POLITECHNIKA POZNAŃSKA

IMPLEMENTACJA SZTUCZNEJ SIECI NEURONOWEJ W PYTHONIE – TUTORIAL

ŁK

Spis treści

I. WSTĘP:

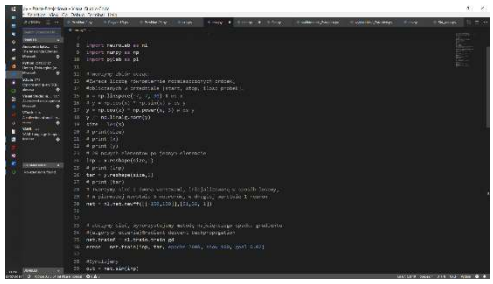
Celem pracy jest przeprowadzenie tutorialu z aproksymowania zadanych funkcji przy użyciu Sztucznej Sieci Neuronowej z zastosowaniem języka programowania Python wraz ze wszystkimi niezbędnymi bibliotekami i programami. Ponadto opracowanie zawiera w sobie podstawową wiedzę z zakresu przytoczonych sieci. Celem końcowym jest takie dobranie parametrów sieci, aby uzyskać jak najwierniejszą kopie rozważanych funkcji. Będziemy analizować funkcję typu: sinus, kosinus, wykładnicze 2 i 3 stopnia, miksy tych funkcji czy dodatkowe stałe modyfikujące ich wartości np. amplitudę.

II. ZASTOSOWANE OPROGRAMOWANIE:

1. PROGRAMY:

a) Visual Studio Code v. 1.36.0

Program typu IDE (Integrated Development Environment) ułatwiający analizę kodu, jego debugowanie, szeroką personalizację oraz możliwość zainstalowania wielu pakietów jak np. Python, C++, integracja z GitHub czy udostępniający nam pożyteczne rozszerzenia typu IntelliSense umożliwiające automatyczne uzupełnianie i podpowiadanie elementów kodu **(ilustracja 1)**.



Ilustracja 1

b) Anaconda Navigator v. 1.9.7

Jest to darmowa oraz otwarta dystrybucja narzędzi i pakietów do pracy przy pomocy Pythona w dziedzinach takich jak uczenie maszynowe czy ogólnie pojętej nauce o danych. Za pomocą tego pakietu integrujemy wszystkie niezbędne nam biblioteki z Visual Studio Code **(ilustracja 2)**.

Ilustracja 2

2. JĘZYK PROGRAMOWANIE PYTHON:

Jest otwarto źródłowym, wieloparadygmatowym, uniwersalnym językiem programowania. Umożliwia programowanie obiektowe, funkcyjne i proceduralne. Można w nim pisać skrypty jak i aplikacje. Charakteryzuje się czytelności kodu oraz

szeroka gama dodatkowych bibliotek. Jest to język uważany z dość łatwy w nauce programowania.[1]

3. BIBLIOTEKI:

a) NeuroLab:

W bibliotece tej znajdziemy podstawowe algorytmy związane z Sieciami Neuronowymi (SN). Biblioteka bazuje na pakiecie Numpy i jest podobna w zastosowaniu do Neural Network Toolbox z płatnego Matlaba[2]. Posiada szczegółową dokumentację modułów wchodzących w jej skład oraz kilka przykładowych programów na stronie internetowej. [3]

b) Numpy:

Wykorzystywany powszechnie pakiet fundamentalnych modułów w obliczeniach naukowych rożnego typu, wraz z matplotlib stanowią poważna alternatywę dla pakietu Matlab. Ułatwią działania na tablicach i macierzach wielowymiarowych oraz zawiera wbudowana kolekcje wysokopoziomowych funkcji matematycznych. Więcej można się dowiedzieć na oficjalnej stronie pakietu.[4]

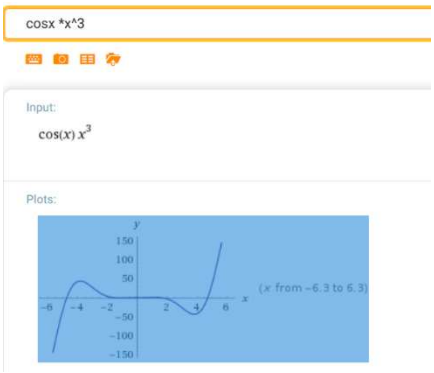
c) Matplotlib:

