Podstawy sztucznej inteligencji

Sieć neuronowa jako aporoksymator

Kamil Rega

Contents

1. Wstęp

Tematem tej pracy jest implementacja aplikacji, która wykorzystuje sieć neuronową, wybrano zastosowanie mające na celu aproksymacje funkcji podanej jako wektor uczący. Sztuczne sieci neuronowe to matematyczny odpowiednik procesów zachodzących w ludzkim mózgu i układzie nerwowym. Ich struktura jest skomplikowana. Sieci neuronowe znalazły swoje zastosowanie w różnych dziedzinach. Za ich pomocą można analizować i przewidywać zmiany na giełdach, dokonywać diagnozy stanu pacjentów,jak i również po wprowadzeniu odpowiedniej ilości danych wejściowych rozpoznawać pismo czy obrazy. Podłożem do omówienia pod względem matematycznym sztucznych sieci jest budowa i właściwości ludzkiego mózgu i komórek nerwowych. Kora mózgowa zawiera 10^{10} komórek nerwowych i 10^{12} komórek glejowych. Impulsy komórek nerwowych posiadają częstotliwość do 100 Hz i posiadają czas trwania do 2 ms. Całą zaś moc obliczeniową mózgu można oszacować na 10^{18} operacji na sekundę. Sama sieć neuronowa odzwierciedla działanie układu nerwowego tylko w niewielkim stopniu. Funkcje sztucznych neuronów w stosunku do komórek nerwowych to tylko uproszczony model. Cały ludzki układ nerwowy składa się ze 100 mld komórek (neuronów). Każdy z neuronów biologicznych składa się z następujących części:

- ciała komórki (somy);
- dendrytów ‑ wprowadzają one informacje do neuronu;
- aksonu ‑ wyprowadza on informacje z neuronu;
- synapsy ‑ są to złącza nerwowe, odpowiadające za przekazywanie sygnałów innym neuronom

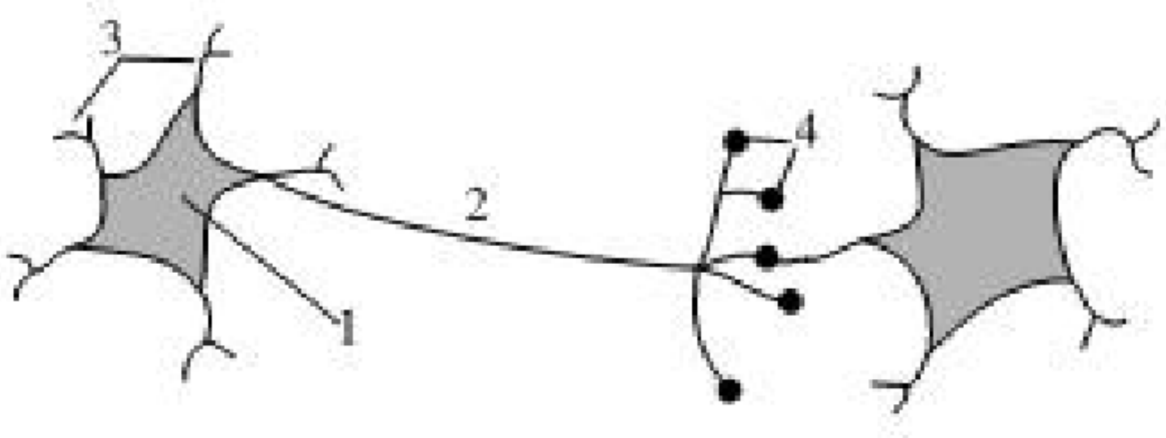

Figure 1 Schemat połączenia dwóch neuronów. Na rysunku poszczególne numery

1) Wprowadzenie informacji do neuronu poprzez dendryty;

2) Przetwarzanie sygnałów wewnątrz ciała komórki;

3) Wyprowadzenie informacji z neuronu przez akson;

4) Przekazanie pobudzenia do innych neuronów przez synapsy.

Na transmisję sygnałów składają się procesy elektrochemiczne. Synapsy mogą działać wzmacniająco lub osłabiająco na przekazywany sygnał. Jest on przesyłany tylko wtedy, gdy suma impulsów, które pobudzają i tych o charakterze hamującym przekracza założony próg.

