – jeżeli warstwa *i*-ta otrzymuje na wejście sygnały z *j*-tej warswy to odwołanie: **net.layerWeights{i,j}** daje dostęp do informacji np.: o rozmiarze macierzy wag (**size**), możliwe jest również ustalenie sposobu inicjalizacji wartości wag (**initFcn**) czy metod wykorzystywanych do zmiany wartości wag w trakcie uczenia (**learn**, **learnFcn**).

Przykład 2.

Należy uściślić definicję zmiennej z przykładu 1 uwzględniając następujące informacje:

- sieć ma dwa wejścia (na każde z wejść podawane są sygnały o wartościach z przedziału [0 1]),
- na pierwszej warstwie znajdują się dwa neurony, a na drugiej jeden neuron,
- funkcją aktywacji wszystkich neuronów sieci jest funkcja sigmoidalna.

>> **net.inputs{1}.size = 2;**	w przykładzie poprzednim zdefiniowane zostało tylko jedno źródło (jeden wektor) sygnałów wejściowych, przypisanie **net.inputs{1}.size = 2;** ustala rozmiar tego wektora na 2 (sieć może więc otrzymywać dwa sygnały wejściowe)
>> **net.inputs{1}.range = [0 1; 0 1];**	obydwa sygnały wejściowe mogą przyjmować wartości z przedziału [0 1]
>> **net.layers{1}.size = 2;**	na pierwszej warstwie sieci znajdują się dwa neurony
>> **net.layers{2}.size = 1;**	na drugiej warstwie sieci znajduje się jeden neuron
>> **net.layers{1}.transferFcn = 'logsig';** >> **net.layers{2}.transferFcn = 'logsig';**	funkcją aktywacji neuronów pierwszej i drugiej warstwy jest funkcja sigmoidalna

4.1.3. Znaczenie składników opisujących podstawowe operacje wykonywane przez sieć:

(podpunkt *functions* widoczny podczas wyświetlania wartości zmiennej reprezentującej sieć neuronową)

net.initFcn – definiuje sposób inicjalizacji sieci - metodę wyznaczania początkowych wartości wag i bias-ów neuronów sieci (inicjalizacja jest przeprowadzana po wywołaniu polecenia **init** – patrz opis w punkcie 2.1). Użytkownik może napisać własną funkcję inicjalizacyjną lub skorzystać ze standardowej funkcji '**initlay**'. Funkcja '**initlay**' korzysta z metod inicjalizacyjnych przypisanych poszczególnym warstwom sieci. Funkcję inicjalizacyjną dla *i*-tej warstwy można wybrać ustalając wartość parametru **net.layers{i}.initFcn**. Podobnie jak w przypadku funkcji inicjalizacyjnej dla całej sieci, użytkownik może napisać własną funkcję inicjalizacyjną dla warstwy sieci lub wykorzystać standardowe funkcje '**initnw**' lub '**initwb**'. Funkcja '**initnw**' wykorzystuje metodę Nguyen'a-Widrow'a, a funkcja '**initwb**' wykorzystuje metod inicjalizacyjne przypisane parametrom **net.inputWeights{i,j}.initFcn** czy **net.layerWeights{i,j}.initFcn** (parametrom tym można przypisać funkcje: '**initzero**' (wagi i bias-y otrzymują wartości zero), '**rands**' (wagi i bias-y otrzymują wartości losowe), '**randnr**', '**randnc**' (wagi i bias-y otrzymują znormalizowane wartości losowe)).

Przykład 3.

Sieć zdefiniowana w przykładzie 1 powinna inicjalizować wagi i biasy pierwszej warstwy wartościami losowymi a drugiej warstwy wartościami równymi zero.

>> **net.initFcn = 'initlay';**	inicjalizację wag wykonają funkcje inicjalizacyjne odpowiednich warstw sieci (funkcje przypisane do **net.layers{i}.initFcn**)
>> **net.layers{1}.initFcn = 'initwb';** >> **net.layers{2}.initFcn = 'initwb';**	funkcje inicjalizacyjne odpowiednich warstw sieci wykorzystają metody przypisane parametrom **net.inputWeights{1,1}.initFcn** i **net.layerWeights{1,1}.initFcn**
>> **net.inputWeights{1,1}.initFcn = 'rands';**	wagi (i bias-y) pierwszej warstwy będą inicjalizowane losowo
>> **net.layerWeights{2,1}.initFcn = 'initzero';**	wagi (i bias-y) drugiej warstwy otrzymają wartości zero

net.performFcn – definiuje metodę wyznaczania wskaźnika określającego stopień „nauczenia” sieci, dostępne w toolbox-ie metody wykorzystują do obliczeń macierz zawierającą wartości błędów uczenia dla każdej z par uczących. Składnikowi **net.performFcn** można przypisać jedną z metod:

Metoda	Nazwa	Opis
mae	Mean absolute error	średnia wartości bezwzględnych błędów sieci
mse	Mean squared error	średnia kwadratów błędów sieci
msereg	Mean squared error with regularization	obliczane są dwie wartości: a) średnia kwadratów błędów sieci oraz b) średnia kwadratów wag i bias-ów sieci. Wskaźnik określający stopień nauczenia sieci jest wyznaczany jako suma ważona wartości a) i b) (wagę dla wartości a) określa **net.performParam.ratio**, waga wartości b) wynosi: 1-**net.performParam.ratio**)
sse	Sum squared error	suma kwadratów błędów sieci

Przykład 4.

Wskaźnikiem określającym stopień „nauczenia”, dla sieci zdefiniowanej w przykładzie 1, powinna być suma kwadratów błędów.

```
>> net.performFcn = 'sse';
```

net.adaptFcn i net.trainFcn

Po zdefiniowaniu zmiennej reprezentującej sieć, określeniu jej architektury i zainicjowaniu wag i bias-ów sieci można przejść do uczenia.

W bibliotece *Neural Network Toolbox* dostępne są dwa style uczenia sieci:

- uczenie ‘*incremental training*’ (wzorce uczące prezentowane są sieci w określonej kolejności, a modyfikacja wag i bias-ów następuje po prezentacji pojedynczego wzorca),
- uczenie ‘*batch training*’ (modyfikacja wag i bias-ów następuje po prezentacji całego zbioru wzorców uczących).

Wybór stylu uczenia zależy od definicji wzorców uczących. Załóżmy, że należy nauczyć sieć działania bramki logicznej AND, tzn. dla sygnałów wejściowych (1, 1) odpowiedź na wyjściu sieci powinna wynosić 1, dla sygnałów (1, 0), (0, 1) i (0, 0) – odpowiedź powinna wynosić 0.

W przypadku uczenia typu '*incremental training*' wzorce uczące (wektory sygnałów wejściowych i odpowiedzi sieci) muszą być zdefiniowane jako tablice blokowe, tzn.:

>> **x = { [1; 1], [1; 0], [0; 1], [0; 0] }**

>> **t = { [1], [0], [0], [0] }**

W przypadku uczenia typu '*batch training*' wzorce uczące muszą być zdefiniowane jako macierze (kolejne kolumny macierzy odpowiadają kolejnym wzorcom uczącym), tzn:

>> **x = [1 1 0 0; 1 0 1 0]**

>> **t = [1 0 0 0]**

Proces uczenia sieci reprezentowanej przez zmienną np. **net** można zainicjować pisząc jedno z podanych poniżej poleceń:

>> **net.adapt**,

>> **net.train**.

W przypadku wywołania polecenia **adapt** możliwe jest uczenie typu '*incremental training*' oraz '*batch training*' (wybór typu zależy od definicji wzorców uczących), w przypadku polecenia **train** możliwe jest tylko uczenie typu '*batch training*' (niezależnie od definicji wzorców uczących).

Określoną metodę uczenia sieci wykorzystywaną po wywołaniu polecenia **adapt** można wskazać definiując parametr **net.adaptFcn**, a w przypadku polecenia **train** metodę uczenia należy wskazać definiując **net.trainFcn**.

net.adaptFcn – definiuje metodę uczenia sieci. Proces uczenia można zainicjować poleceniem **net.adapt**, polecenie to wywołuje odpowiednią metodę uczenia (wskazaną w parametrze **net.adaptFcn**) z parametrami określonymi przez **net.adaptParam**. W toolbox-ie dostępna jest metoda '**trains**', która może być przypisana parametrowi **net.adaptFcn**. Metoda ta modyfikuje wagi zgodnie algorytmami wskazanymi w parametrach **net.inputWeights{i,j}.learnFcn** i **net.layerWeights{i,j}.learnFcn**, a biasy zgodnie z algorytmami wskazanymi w **net.biases{i}.learnFcn**.