Biblioteka ta umożliwia przedstawianie danych, na których pracujemy w formie graficznych wykresów rożnego typu. Zapewnia ponadto szeroka możliwość personalizacji wyświetlania danych.[5]

III. TEORIA O SSN:

1. TEORIA MATEMATYKA:

Aby sprawdzić jaki jest kształt oraz zakresy osi funkcji, które nasza siec będzie miała za zadanie aproksymować warto wykorzystać bezpłatną stronę internetową **WolframAlpha**[6]. Dzięki uzyskanym w ten sposób informacją będziemy mogli efektywnie dobrać zbiór uczący na bazie którego nasza siec będzie się kształtować. Przykładowy zrzut ekranu widoczny jest na **Ilustracja 3**.



Ilustracja 3

[1] „Python. Leksykon kieszonkowy. Książka. Mark Lutz. Księgarnia informatyczna Helion.pl", 7, udostępniono 12 lipiec 2019, https://helion.pl/ksiazki/python-leksykon-kieszonkowy-mark-lutz,pythlk.htm#format/d.

[2] „Deep Learning Toolbox", udostępniono 13 lipiec 2019, https://uk.mathworks.com/products/deep-learning.html.

[3] „Welcome to NeuroLab's documentation! — NeuroLab 0.3.5 documentation", udostępniono 7 lipiec 2019, https://pythonhosted.org/neurolab/index.html#.

[4] „NumPy — NumPy", udostępniono 7 lipiec 2019, https://www.numpy.org/.

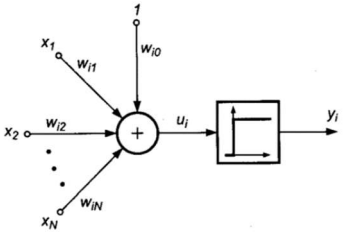
[5] „Matplotlib: Python plotting — Matplotlib 3.1.1 documentation", udostępniono 12 lipiec 2019, https://matplotlib.org/.

[6] „Wolfram|Alpha: Making the World's Knowledge Computable", udostępniono 12 lipiec 2019, www.wolframalpha.com/.

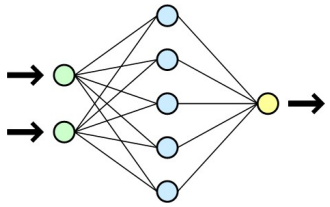
2. TEORIA SSN:

a) Teoria:

Na podstawie rzeczywistych neuronów stworzono modele matematyczne w rożnym stopniu wykorzystujące ich cechy. Jednym z nich jest model **McCullocha-Pittsa** charakteryzujacy się tym, że posiada wiele wejsc i tylko jedno wyjscie. Każdemu z wejsc przyporzadkowana jest waga wejsciowa w_{ij} bedaca pewna liczba rzeczywista. Wartosc dodatnia oznacza synapse pobudzajacą ujemna hamujaca. Funkcja aktywacji u_i w modelu jest funkcja skokowa **Ilustracja 4**. Uzyskany perceptron to podstawowy element sieci neuronowej którą będziemy wykorzystywać **Ilustracja 6**.[7]



Ilustracja 6



Ilustracja 5

$$f(u) = \begin{cases} 1, & u > 0 \\ 0, & u \leqslant 0 \end{cases}$$

Ilustracja 4

b) Algorytm uczenia:

Uczenie perceptronu polega na znalezieniu wartości jego wag. Dla rozważanych przykładów został zastosowany algorytm wstecznej propagacji błędu. Działa on na zasadzie obliczenia odpowiedzi sieci na zadany wzorzec, następnie oblicza dla ostatniej warstwy neuronów gradient funkcji błędu. Kolejnym krokiem jest modyfikacja wagi neuronów a błąd jest propagowany do warstwy wcześniejszej sieci, gdzie proces się powtarza w oparciu o wartość gradientu z warstwy wyższej. Algorytm postępuje tak aż do warstwy wejściowej. Procedura zostaje powtórzona aż sygnały wyjściowe sieci będą dostatecznie bliskie oczekiwanym.[8] Uproszczony schemat jednokierunkowej sieci neuronowej o dwóch wejściach, jednym wyjściu i 5 neuronach ukrytych przedstawia **ilustracja 6**[9]. Sieć jest uczona w trybie z nauczycielem.