2. Opis matematyczny sztucznego neuron

Sztuczny neuron składa się z dwóch podstawowych elementów:

- części sumującej
- części aktywującej (bloku aktywacji);

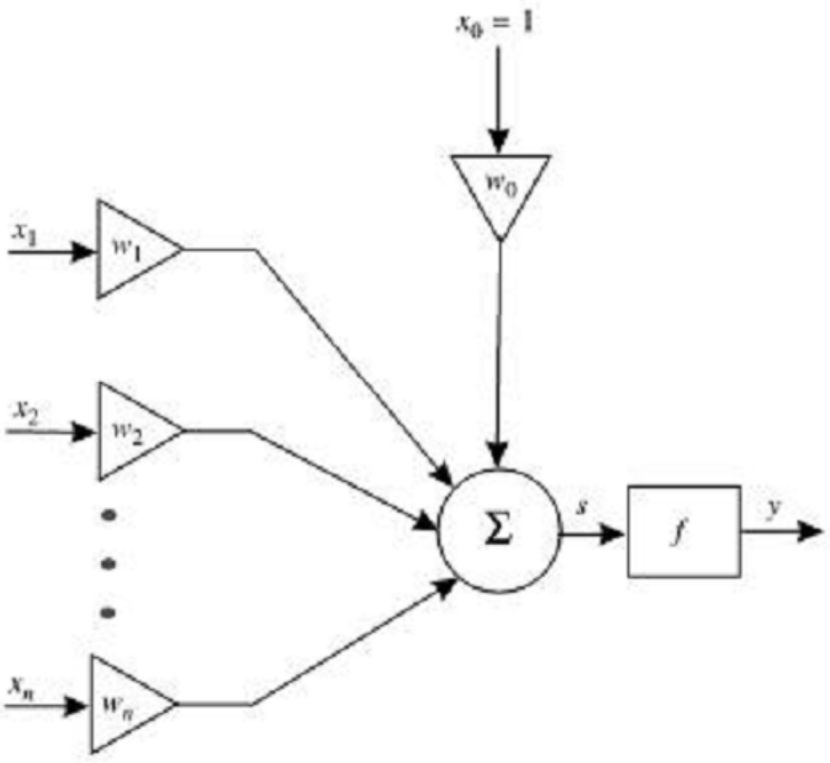

Figure 2Model sztucznego neuron

Działanie sztucznego neuronu opisuje poniższa zależność:

$$y = f(s)$$

gdzie:

$$s = \sum_{i=0}^{n} x_i w_i$$

Wytłumaczenie oznaczeń:

n - liczba wejść neuronu

x = $[x_i ... x_n]^T$ - wektor sygnałów wejściowych

w = $[w_1 ... w_n]^T$ - wektor wag

y - wyjście neuronu

w_0 - wartość progowa (bias)

Sygnały, które są wejściami neuronu zostają pomnożone przez przyporządkowane im wagi. To właśnie we współczynnikach wagowych tkwi wiedza neuronu. Otrzymane iloczyny są następnie sumowane. Wynikiem jest sygnał s, reprezentujący charakter liniowy neuronu. Musi on jeszcze zostać przetworzony przez funkcję aktywacji. Jest ona najczęściej nieliniowa, jednak może przyjmować różne postacie. Jej rodzaje wymieniono poniżej:

Figure 3 Liniowa funkcja aktywacji y = ax

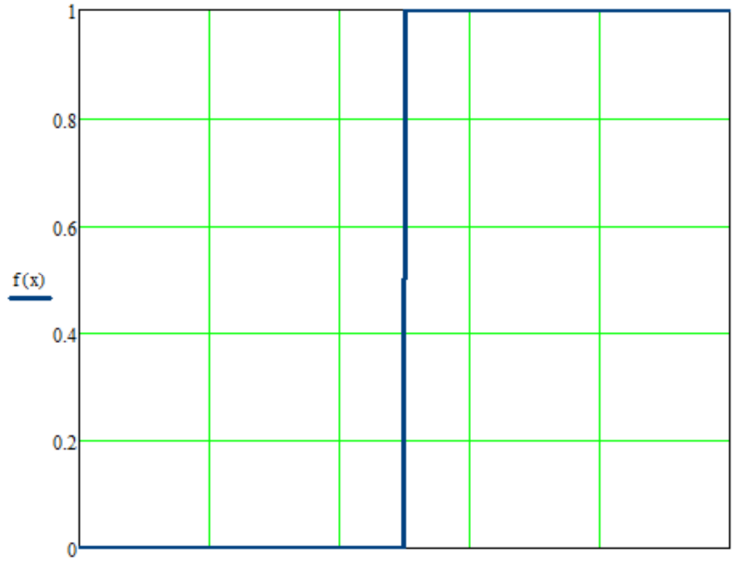

Figure 4 Progowa (unipolarna) funkcja aktywacji

$$y = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

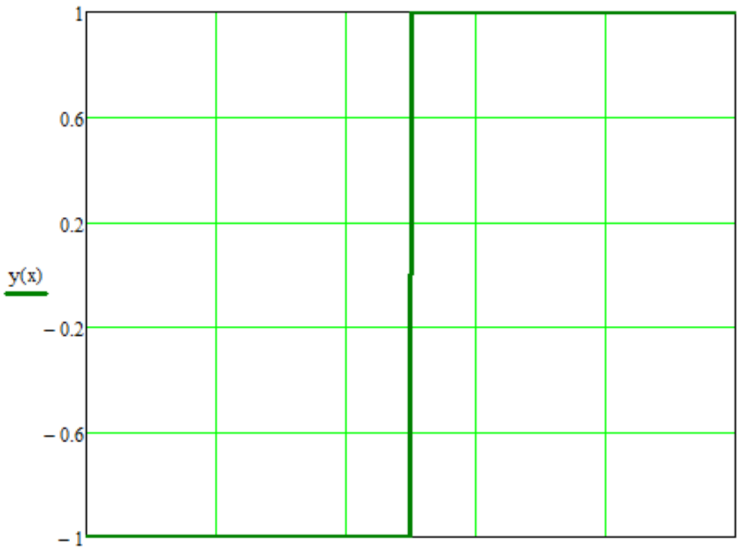

Figure 5Funkcja aktywacji signum (bipolarna)

$$y = \begin{cases} 1, & x > 0 \\ -1, & x \leq 0 \end{cases}$$

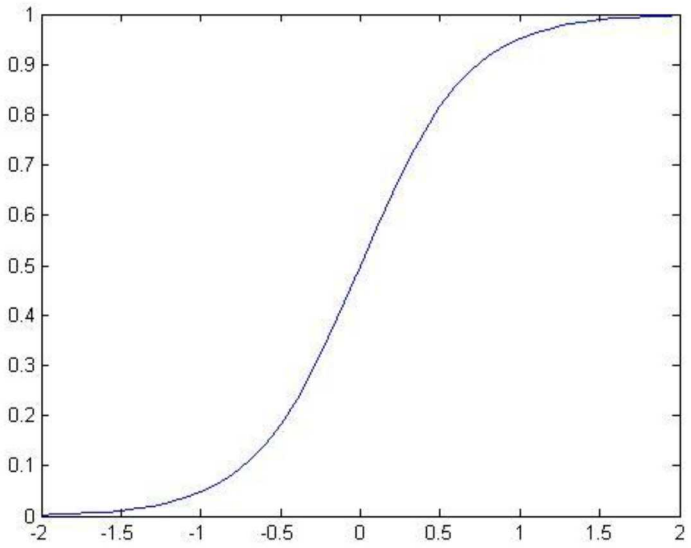

Figure 6 Funkcja aktywacji signum (bipolarna)

$$y = \frac{1}{1+ e^{-\beta x}} \text{ , } \beta > 0$$

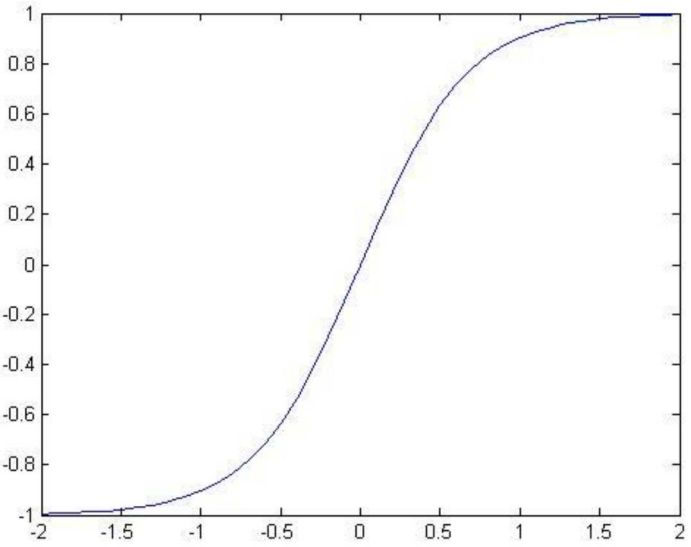

Figure 7Funkcja aktywacji tangensoidalna (bipolarna ciągła)