W poniższej tabelce zestawione zostały wybrane metody, które mogą być przypisane składnikom **net.inputWeights{i,j}.learnFcn, net.layerWeights{i,j}.learnFcn** i **net.biases{i}.learnFcn**:

Metoda	**Nazwa**	**Opis**
learnp	Perceptron weight/bias learning function	Metoda uczenia „z nauczycielem”, modyfikacja wag i bias-ów odbywa się zgodnie z *regułą perceptronu*.
learnpn	Normalized perceptron weight/bias learning function	Metoda wykorzystuje zmodyfikowaną wersję *reguły perceptronu*, w której przed wyznaczeniem korekty wag, wektor sygnałów wejściowych poddawany jest normalizacji.
learnwh	Widrow-Hoff weight/bias learning function	Metoda uczenia „z nauczycielem”, aktualizacja wag następuje zgodnie z metodą *Widrowa-Hoffa* (jest również nazywana *regułą delta* i *regułą najmniejszych kwadratów*). Metoda ta jest uogólnieniem *reguły perceptronu*.
learnh	Hebb weight learning rule	Metoda uczenia „bez nauczyciela” oparta na *regule Hebba*.
learnhd	Hebb with decay weight learning rule	Metoda wykorzystuje zmodyfikowaną wersję *reguły Hebba*, korekta wag wyznaczana zgodnie z *regułą Hebba* jest dodatkowo zmniejszana o tak zwany współczynnik zapominania.
learnk	Kohonen weight learning function	Metoda uczenia „bez nauczyciela” typu konkurencyjnego, oparta na *regule Kohonena*.
learngd	Gradient descent weight/bias learning function	Metoda uczenia „z nauczycielem”, proces uczenia (modyfikacja wag i bias-ów) opiera się na minimalizacji funkcji opisującej aktualną wartość błędu „nauczenia” sieci. Metoda wykorzystuje algorytm *największego spadku*. (korekta wag i bias-ów następuje w kierunku przeciwnym do kierunku wskazywanego przez gradient minimalizowanej funkcji).
learngdm	Gradient descent w/ momentum weight/bias learning function	Metoda wykorzystuje zmodyfikowaną wersję algorytmu *największego spadku* z tzw. *momentem* (*momentum*). Podczas wykonywania korekty wag i bias-ów uwzględniania jest informacja o poprzedniej korekcie (niezależnie od aktualnej wartości gradientu minimalizowanej funkcji).

Dodatkowo dostępne są również funkcje: **learncon**, **learnis**, **learnos**, **learnlv1**, **learnlv2**, **learnsom**.

Wybór metody uczenia sieci (przypisanie parametrowi **net.adaptFcn** określonej funkcji) powoduje automatyczne ustawienie parametrów (**net.adaptParam**) niezbędnych do funkcjonowania tej metody. Podstawowym parametrem, który wymaga ustawienia jest **net.adaptParam.passes**. Parametr ten określa maksymalną liczbę prezentacji wzorców uczących w trakcie uczenia. Domyślnie otrzymuje on wartość 1 – co oznacza, że po zainicjowaniu uczenia (poleceniem **adapt**) wzorce uczące zostaną tylko raz zaprezentowane.

Przykład 5.

Sieć zdefiniowana w przykładzie 1 powinna być uczona zgodnie z algorytmem *największego spadku*. Wzorce uczące mogą być zaprezentowane do 10 razy po wywołaniu polecenia **adapt**.

>> **net.adaptFcn = 'trains';**	metody uczenia zostaną określone dla każdej z warstw sieci oddzielnie
>> **net.inputWeights{1,1}.learnFcn = 'learngd';** >> **net.layerWeights{2,1}.learnFcn = 'learngd';**	wagi pierwszej i drugiej warstwy sieci będą modyfikowane zgodnie z algorytmem *największego spadku*.
>> **net.biases{1}.learnFcn = 'learngd';** >> **net.biases{2}.learnFcn = 'learngd';**	bias-y pierwszej i drugiej warstwy sieci będą modyfikowane zgodnie z algorytmem *największego spadku*.
>> **net.adaptParam.passes = 10;**	

net.trainFcn – definiuje metodę uczenia sieci. Proces uczenia można zainicjować poleceniem **net.train**, polecenie to wywołuje odpowiednią metodę uczenia (wskazaną w parametrze **net.trainFcn**) z parametrami określonymi przez **net.trainParam**. Składnikowi **net.trainFcn** można przypisać jedną z metod:

Metoda	Nazwa*	Opis
trains		Pozwala na wybór metody uczenia oddzielnie dla każdej z warstw sieci – patrz opis parametru **net.adaptFcn**
traingd	Gradient descent backpropagation	Zmodyfikowana wersja metody **learngd** dla uczenia typu '*batch training*', metoda wykorzystuje algorytm n*ajwiększego spadku*.
traingdm	Gradient descent with momentum backpropagation	Zmodyfikowana wersja metody **learngdm** dla uczenia typu '*batch training*', metoda jest oparta na algorytmie *największego spadku z tzw. momentem*.
traingda	Gradient descent with adaptive learning rate backpropagation	Metoda jest oparta na algorytmie *największego spadku z adaptacyjnym doborem współczynnika uczenia*.
traingdx	Gradient descent with momentum & adaptive learning rate. backpropagation	Metoda jest oparta na algorytmie *największego spadku z tzw. momentem i adaptacyjnym doborem współczynnika uczenia*.
trainbfg	BFGS quasi-Newton backpropagation	Metoda jest oparta na algorytmie *Broydena Fletchera Goldbara Shamio* - BFGS (metoda quasi newtonowska z aproksymacji odwrotności hesjanu).
trainlm	Levenberg-Marquardt backpropagation	Metoda wykorzystuje algorytm Newtona z regularyzacją Levenberga-Marquardta do aproksymacji hesjanu
traincgf	Conjugate gradient backpropagation with Fletcher-Reeves updates	Metoda jest oparta na algorytmie *gradientów sprzężonych* (współczynnik sprzężenia obliczany według reguły Fletchera-Reevesa)
traincgp	Conjugate gradient backpropagation with Polak-Ribiere updates	Metoda jest oparta na algorytmie *gradientów sprzężonych* (współczynnik sprzężenia obliczany według reguły Polaka-Ribierea)

traincgb	Conjugate gradient backpropagation with Powell-Beale restarts	Metoda jest oparta na algorytmie *gradientów sprzężonych* (dodatkowa funkcja testu określająca konieczność restartu kierunku poprawy na kierunek przeciwny do gradientu)
trainscg	Scaled conjugate gradient backpropagation	Metoda jest oparta na algorytmie *gradientów sprzężonych* (bez minimalizacji w kierunku w każdej iteracji)
trainrp	RPROP backpropagation	Metoda jest oparta na algorytmie *Riedmillera i Brauna* (zwanym RPROP od ang. Resilient backPROPagation)

*Jeżeli w nazwie metody występuje 'backpropagation', to składniki gradientu minimalizowanej funkcji wyznaczane są w oparciu o algorytm *propagacji wstecznej* (*backpropagation*).

Dodatkowo, w toolbox-ie dostępne są również funkcje: **trainbr**, **trainoss**.

Wybór metody uczenia sieci (przypisanie parametrowi **net.trainFcn** określonej funkcji) powoduje automatyczne ustawienie parametrów (**net.trainParam**) niezbędnych do funkcjonowania tej metody. Podstawowym parametrem, który wymaga ustawienia jest **net.trainParam.epochs**. Parametr ten określa maksymalną liczbę prezentacji wzorców uczących w trakcie uczenia. Domyślnie otrzymuje on wartość 1 – co oznacza, że po zainicjowaniu uczenia (poleceniem **train**) wzorce uczące zostaną tylko raz zaprezentowane.

Przykład 6.

Sieć zdefiniowana w przykładzie 1 powinna być uczona zgodnie z algorytmem *Levenberga-Marquardta*. Wzorce uczące mogą być zaprezentowane do 10 razy po wywołaniu polecenia **train**.