---

[7] *SIECI NEURONOWE DO PRZETWARZANIA INFORMACJI wyd.3*, 12–17, udostępniono 12 lipiec 2019, http://ekonomiczna24.osdw.pl/ksiazka/STANISLAW-OSOWSKI/SIECI-NEURONOWE-DO-PRZETWARZANIA-INFORMACJI-wyd-3,ekonoJB3YG9J3?ms=-1.

[8] *SIECI NEURONOWE DO PRZETWARZANIA INFORMACJI wyd.3*, 40–62.

[9] „Sieć neuronowa”, w *Wikipedia, wolna encyklopedia*, 6 czerwiec 2019, https://pl.wikipedia.org/w/index.php?title=Sie%C4%87_neuronowa&oldid=56831017.

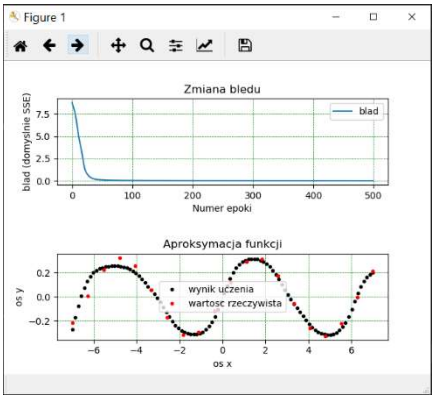
IV. OPIS OPRACOWANEGO KODU:

1. FUNKCJA SIN

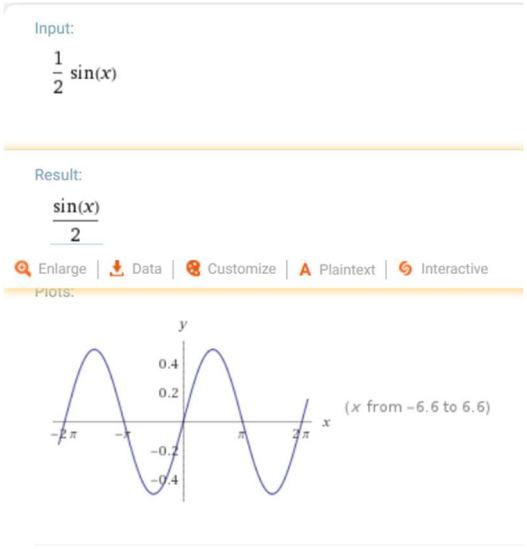
```
1.  import neurolab as nl
2.  import numpy as np
3.  import pylab as pl

4.  #Tworzymy zbiór ucząc:
5.  #Zwraca liczbę równomiernie rozmieszczonych próbek,
6.  #obliczanych w przedziale [start, stop, ilosc probek].
7.  x = np.linspace(-7, 7, 20) # os x
8.  y = np.sin(x) * 0.5 # os y
9.  y /= np.linalg.norm(y)
10. size = len(x)
11. # 20 nowych elementow po jednym elemencie
12. inp = x.reshape(size,1)
13. print (inp)
14. tar = y.reshape(size,1)
15. print (tar)
16. # Tworzymy sieć z dwoma warstwami, inicjalizowaną w #sposób
    losowy, pierwszej warstwie 5 neuronów, w drugiej 1
17. net = nl.net.newff([[-7, 7]],[7, 1])
18. # #Uczymy sieć, wykorzystujemy metodę największego spadku
    gradientu
19. #(algorytm uczenia)Gradient descent backpropogation
20. net.trainf = nl.train.train_gd
21. error = net.train(inp, tar, epochs=500, show=100, goal=0.01)
22. #Symulujemy
23. out = net.sim(inp)
24. fig = pl.figure()
25. fig.subplots_adjust(hspace=0.8, wspace=0.4)
26. #wykres z iloscia epok i bledem uzyskanym
27. pl.subplot(211)
28. pl.plot(error)
29. pl.grid(color='g', linestyle='--', linewidth=0.5)
30. pl.xlabel('Numer epoki')
31. pl.ylabel('blad (domyslnie SSE)')
32. pl.legend(['blad'])
33. pl.title('Zmiana bledu')
34. # #Tworzymy wykres z wynikami
35. x2 = np.linspace(-7.0,7.0,100)
36. y2 = net.sim(x2.reshape(x2.size,1)).reshape(x2.size)
37. y3 = out.reshape(size)
38. #wykres z funkcja wzorcowa i wyuczona
39. pl.subplot(212)
40. pl.plot(x2, y2, '.',color='black')
41. pl.title('Aproksymacja funkcji')
42. pl.grid(color='g', linestyle='--', linewidth=0.5)
43. pl.xlabel('os x')
44. pl.ylabel('os y')
45. pl.plot(x , y, '.',color='red')
46. pl.legend(['wynik uczenia','wartosc rzeczywista'])
47. pl.show()
```