$$y = \frac{1 - e^{-\beta x}}{1+ e^{-\beta x}} \text{ , } \beta > 0$$

Sieci neuronowe posiadają właściwości adaptacyjne, dzięki którym można stworzyć strukturę i dobrać parametry odpowiednie do określonego zadania. Uczenie polega na takim doborze wag, dzięki któremu można odpowiednio przetworzyć sygnały wejściowe w wyjściowe. W każdym cyklu następuje adaptacja wag, która określona jest zależnością:

$$W_{ij}(k+1) = W_{ij}(k) + \Delta W_{ij}(k)$$

gdzie:

k ‑ numer cyklu

W_{ij}(k) - stara waga

W_{ij}((k + 1) - nowa waga, która łączy i-ty neuron z j-tym

Wyróżnić można trzy podstawowe typy uczenia sieci neuronowych:

a. uczenie pod nadzorem (z nauczycielem),

b. uczenie z krytykiem,

c. uczenie bez nadzoru ‑ samoorganizujące się.

3. Implementacja sieci neuronowej jako aproksymatora

W pracy postanowiono stworzyć sieć neuronową o niewielkim stopniu skomplikowania, a mianowicie 1 - 3 - 1. Sieć ta ma za zadanie jak najdokładniej przybliżyć fragment funkcji podanej jako wektor uczący. Uczenie sieci zrealizowano w oparciu o algorytm wstecznej propagacji błędów. Sieć posiada jeden neuron w warstwie wejściowej, trzy w warstwie ukrytej i jeden w wyjściowej. Strukturę sieci przedstawia poniższy rysunek:

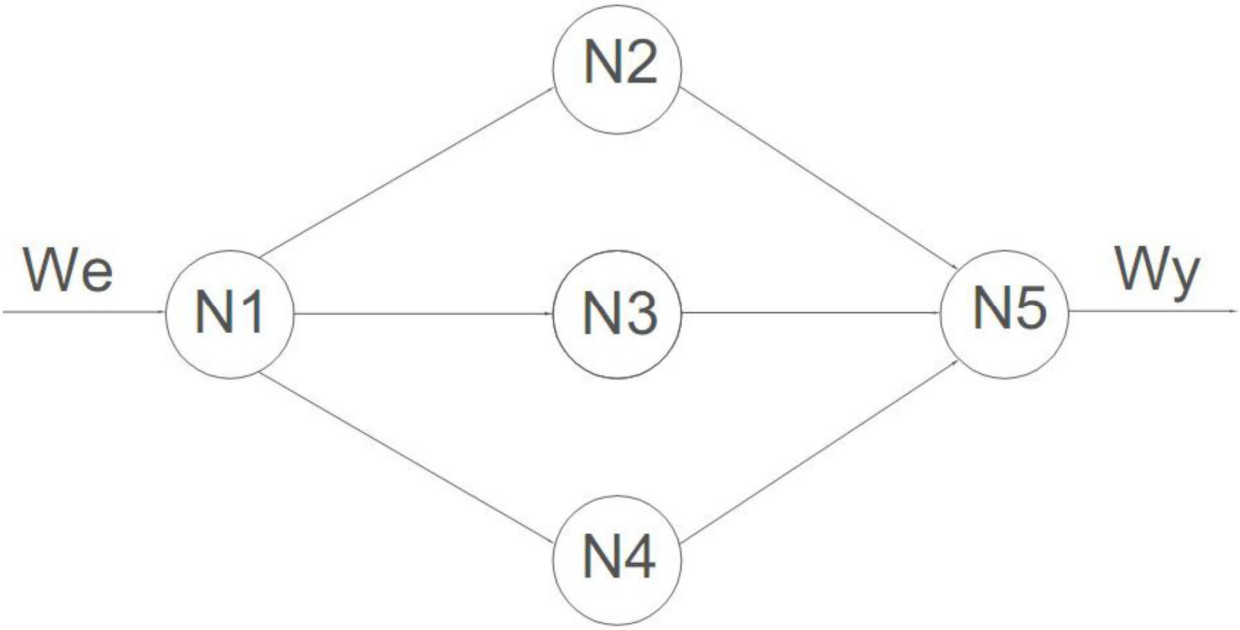

Figure 8 Sieć typu 1 – 3 – 1

Aplikacje zaimplementowano w języku skryptowym Python w wersji 2.7. W celu abstrakcyjnego przedstawienia neuronu stworzono klase neuron mająca nastepujace pola:

```
self.inputs_nr = inputs_nr     #ilość wejść do neurony
self.wages = list()            #wagi konkretnych wejść
self.out = list()              #wyjścia neuronu
self.bias = 0                 #bias
self.signal_out = 0            #sygnal po sumatorze
self.out_value = 0            #wartość wyjściowa neuronu
self.error = 0                #błąd neuronu
```

W celu odwzorowania struktury sieci napisano klasę neu_net, z nastepujacymi polami, których wartości należy podać w konstruktorze:

```
self.samples = list()         #argumenty wektora uczącego
self.values = list()          #wartości wektora uczącego
self.layers_nr                #liczba wartstw sieci
self.n = coeff                #wspolczynnik uczenia
self.epoch                    #liczba epok uczących
self.layers = list()          #lista przechowująca liste neuronów w poszczególnych warstwach
self.error = 0                #suma błędu
```

```
self.out_teached = list()    #lista przechowująca dane wyjściowe nauczonej sieci
self.epoch_n = list()        #numer epoki (potrzebny do wykresu błędu)
self.err_ep = list()         #lista przechowująca błąd w danej epoce uczącej
```

Do inicializacji kolejnych warstw użyto poniżej funkcji:

```
def initialize_layer(self, nr_warstwy, ilosc_neuronow, ilość_wejść_na_neuron, ilość_wyjść_na_neuron)
```

Każdy z neuronów inicjalizuje sie funkcją z poniższą sygnaturą:

```
def initialize_wages(self, wagi, bias)
```

Do stworzenia interfejsu graficznego wykorzystano bibliotekę Tkinter.