>> **net.trainFcn = 'trainlm';**	Uczenie przeprowadzane w oparciu o metodę *Levenberga-Marquardta*.
>> **net.trainParam.epochs = 10;**	

4.1.4 Znaczenie składników zawierających wagi i bias-y neuronów sieci:

(podpunkt ***weight and bias values*** widoczny podczas wyświetlania wartości zmiennej reprezentującej sieć neuronową)

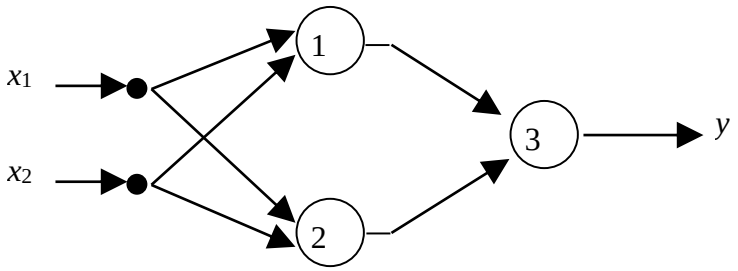
net.IW – zawiera wagi warstw sieci otrzymujących na wejścia sygnały z wejść sieci, jest tablicą blokową o wymiarze **net.numLayers** x **net.numInputs** – jeżeli warstwa *i*-ta otrzymuje na wejście sygnały z *j*-tego źródła wejściowego to odwołanie: **net.IW{i,j}** daje wagi neuronów tej warstwy sieci,

net.LW – jest tablicą blokową o wymiarze **net.numLayers** x **net.numLayers** – jeżeli warstwa *i*-ta otrzymuje na wejście sygnały z *j*-tej warstwy to odwołanie: **net.LW{i,j}** daje wagi neuronów tej warstwy sieci,

net.b – jest tablicą blokową o wymiarze **net.numLayers** – jeżeli neurony *i*-tej warstwy sieci posiadają bias to odwołanie: **net.b{i}** daje dostęp do informacji o wartościach bias-ów neuronów tej warstwy sieci.

Przykład 7.

W przykładach 1 i 2 została zdefiniowana zmienna **net** reprezentująca sieć o architekturze przedstawionej na poniższym rysunku:



Należy przypisać odpowiednim składnikom zmiennej podane poniżej wartości wag i bias-ów:

	Neuron 1 (górny z 1 warstwy)	Neuron 2 (dolny z 1 warstwy)	Neuron 3 (z 2 warstwy)
waga odpowiadająca pierwszemu wejściu	1	2	1
waga odpowiadająca drugiemu wejściu	0	1	3
bias	2	3	2

Wagi każdej warstwy neuronów są przechowywane w postaci macierzy której liczba wierszy odpowiada licznie neuronów a liczba kolumn liczbie wejść sieci (wagi związane z *i*-tym neuronem stanowią *i*-ty wiersz tej macierzy). Bias-y są zapisywane w postaci wektorów kolumnowych.

Wagi pierwszej warstwy neuronów są przechowywane w składowej **net.IW{1,1}** (pierwsza warstwa sieci otrzymuje na wejściu sygnały z pierwszego źródła), wagi neuronów drugiej warstwy są przechowywane w **net.LW{2,1}** (druga warstwa sieci otrzymuje na wejściu sygnały wyjściowe pierwszej warstwy).

Ostatecznie, przykładowe zadanie rozwiązują polecenia:

```
>> net.IW{1,1} = [1 0; 2 1];
>> net.LW{2,1} = [1 3];
>> net.b{1} = [2; 3];
>> net.b{2} = [2];
```

4.2. Specjalizowane funkcje do tworzenia konkretnych rodzajów sieci - uzupełnienie

Zmienną reprezentującą sieć o dowolnej strukturze można utworzyć korzystając z polecenia **network** (omówionego w poprzednim podpunkcie). Definiowanie struktury takiej sieci jest jednak pracochłonne – wymaga ustalania wielu składników zmiennej sieciowej. Sieci o standardowych architekturach (np. perceptron jednowarstwowy czy jednokierunkowa sieć wielowarstwowa), można również tworzyć posługując się specjalizowanymi funkcjami, np.: **newp** (perceptron jednowarstwowy), **newff** (jednokierunkowa sieć wielowarstwowa), **newlin** (jednowarstwowa sieć zbudowana z neuronów liniowych), **newrb** (sieć radialna RBF ang. Radial Basis Functions), **newc** (sieć samoorganizująca się na zasadzie współzawodnictwa), **newelm** (sieć rekurencyjna Elmana), **newhop** (sieć rekurencyjna Hopfielda), ...

4.2.1. Perceptron jednowarstwowy (**newp**)

Perceptrony mogą być wykorzystywane do rozwiązywania problemów liniowoseparowalnych. Perceptron jednowarstwowy jest siecią złożoną z neuronów z bipolarną lub unipolarną funkcją skoku jako funkcją aktywacji. Najpopularniejszą metodą uczenia perceptronu jest tzw. *reguła perceptronu*.