Ilustracja 7

W liniach 1-3 importujemy wymagane biblioteki z których będziemy korzystać w naszym programie. W linii 7 tworzymy zbiór punktów na osi x w przedziale od -7 do 7 w ilości 20. Linia 8 tworzy przypisanie algorytmu generującego funkcje $\mathbf{y}=\frac{1}{2}\sin(x)$ do zmiennej *y*. Linia 9 tworzymy przestrzeń unormowana (norma - uogólnienie długości) wektora w przestrzeni liniowej. Linia 10 dla zmiennej *size* przypisujemy wartość naturalną określająca liczbę elementów wektora *x*. W 12 i 14 dokonujemy zmiany kształtu tablic *x* oraz *y* na dwadzieścia wierzy i jedna kolumnę. Punkt 17 ma za zadanie stworzyć przy pomocy biblioteki Neurolab wielowarstwowy perceptron o jednym wejściu oraz wyjściu i 7 neuronach warstwy ukrytej. Lista *[-7,7]* zawiera zakres wartości wejściowych. W 20 linii wybieramy algorytm uczenia (algorytm wstecznej propagacji gradientu), natomiast w 21 przypisujemy zbiór uczący, czyli wartość wejściowa (inp) oraz docelowy cel aproksymacji sieci. Wybieramy tu również ilość epok, czyli powtórzeń procesu (epochos) oraz maksymalny błąd jaki siec może osiągnąć. Kolejno 23 linia rozpoczyna proces symulacji i przypisuje go do zmiennej out. W 24-25 linii rysujemy płaszczyznę pod wykorzystywane wykresy oraz określamy pomiędzy nimi odległości. Od 27 do 33 kolejno dodajemy wykres, który jest częścią wcześniejszej płaszczyzny i zajmuje 1 miejsce w przestrzeni dwóch wierszy i jednej kolumny. Rysujemy linie błędu, wybieramy kolor, styl oraz szerokość linii, oznaczamy osie i dodajemy tytuł wykresu. Od 35 – 37 linii dzielimy os x w zakresie [-7,7] na sto części, następnie zmieniamy kształt wektora będącego zbiorem elementów po przecinku na wektor będący zbiorem tych elementów podzielonych każdy z osobna na oddzielny wektor. Z tak uzyskanych danych wracamy do formy pierwotnej. Od 39 do 47 podobnie jak od 27 do 33 rysujemy drugi wykres przedstawiający korelacje pomiędzy funkcja nauczona a ucząca.