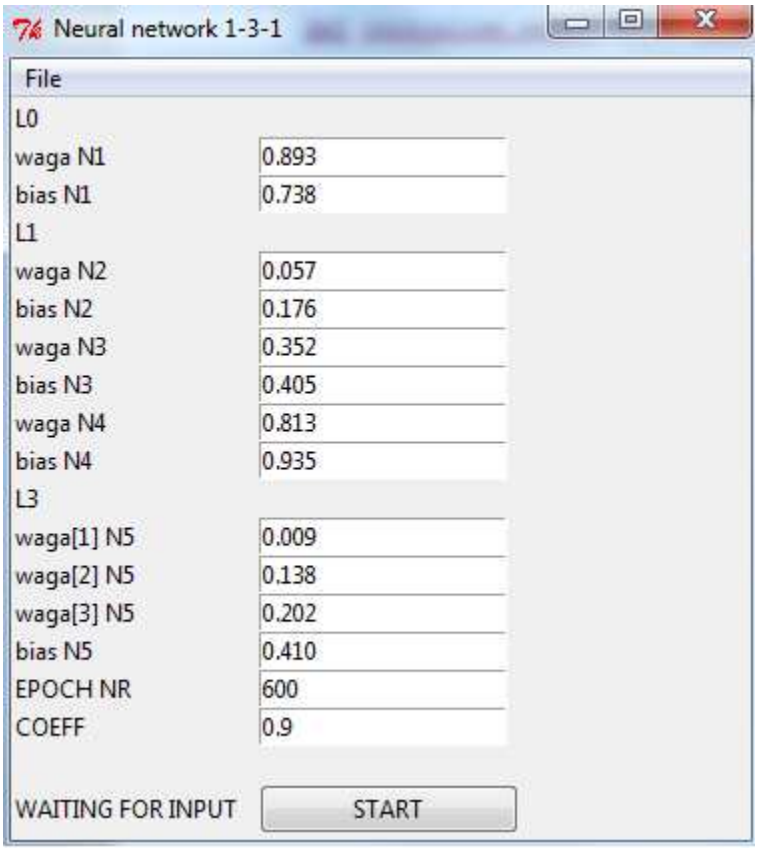

Figure 9 Grafinczy interfejs użytkownika

Po starcie aplikacji wartości wejściowe przyjmują wartości domyślne, przed procesem nauki (naciśnięciem przycisku START), można je dowolnie zmieniać. Wartości powinny zostać wpisane w formie zmiennoprzecinkowej, z wyjątkiem ilości epok uczących, która powinna być liczbą całkowitą. W przypadku dostarcznia wartości w złej postaci, użytkownik zostanie o tym poinformowany, poprzez „popup window". Następnym krokiem jest dostarczenie do aplikacji wektoru uczącego, poprzez wskazanie ścieżki do pliku z wartościami wektora uczącego (po kliknięciu w menu File). Po dostarczeniu wektora uczącego, należy wcisnąc przycisk START, co skutkuje rozpoczęciem procesu uczenia, a konkretnie wywołaniem poniższej funkcji:

```
def teach_network(self):
    teached = 0
    epoch = 0
    while teached == 0:
        #uczenie sieci
        index_count = 0
        for element in self.samples:
            #obliczenie wyjscia pierwszej warstwy (neuronu N1)
            self.calculate_perceptron_output(self.layers[0][0], [element])
            #obliczenie wyjscia drugiej warstwy
             #obliczenie wyjscia neuronu N2
            self.calculate_perceptron_output(self.layers[1][0], [self.layers[0][0].out_value])
            #obliczenie wyjscia neuronu N3
            self.calculate_perceptron_output(self.layers[1][1], [self.layers[0][0].out_value])
            #obliczenie wyjscia neuronu N4
            self.calculate_perceptron_output(self.layers[1][2], [self.layers[0][0].out_value])
            #obliczenie wyjscia warstwy trzeciej
            #obliczenie wyjscia neuronu N5
            self.calculate_perceptron_output(self.layers[2][0], [self.layers[1][0].out_value,
                                                                 self.layers[1][1].out_value,
                                                                 self.layers[1][2].out_value])
            #obliczenie bledu
            e = self.values[index_count] - self.layers[2][0].signal_out
            #wsteczna propagacja bledu
            #blad neuronu N5
            self.layers[2][0].error = e
            #propagacja bledu do warstwy drugiej
            #blad neuronu N2
            self.layers[1][0].error = ((self.layers[1][0].out_value *
                                        (1.0 - self.layers[1][0].out_value)) *
                                        self.layers[2][0].wages[0] * self.layers[2][0].error)
             #blad neuronu N3
             self.layers[1][1].error = ((self.layers[1][1].out_value *
                                        (1.0 - self.layers[1][1].out_value)) *
                                        self.layers[2][0].wages[1] * self.layers[2][0].error)
             #blad neuronu N4
             self.layers[1][2].error = ((self.layers[1][2].out_value *
                                       (1.0 - self.layers[1][2].out_value)) *
                                        self.layers[2][0].wages[2] * self.layers[2][0].error)
             #propagacja bledu do warstwy pierwszej
             # blad neuronu N1
             self.layers[0][0].error = ((self.layers[0][0].out_value *
                                        (1.0 - self.layers[0][0].out_value)) *
                                        ((self.layers[1][0].wages[0] * self.layers[1][0].error) +
                                         (self.layers[1][1].wages[0] * self.layers[1][1].error) +
                                         (self.layers[1][2].wages[0] * self.layers[1][2].error)))

             #obliczenie biasu neuronu N5 do kolejnego cyklu
             self.layers[2][0].bias += self.layers[2][0].error * self.n
             #obliczenie wagi1 neuronu N5 do kolejnego cyklu
             self.layers[2][0].wages[0] = (self.layers[2][0].wages[0] +
                                           (self.n * self.layers[2][0].error *
                                            self.layers[1][0].out_value))
             #obliczenie wagi2 neuronu N5 do kolejnego cyklu
             self.layers[2][0].wages[1] = (self.layers[2][0].wages[1] +
                                           (self.n * self.layers[2][0].error *
                                            self.layers[1][1].out_value))
             #obliczenie wagi3 neuronu N5 do kolejnego cyklu
             self.layers[2][0].wages[2] = (self.layers[2][0].wages[2] +
                                           (self.n * self.layers[2][0].error *
                                            self.layers[1][2].out_value))
             #obliczenie biasu neuronu N2 do kolejnego cyklu
             self.layers[1][0].bias += self.layers[1][0].error * self.n
             #obliczenie biasu neuronu N3 do kolejnego cyklu
             self.layers[1][1].bias += self.layers[1][1].error * self.n
             #obliczenie biasu neuronu N4 do kolejnego cyklu
             self.layers[1][2].bias += self.layers[1][2].error * self.n
             #obliczenie wagi neuronu N2 do kolejnego cyklu
             self.layers[1][0].wages[0] = (self.layers[1][0].wages[0] +
                                           (self.n * self.layers[1][0].error *
                                            self.layers[0][0].out_value))
             #obliczenie wagi neuronu N3 do kolejnego cyklu
```

```
            self.layers[1][1].wages[0] = (self.layers[1][1].wages[0] +
                                          (self.n * self.layers[1][1].error *
                                          self.layers[0][0].out_value))
            #obliczenie wagi neuronu N4 do kolejnego cyklu
            self.layers[1][2].wages[0] = (self.layers[1][2].wages[0] +
                                          (self.n * self.layers[1][2].error *
                                          self.layers[0][0].out_value))

            #obliczenie wagi neuronu N1 do kolejnego cyklu
            self.layers[0][0].wages[0] = (self.layers[0][0].wages[0] +
                                          (self.n * self.layers[0][0].error * element))
            #obliczenie biasu neuronu N1 do kolejnego cyklu
            self.layers[0][0].bias += self.layers[0][0].error * self.n
            index_count += 1
        #po zadanej ilosci epok uznajemy, ze siec jest nauczona
        if (epoch == self.epoch):
            print "teaching was finished"
            teached = 1
        #zapisanie bledu i epoki (wykres blad vs epoka)
        self.err_ep.append(float(self.layers[2][0].error))
        self.epoch_n.append(float(epoch))
        epoch += 1
```