Do zaprojektowania struktury sieci perceptronowej można wykorzystać funkcję **newp**:

```
>> net = newp(PR,S,TF,LF);
```

Parametrami wejściowymi funkcji są:

Parametr	Opis
PR	macierz o ilości wierszy odpowiadającej liczbie wejść sieci i liczbie kolumn równej dwa (w pierwszej kolumnie powinna znajdować się minimalna wartość sygnału wejściowego a w drugiej jego maksymalna wartość),
S	liczba neuronów perceptronu, *(wartość domyślna: 1)*,
TF	funkcja aktywacji neuronów sieci, *(wartość domyślna: 'hardlim')*,
LF	metoda uczenia, *(wartość domyślna: 'learnp')*.

Funkcja zwraca zmienną reprezentującą odpowiednią sieć neuronową.

Jeżeli podczas wywołania nie zostaną podane wartości parametrów **S**, **TF** czy **LF**, przyjęte zostaną (podane w powyższej tabelce) wartości domyślne tych parametrów.

Funkcja **newp** tworzy zmienną reprezentującą sieć przypisując odpowiednie wartości jej składnikom, np.:

net.inputs{1}.range = PR;

net.layers{1}.size = S;

net.layers{1}.transferFcn = TF;

net.inputWeights{1,1}.learnFcn = LF;

net.biases{1}.learnFcn = LF;

Dodatkowo:

- wagi i bias-y neuronów będą inicjalizowane zerami
  (tzn.: **net.biases{1}.initFcn = 'initzero'**; **net.inputWeights{1,1}.initFcn = 'initzero'**;),
- adaptacja będzie przeprowadzana przy pomocy funkcji 'trains', a trenowanie 'trainc'
  (tzn.: **net.adaptFcn = 'trains'**; **net.adaptFcn = 'trainc'**;),
- stopień „nauczenia” sieci określa wskaźnik 'mae'
  (tzn.: **net.performFcn = 'mae'**;).

Przykład 8

Należy utworzyć zmienną reprezentującą perceptron o dwóch wejściach z jednym neuronem na warstwie. Sygnały podawane na wejścia neuronu osiągają wartości z przedziału [-1 1]. Funkcją aktywacji neuronu jest bipolarna funkcja skoku.

>> **net = newp([-1 1; -1 1], 1, 'hardlims');**

1.2.1 Wielowarstwowa sieć jednokierunkowa (newff)

Polecenie **newff** tworzy sieć zbudowaną z wielu warstw, w której sygnały przesyłane są w jednym kierunku od wejścia do wyjścia. Do uczenia takiej sieci wykorzystywany jest algorytm *propagacji* wstecznej – w związku z tym zarówno funkcje aktywacji neuronów jak i wskaźnik określający stopień „nauczenia" sieci muszą być funkcjami ciągłymi (niedozwolone są więc np. funkcje aktywacji 'hardlim' i 'hardlims' i wskaźnik 'mae').

Aby utworzyć zmienną reprezentującą taką sieć neuronową należy wykonać polecenie:

>> **net = newff(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF);**

Parametrami wejściowymi funkcji są:

Parametr	Opis
PR	macierz o ilości wierszy odpowiadającej liczbie wejść sieci i liczbie kolumn równej dwa (w pierwszej kolumnie powinna znajdować się minimalna wartość sygnału wejściowego a w drugiej jego maksymalna wartość),
S1,S2...SN1	liczba neuronów na kolejnych warstwach sieci, **S1** – ilość neuronów pierwszej warstwy, **S2** – ilość neuronów drugiej warstwy, ..,
TF1,TF2...TFN1	funkcje aktywacji neuronów kolejnych warstw sieci, **TF1** – funkcja aktywacji neuronów pierwszej warstwy, **TF2** – funkcja aktywacji neuronów drugiej warstwy, .., *(wartość domyślna: 'tansig')*,
BTF	metoda wykorzystywana podczas uczenia sieci poleceniem **train**, można używać metod opisanych w punkcie 1.1.3, w których nazwach występuje słowo 'backpropagation' *(wartość domyślna: 'trainlm'),*
BLF	metoda uczenia wykorzystywana podczas uczenia sieci poleceniem **adapt**, *(wartość domyślna: 'learngdm'),*
PF	wskaźnik określający stopień „nauczenia" sieci, *(wartość domyślna: 'mse'),*

Funkcja zwraca zmienną reprezentującą odpowiednią sieć neuronową.