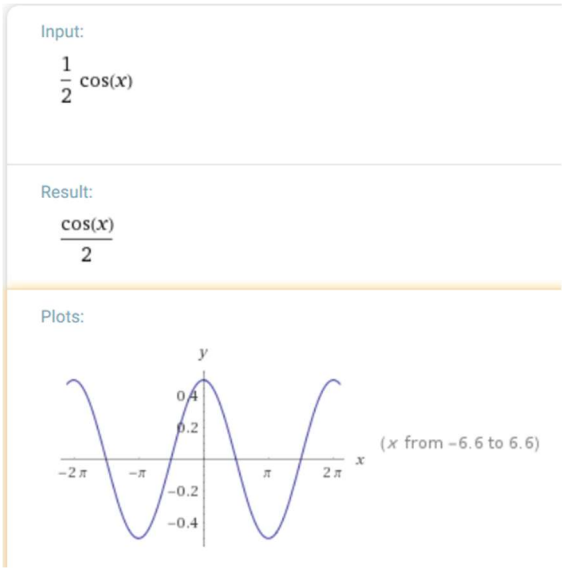


Ilustracja 8

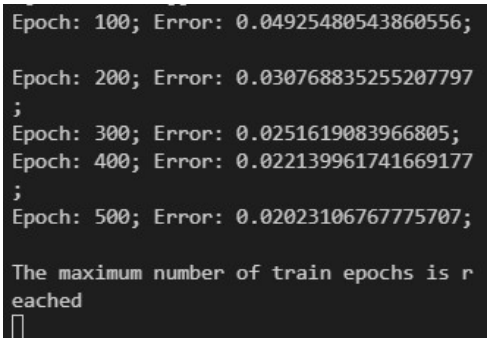


Ilustracja 9

Ilustracja 8 przedstawia wynik programu z **ilustracji 7**. Widzimy na pierwszym wykesie, że po około 30 epokach wartosc błędu zadanego ustalila się. Na dolnym wykresie przedstawiony efekt uczenia w posaci czarnych punktow dobrze odwzorowuje funkcje rzeczywistą wyznaczoną przez punkty czerwone. **Ilustracja 9** obrazuje wygenerowaną przez stronę **WolframAlpha** funkcje pogladową, natomiast **ilustracja 10** to wycinek z konsoli programu **Visual Studio Code** ukazujacy spadek błędu aproksymacji funkcji względem zadenego zbioru uczącego.



Ilustracja 10



Ilustracja 11

2. FUNKCJA COS:

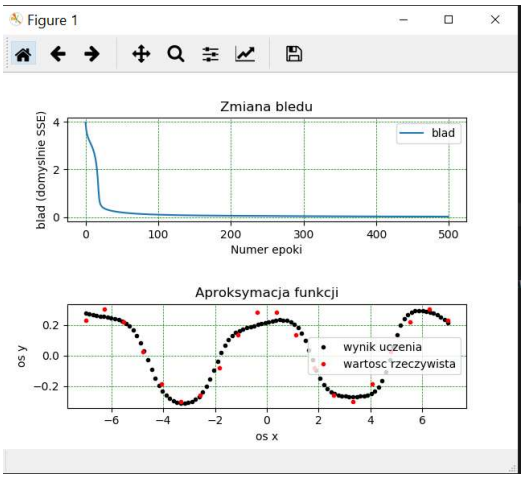
Kod programu **(ilustracja 12)** do aproksymacji funkcji kosinus jest analogiczny z przykładem z **ilustracji 7**. Jedyną zmianą jest zastosowanie analogicznej do analizowanego przykładu funkcji $y=\frac{1}{2}\cos(x)$ w linii 8. Jak można było się spodziewać przy tak nieznacznej modyfikacji programu czas osiągniecia zadanego błędu pomiędzy wynikiem uczenia a wartością rzeczywistą jest analogiczny do aproksymacji funkcji sinus i wynosi około 30 epok **ilustracja 12 oraz 13. Ilustracja 10** to funkcja docelowa.

```
1.  import neurolab as nl
2.  import numpy as np
3.  import pylab as pl

4.  #Tworzymy zbiór ucząc:
5.  #Zwraca liczbę równomiernie rozmieszczonych
    próbek,
6.  #obliczanych w przedziale [start, stop, ilosc
    probek].
7.  x = np.linspace(-7, 7, 20) # os x
8.  y = np.cos(x) * 0.5
9.  y /= np.linalg.norm(y)
```

Ilustracja 14

```
Epoch: 100; Error: 0.09890351768334227;

Epoch: 200; Error: 0.05411531968135481;

Epoch: 300; Error: 0.0361865413365856;
Epoch: 400; Error: 0.023405232678432544
;
Epoch: 500; Error: 0.0181498059450648;
The maximum number of train epochs is r
eached
```

Ilustracja 13



Ilustracja 12

3. FUNKCJA WYKŁADNICZA 2 STOPNIA:

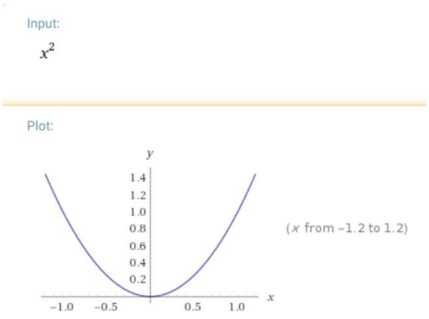
W tym przykładzie zajmiemy się funkcja wykładnicza typu $y=x^2$. Analizowany fragment kodu z **ilustracji 15** tak jak do tej pory zawiera tylko linie różniące się w stosunku do kodu z **ilustracji 7**. Bazując na **ilustracji 17** zmodyfikowaliśmy kolejno zakres danych wejściowych w linii kodu nr. 1, algorytm wyliczania funkcji wzorcowej w linii 2, ilość neuronów warstwy ukrytej na 10, zwiększyliśmy liczbę epok uczenia do 1000 oraz dopasowaliśmy pod nowy wykres zakres punktów na osi x. Jak wynika z **Ilustracji 15** pomimo zwiększenia ilości epok program i tak zakończy swoje działanie wraz z osiągnieciem zadanej różnicy dokładności aproksymacji. **Ilustracja 18** pokazuje dobry efekt odwzorowania funkcji.

```
1.  x = np.linspace(-1.6, 1.6, 30) # os x
2.  y = np.power(x, 2)  # os y
3.  y /= np.linalg.norm(y)

4.  net = nl.net.newff([[-1.6, 1.6]],[10, 1])

5.  error = net.train(inp, tar, epochs=1000,
    show=100, goal=0.01)

6.  x2 = np.linspace(-1.5,1.5,30)
```