Następnie następuje test sieci, na jej wejście podano argumenty wektora uczącego, a jej wyjście porónuję się z wartością wektora uczącego.

```
def start_net(self, input_vec):
    print 'Startujemy siec'
    for x in input_vec:
        #obliczenie sumy neuronu N1
        self.layers[0][0].signal_out = x * self.layers[0][0].wages[0] + self.layers[0][0].bias
        #suma neuronu N1 podana na funkcje aktywacji (obliczenie wyjscia neuronu)
        self.layers[0][0].out_value = 1.0 / (1.0 + math.exp(-self.layers[0][0].signal_out))
        # obliczenie sumy neuronu N2
        self.layers[1][0].signal_out = (self.layers[0][0].out_value *
                                        self.layers[1][0].wages[0] + self.layers[1][0].bias)
        #suma neuronu N2 podana na funkcje aktywacji (obliczenie wyjscia neuronu)
        self.layers[1][0].out_value = 1.0 / (1.0 + math.exp(-self.layers[1][0].signal_out))
        #obliczenie sumy neuronu N3
        self.layers[1][1].signal_out = (self.layers[0][0].out_value *
                                        self.layers[1][1].wages[0] + self.layers[1][1].bias)
        #suma neuronu N3 podana na funkcje aktywacji (obliczenie wyjscia neuronu)
        self.layers[1][1].out_value = 1.0 / (1.0 + math.exp(-self.layers[1][1].signal_out))
        #obliczenie sumy neuronu N4
        self.layers[1][2].signal_out = (self.layers[0][0].out_value *
                                        self.layers[1][2].wages[0] + self.layers[1][2].bias)
        #suma neuronu N4 podana na funkcje aktywacji (obliczenie wyjscia neuronu)
        self.layers[1][2].out_value = 1.0 / (1.0 + math.exp(-self.layers[1][2].signal_out))
        #obliczenie sumy neuronu N5
        self.layers[2][0].signal_out = (self.layers[1][0].out_value * self.layers[2][0].wages[0] +
                                        self.layers[1][1].out_value * self.layers[2][0].wages[1] +
                                        self.layers[1][2].out_value * self.layers[2][0].wages[2] +
                                        self.layers[2][0].bias)
        #neuron N5 posiada liniowa funkcje aktywacji wyjscie z sumatora jest wyjsciem sieci
       #zapisanie wyjscia z sieci do wektora (potrzebne do wygenerowania wykresu)
        self.out_teached.append(self.layers[2][0].signal_out)
        self.error_teached.append(self.layers[2][0].signal_out - math.cos(x))
```

Wyniki można porównać na podstawie każdorazowo wygenerowanych wykresów.