Jeżeli podczas wywołania nie zostaną podane wartości ostatnich czterech parametrów, przyjęte zostaną (podane w powyższej tabelce) wartości domyślne tych parametrów.

Podobnie jak funkcja **newp**, polecenie **newff** tworzy zmienną reprezentującą sieć przypisując odpowiednie wartości jej składnikom (wykorzystując podane wartości parametrów wejściowych). Dodatkowo, metoda Nguyen'a-Widrow'a ('initnw') jest ustawiana jako domyślna funkcja inicjująca wagi i bias-y sieci. Domyślną metodą adaptacji jest 'trains'.

Przykład 9

Należy utworzyć zmienną reprezentującą dwuwarstwową sieć o dwóch wejściach. Na pierwszej warstwie sieci znajdują się dwa neurony, na drugiej warstwie jeden neuron. Funkcją aktywacji neuronów sieci jest funkcja tangensoidalna. Sygnały podawane na wejścia neuronu osiągają wartości z przedziału [-1 1]. Uczenie sieci będzie przeprowadzane metodą Levenberga-Marquardta.

```
>> net = newff([-1 1; -1 1],[2 1],{'tansig' 'tansig'}, 'trainlm');
```

5. UCZENIE SIECI

Przed rozpoczęciem uczenia sieci należy:

- przygotować zbiór wzorców uczących,
  w przypadku uczenia '*bez nauczyciela*' zbiór ten tworzą wektory sygnałów wejściowych,
  w przypadku uczenia 'z *nauczycielem*' należy przygotować dwa zbiory: zbiór wektorów z sygnałami wejściowymi i zbiór odpowiadających im wektorów z sygnałami wyjściowymi,
  sposób definiowania wzorców uczących zależy od typu uczenia ('*incremental training*' lub '*batch training*'), został przedstawiony w punkcie 1.1.3 (podczas omawiania składników **net.adaptFcn** i **net.trainFcn),**
- ustawić parametry uczenia, np. maksymalną liczbę prezentacji wzorców uczących w trakcie uczenia.

Następnie można już zainicjować proces uczenia poleceniem **adapt** lub **train.**

5.1 Wykorzystanie metody adapt

Aby zainicjować proces uczenia metodą **adapt** należy wykonać polecenie:

>> **[net,Y,E] = adapt(net,P,T)**

Parametrami wejściowymi funkcji są:

Parametr	Opis
net	zmienna reprezentująca sieć neuronową
P	tablica blokowa (uczenie typu '*incremental training*') lub macierz (uczenie typu '*batch training*') zawierająca wektory kolumnowe sygnałów wejściowych
T	tablica blokowa lub macierz zawierająca wektory kolumnowe żądanych sygnałów wyjściowych (niezbędna w przypadku uczenia *'z nauczycielem'*),

Parametrami wyjściowymi są:

Parametr	Opis
net	zmienna odpowiadająca 'nauczonej' sieci, zmienna ta zawiera zaktualizowane wartości wag i bias-ów (można je pobrać z odpowiednich składników zmiennej: **net.IW**, **net.LW**, **net.b**), załóżmy, że zmienna **net** reprezentuje sieć, dla której należy przeprowadzić proces uczenia, polecenie **adapt** można wywołać pisząc: >> **net2 = adapt(net, P, T)** (dostępne są dwie zmienne sieciowe, zmienna **net** odpowiada sieci w stanie początkowym (nie poddanej procesowi uczenia), zmienna **net2** odpowiada sieci 'nauczonej', zwykle nie jest istotne przechowywanie zmiennej reprezentującej stan początkowy sieci, polecenie **adapt** można również wywołać: >> **net = adapt(net, P, T)** (po wykonaniu polecenia zmienna **net** odpowiada sieci 'nauczonej')
Y	tablica blokowa lub macierz zawierająca wektory kolumnowe sygnałów wyjściowych sieci

E	tablica blokowa lub macierz zawierająca wartości błędów dla każdego z wyjść sieci

W wywołaniu polecenia można pominąć:

- parametr wejściowy **T**, jeżeli uczenie ma się odbywać '*bez nauczyciela*',
- parametry wyjściowe.