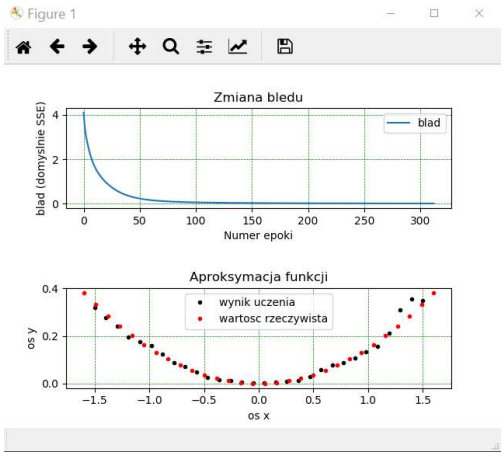
Ilustracja 16

```
Epoch: 100; Error: 0.06179460314563187;

Epoch: 200; Error: 0.019407687256402323
;
Epoch: 300; Error: 0.010616946204884278
;
The goal of learning is reached
```

Ilustracja 15



Ilustracja 17



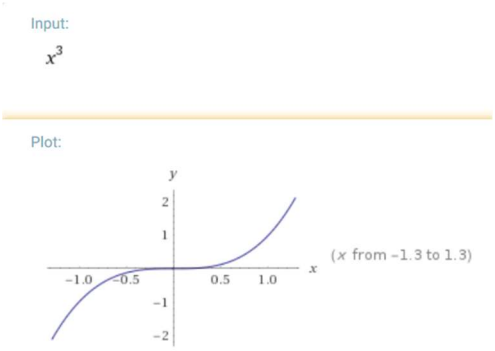
Ilustracja 18

4. FUNKCJA WYKŁADNICZA 3 STOPNIA:

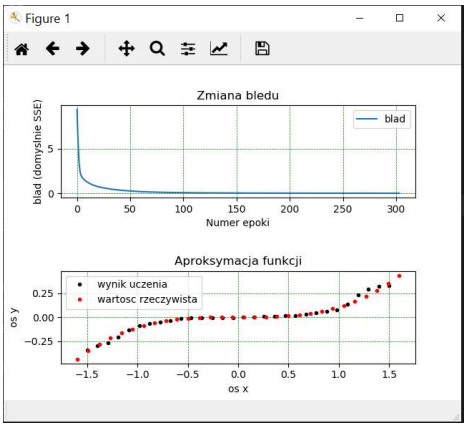
Program funkcji (**Ilustracja 19)** $y = x^3$ został nieznacznie zmodyfikowany względem funkcji $y = x^2$ na bazie danych z **Ilustracji 21**. Zmieniono rodzaj algorytmu funkcji, zmniejszono ilość neuronów warstwy ukrytej oraz epok uczenia co dało przy mniejszym nakładzie obliczeniowym efekt równie dobry jak w przykładzie poprzednim co widać na **ilustracji 20 i 22**.

```
1.  y = np.power(x, 3)  # os y
2.  net = nl.net.newff([[-1.6, 1.6]],[8, 1])
3.  error = net.train(inp, tar, epochs=500, show=100, goal=0.01)
```

Ilustracja 19



Ilustracja 20



Ilustracja 21

```
Epoch: 100; Error: 0.07438891291047789;

Epoch: 200; Error: 0.021469349059033343
;
Epoch: 300; Error: 0.010249391257079313
;
The goal of learning is reached
```