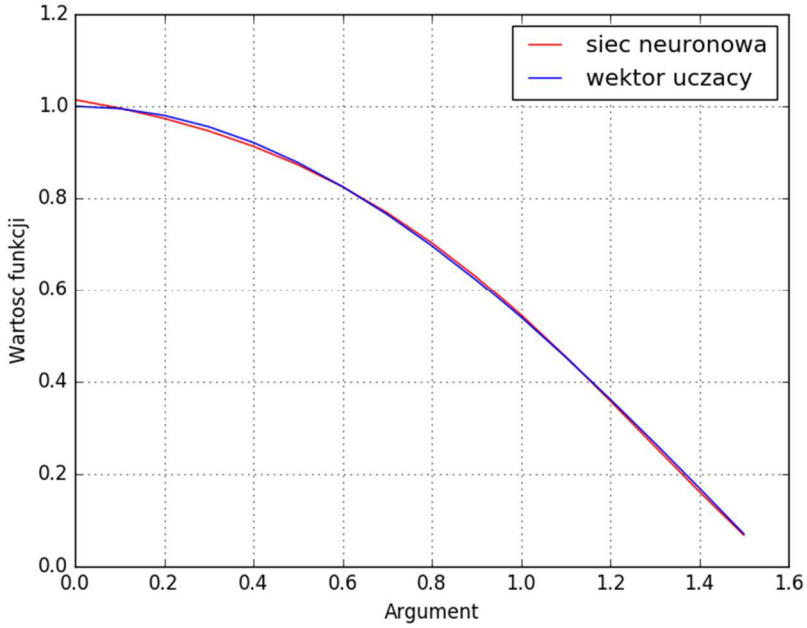

Figure 10 Wykres porównujący wektor uczący i wyjście z sieci

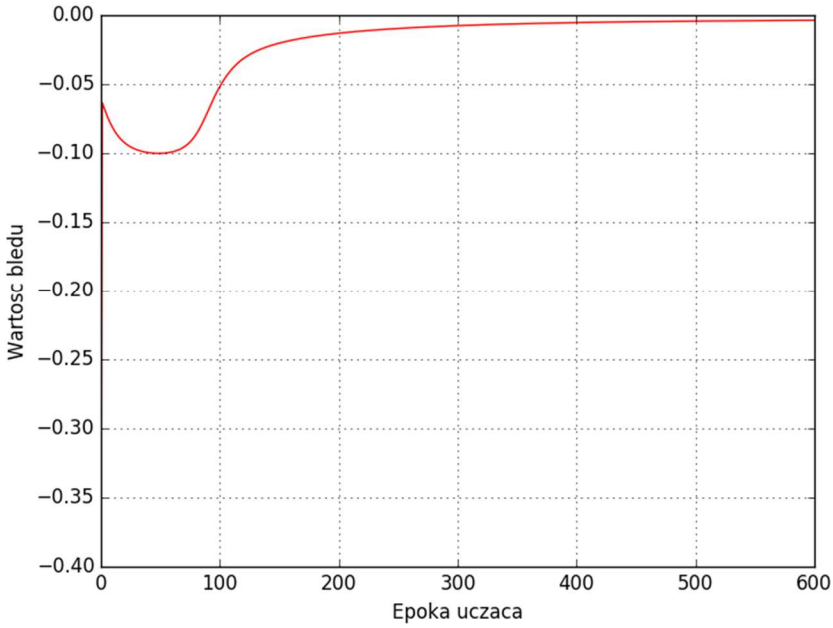

Figure 11Wykres błędu w zależności od epoki uczącej

Jak widać na powyższych wykresach, przbieg funkcji z nauczonej sieci neuronowej nie odbiega znacząco od przebiegu idealnego. Błąd stabilizuje się na wartości bliskiej zeru około 400 epoki uczącej.