Przykład 10

Przeprowadzić proces uczenia perceptronu reprezentowanego przez zmienną utworzoną w przykładzie 8. Perceptron powinien realizować funkcję logiczną OR. Wzorce powinny zostać zaprezentowane 3 razy. Należy wyświetlić wartości sygnałów wyjściowych i błędów sieci oraz znalezione wartości wag i biasów.

>> **P = [1 1 -1 -1 ; 1 -1 1 -1];**	macierz zawiera 4 wektory sygnałów wejściowych; [1; 1], [1; -1], [-1; 1], [-1; -1]
>> **T = [1 1 1 -1]**	wektor zawiera żądane wartości sygnałów wyjściowych, tzn: wartość 1 dla pierwszych trzech wektorów wejściowych i wartość -1 dla ostatniego wektora
>> **net.adaptParam.passes = 3;**	wzorce będą prezentowane 3 razy
>> **[net, Y, E] = adapt(net, P, T)**	uczenie, wyświetlane są: zaktualizowana zmienna **net**, wektor sygnałów wyjściowych **Y** oraz wektor błędów **E**,
>>**net.IW{1,1}** >>**net.b{1}**	wyświetlane są wagi oraz bias

5.2 Wykorzystanie metody train

Aby zainicjować proces uczenia metodą **train** należy wykonać polecenie:

>> **[net,tr,Y,E] = train(net,P,T)**

Wywołanie jest więc bardzo podobne do wywołania polecenia **adapt**, pojawia się tylko jeden dodatkowy parametr wyjściowy **tr**. Parametr ten zawiera informacje dotyczące przebiegu procesu uczenia, np. wartość wskaźnika określającego stopień 'nauczenia' sieci.

Przykład 11

Przeprowadzić proces uczenia sieci reprezentowanej przez zmienną utworzoną w przykładzie 9. Sieć powinna realizować funkcję logiczną OR. Wzorce mogą zostać zaprezentowane maksymalnie 20 razy. Należy wyświetlić wartości sygnałów wyjściowych i błędów sieci oraz znalezione wartości wag i biasów.

>> **P = [1 1 -1 -1 ; 1 -1 1 -1];** **>> T = [1 1 1 -1]**	definicja analogiczna jak w przykładzie 10
>> **net.trainParam.epochs = 20;**	wzorce będą prezentowane maksymalnie 20 razy
>> **[net, tr, Y, E] = train(net, P, T)**	uczenie, wyświetlane są: zaktualizowana zmienna **net**, parametry uczenia **tr**, wektor sygnałów wyjściowych **Y** oraz wektor błędów **E**,
>>**net.IW{1,1}** **>>net.b{1}** **>>net.LW{2,1}** **>>net.b{2}**	wyświetlane są wagi i biasy neuronów pierwszej i drugiej warstwy

3 SYMULACJA DZIAŁANIA SIECI – UZUPEŁNIENIE

W każdej chwili można sprawdzić w jaki sposób sieć przetwarza sygnały wejściowe w wyjściowe. Do symulacji działania sieci należy wykorzystać funkcję **sim**. W najprostszej postaci funkcję tą można wywołać w poniższej formie:

>> **Y = sim(net,P)**

Parametrami wejściowymi funkcji są:

Parametr	Opis
net	zmienna reprezentująca testowaną sieć neuronową
P	tablica blokowa (uczenie typu '*incremental training*') lub macierz (uczenie typu '*batch training*') zawierająca wektory kolumnowe sygnałów wejściowych

Funkcja zwraca tablicę blokową lub macierz zawierająca wektory kolumnowe sygnałów wyjściowych.

Przykład 12

Przeprowadzić symulację działania sieci przygotowanej w przykładzie 10 dla:

a) sygnałów wejściowych odpowiadających funkcji logicznej,

b) zaburzonych wartości sygnałów z podpunktu a).

>> **y = sim(net, P)**	badane są wartości sygnałów na wyjściu sieci dla wzorców wykorzystywanych w trakcie uczenia (jeżeli sieć jest dobrze 'nauczona' to wektor **y** zawiera wartości [1 1 1 –1])
>> **y1 = sim(net, [1.1; 1.1])** >> **y2 = sim(net, [1.1; -1.1])** >> **y3 = sim(net, [-0.9; 0.9])** >> **y4 = sim(net, [-1.1; -0.9])**	niewielkie odchylenia wartości sygnałów wejściowych nie mają wpływu na wartość sygnału na wyjściu sieci