Politechnika Łódzka Instytut Informatyki Stosowanej

Podstawy Sztucznej Inteligencji Laboratorium

Ćwiczenie 4.

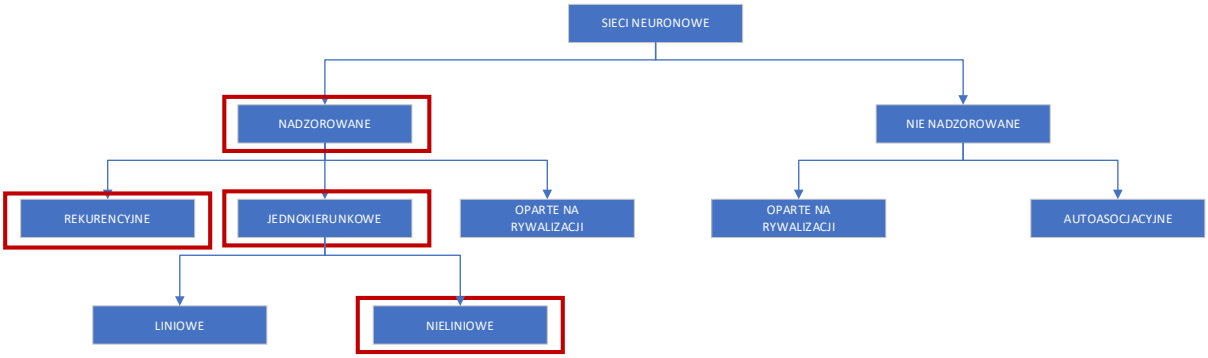
Sztuczne Sieci Neuronowe typu „Feed-Forward” jako uniwersalne aproksymatory

Opracował:

Dr inż. Piotr Urbanek

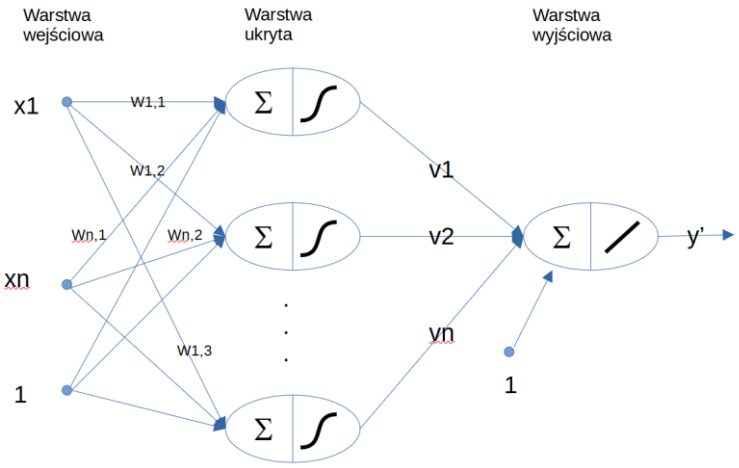
Wstęp.

Współcześnie Sztuczne Sieci Neuronowe (SSN) dzielą się sieci płytkie (shall NN) oraz głęboki (Deep Neural Network). Tematem tego ćwiczenia jest badanie właściwości niektórych rodzajów płytkich sieci neuronowych. Głębokie sieci neuronowe stanowią rozwinięcie teorii klasycznych (płytkich sieci) i będą tematem dyskusji na przedmiotach Sieci neuronowe 1 i 2 na drugim stopniu studiów (magisterskich).
Klasyfikacja płytkich sieci neuronowych może być przedstawiana w postaci diagramu:
Na rysunku 1 przedstawiono systematykę rozwoju płytkich sieci neuronowych. W ramkach ujęto sieci, które będą przedmiotem badań w obecnym ćwiczeniu.



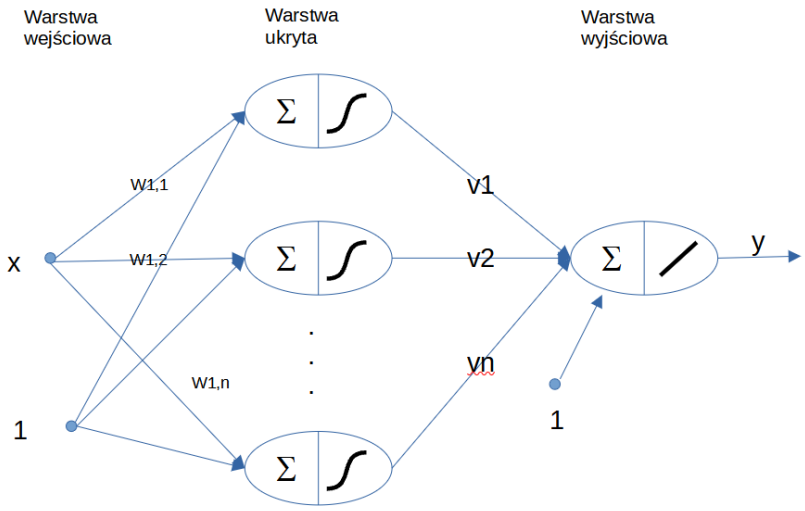
Rys. 1. Klasyfikacja płytkich SSN

Nadzorowana, jednokierunkowa sieć nieliniowa jako uniwersalny aproksymator.