Ilustracja 22

5. MIX FUNKCJI:

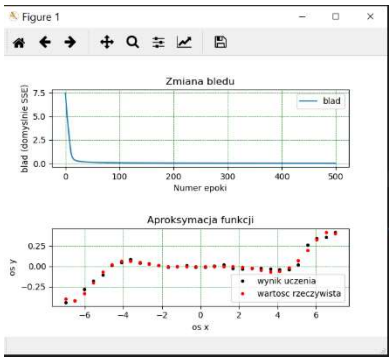
```
1.  y = np.cos(x) * np.power(x, 3) # os y
2.  net = nl.net.newff([[-7,7]],[13,7,1])
3.  x2 = np.linspace(-7.0,7.0,30)
```

Ilustracja 24

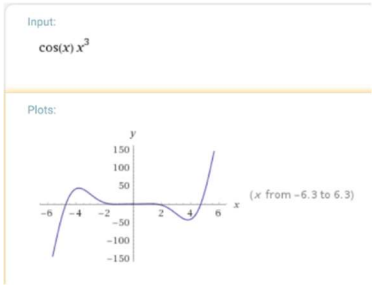
Ostatnim przykładem jest złożenie kilku funkcji w jedna poprzez operator mnożenia, co ma sprawdzić, jak zachowa się program w przypadku bardziej skomplikowanych funkcji. Analizowana funkcja ma postać $y = \cos x \times x^3$ i jest zobrazowana na **ilustracji 25**. Aby uzyskać optymalny efekt uczenia dodaliśmy dodatkową warstwę do naszej sieci neuronowej. Zabieg ten umożliwił nam znaczna optymalizacje całego procesu i uzyskanie nieporównywalnie lepszych wyników uczenia co widzimy na **Ilustracji 24 oraz 26**.

```
Epoch: 100; Error: 0.06208553460287004;
Epoch: 200; Error: 0.03038973927285247 6;
Epoch: 300; Error: 0.02039155127638618;
Epoch: 400; Error: 0.014310325394476941 ;
Epoch: 500; Error: 0.01068082040196094;
```

Ilustracja 23



Ilustracja 26



Ilustracja 25

V. PODSUMOWANIE:

Dzięki analizie powyższych przypadków przy pomocy języka programowania Python, środowiska Conda wraz ze wszystkimi bibliotekami udało nam się wykazać, że posługiwanie się prostymi sieciami neuronowymi i wykorzystywanie ich do aproksymacji funkcji matematycznych nie jest bardzo zaawansowana umiejętnością. Nauczyliśmy się odpowiednio dobierać dane do zbioru uczącego co jest wraz z określeniem ilości warstw oraz neuronów posiadanych przez nie kluczowym elementem uzyskania dobrych wyników wyjściowych. Liczbę neuronów warstwy ukrytej można wstępnie oszacować dodając do siebie ilość neuronów warstwy wyjściowej i przedającej ukryta oraz dzieląc uzyskany wynik przez 2. Ponadto nauczyliśmy się wykorzystywać darmowe środowisko pracy, które nierzadko daję większe możliwości niż komercyjne odpowiedniki.

VI. BIBLIOGRAFIA:

„Deep Learning Toolbox”. Udostępniono 13 lipiec 2019. https://uk.mathworks.com/products/deep-learning.html.

„Matplotlib: Python plotting — Matplotlib 3.1.1 documentation”. Udostępniono 12 lipiec 2019. https://matplotlib.org/.

„NumPy — NumPy”. Udostępniono 7 lipiec 2019. https://www.numpy.org/.

„Python. Leksykon kieszonkowy. Książka. Mark Lutz. Księgarnia informatyczna Helion.pl”. Udostępniono 12 lipiec 2019. https://helion.pl/ksiazki/python-leksykon-kieszonkowy-mark-lutz,pythlk.htm#format/d.

SIECI NEURONOWE DO PRZETWARZANIA INFORMACJI wyd.3. Udostępniono 12 lipiec 2019. http://ekonomiczna24.osdw.pl/ksiazka/STANISLAW-OSOWSKI/SIECI-NEURONOWE-DO-PRZETWARZANIA-INFORMACJI-wyd-3,ekonoJB3YG9J3?ms=-1.

„Sieć neuronowa”. W *Wikipedia, wolna encyklopedia*, 6 czerwiec 2019. https://pl.wikipedia.org/w/index.php?title=Sie%C4%87_neuronowa&oldid=56831017.

„Welcome to NeuroLab’s documentation! — NeuroLab 0.3.5 documentation”. Udostępniono 7 lipiec 2019. https://pythonhosted.org/neurolab/index.html#.

„Wolfram|Alpha: Making the World’s Knowledge Computable”. Udostępniono 12 lipiec 2019. www.wolframalpha.com/.