$$h(x) = \sum_{i=1}^{m} g_i(x) + v_0$$

Uczenie takiej sieci odbywa się przez podanie na jej wejście wartości charakteryzujących badany przebieg. Do takiego zadania konstruujemy sieć składającą się 1 wejścia w warstwie wejściowej, n – neuronów zawierających sigmoidalne funkcje aktywacji w warstwie ukrytej oraz jednego neuronu o liniowej funkcji aktywacji w warstwie wyjściowej. W ten sposób będziemy w stanie odwzorować zależności funkcji jednej zmiennej y=f(x).
Zatem sieć może wyglądać tak, ja na rysunku 2.



Rys. 2. Sieć neuronowa rozwiązująca zagadnienie dopasowania (uniwersalnego aproksymatora).

W celu utworzenia sieci można skorzystać z różnych bibliotek rozwiązujących zagadnienia grafów. Na początek można wykorzystać bibliotekę o nazwie „neurolab”. W tym celu należy ją doinstalować do używanego środowiska.
Biblioteka ta zawiera możliwość tworzenia najważniejszych rodzajów sieci neuronowych w tym oczywiście sieci jednokierunkowe (feed-forward) uczone metodą wstecznej propagacji błędów z kilkoma rodzajami metod optymalizujących dobór wag sieci.

Zadanie.

1. Wygenerować funkcję jednej zmiennej zapewniającej przebieg cykliczny o mało zmiennej amplitudzie i okresie, np. $f(x) = 2xsin(x)$.
2. Wykorzystując funkcję neurolab.net.newff napisać aplikację symulującą uczenie sieci typu feed-forward.
3. Dla różnych metod uczenia oraz liczby neuronów w warstwie ukrytej wyznaczyć sumaryczny błąd uczenia sieci oraz pokazać na wykresie jego dopasowania do przebiegu rzeczywistego.
4. Wygenerować funkcję jednej zmiennej zapewniającej dostatecznie nieregularny przebieg, np. $f(x) = 2\sqrt[3]{x}\, sin\left(\frac{x}{10}\right) cos(3x)$ dla $x \in \langle 0; 9 \rangle$.
5. Sprawdzić możliwość dobrania najlepszej metody uczenia oraz optymalnej liczby neuronów w warstwie ukrytej dla nowego przebiegu.
6. Przedstawić wnioski z uczenia sieci przebiegów szybkozmiennych.
7. Wykorzystując toolbox nnstart w środowisku Matlab sprawdzić proces uczenia z wykorzystaniem takich metod uczenia sieci, jak metoda Levenberga-Marquadta oraz Regularyzacji Bayesowej.
8. Opisać wnioski z nauki.

Uwagi!

1. Dokumentacja wraz z przykładami biblioteki neurolab w środowisku Python znajduje się na stronie https://pythonhosted.org/neurolab/
2. Środowisko Matlab można użyć w trybie on-line wykorzystując konto na stronie www.mathworks.com
3. Wyniki badań można zaprezentować w przykładowych tabelach:

Rodzaj funkcji
Wartość błędu uczenia SSN metodą MSE (Mean Square Method).

Algorytm uczący →	**train_gd()**	**train_gdm()**	**train_gda()**	**train_gdx()**	**train_rprop()**
Liczba neuronów w warstwie ukrytej ↓					
3					
5					
10					
15					
30					
50					

Wyjaśnić krótko oznaczenia parametrów funkcji. Przykładowo: „train_gd” – uczenie metodą największego spadku (gradient descent).

```
# -*- coding: utf-8 -*-

# Ważne - do ćwiczenia!

import neurolab as nl
import numpy as np
import pylab as pl

#Tworzymy zbiór ucząc
x = np.linspace(-7, 7, 30)
# y = np.sin(x) * np.cos(x)
y1 = 2*x* np.cos(x)
wsp=np.abs(max(y1)-min(y1))
y=y1/wsp
# y=np.cos(x)

size = len(x)
print (x)
inp = x.reshape(size,1)
print (inp)
tar = y.reshape(size,1)

#Tworzymy sieć z dwoma warstwami, inicjalizowaną w sposób losowy,
#w pierwszej warstwie x neuronów, w drugiej warstwie 1 neuron
net = nl.net.newff([[-7, 7]],[10, 1])

#Uczymy sieć, wykorzystujemy metodę największego spadku gradientu
# net.trainf = nl.train.train_gd

net.trainf = nl.train.train_gdx
error = net.train(inp, tar, epochs=1000, show=100, goal=0.05)

# error = net.train(x, y, epochs=1000, show=100, goal=0.01)

#Symulujemy
out = net.sim(inp)

#Tworzymy wykres z wynikami
x2 = np.linspace(-6.0,6.0,150)
y2 = net.sim(x2.reshape(x2.size,1)).reshape(x2.size)
y3 = out.reshape(size)

pl.plot(x2, y2, '-',x , y, '.', x, y3, 'p')
pl.legend(['wynik uczenia','wartosc rzeczywista'])
pl.show()
```