Politechnika Łódzka **Instytut Informatyki Stosowanej**

Podstawy Sztucznej Inteligencji

Laboratorium

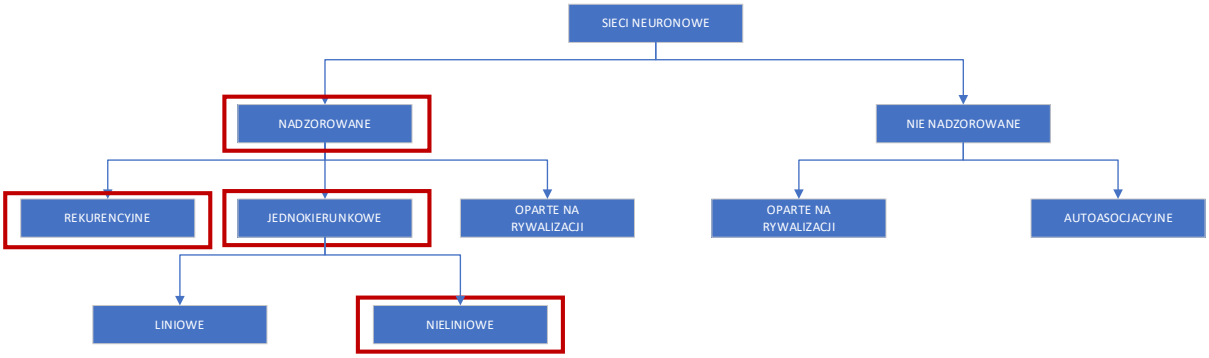
Ćwiczenie 4

Płytkie sieci neuronowe.

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:



Rys. 1. Klasyfikacja płytkich SSN

Zdolności generalizacyjne sieci neuronowych

W środowisku Python istnieje wiele bibliotek zawierających funkcje do modelowania zarówno płytkich, jak i głębokich sztucznych sieci neuronowych. Do najpopularniejszych z nich należą biblioteki Keras i Tensorflow. Są one używane głownie do modelowania głębokich sieci neuronowych. Płytkie sieci mogące rozwiązywać zagadnienia przedstawione w poprzednich punktach są zamodelowane w bibliotece Neurolab. Wszelkie informacje o tej bibliotece można znaleźć na stronie: https://pythonhosted.org/neurolab/index.html
Jako przykład wykorzystania biblioteki neurolab podano dwa programy: pierwszy do rozwiązania zagadnienia klasyfikacji (pojedynczy perceptron), drugi do rozwiązania zagadnienia dopasowania wyjścia sieci neuronowej do przebiegu funkcji liniowej.

```
# -*- coding: utf-8 -*-
"""
import neurolab as nl

# Program 1 - modelowanie działania perceptronu
#Zbiór uczący
train = [[0, 0], [0, 1], [1, 0], [1, 1]]
train_target = [[0], [0], [0], [1]]

#Zbiór testowy
test = [[2, 1], [2,2], [-1,0], [-1,1]]
test_targets=[[1], [1], [0], [0]]

#Tworzymy sieć z 2 wejściami i 1 neuronem
net = nl.net.newp([[-1, 2],[0, 2]], 1)
```

```
#Uczymy sieć
error = net.train(train, train_target, epochs=100, show=10, lr=0.1)

#Symulujemy
out = net.sim(test)

#Obliczamy błąd
f = nl.error.MSE()
test_error = f(test_targets, out)

print ("Błąd klasyfikacji: %f" %test_error)

print("Wzorzec",test_targets)
print("Wynik zwrócony przez SSN",out)
```

```
# Program 2.
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_ncg
error = net.train(inp, tar, epochs=1000, show=100, goal=0.1)
# 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()
```

Zadania do wykonania:

1. Uruchom w środowisku Python "Program 1 – modelowanie działania perceptronu". Sprawdź, jak realizowane jest zagadnienie klasyfikacji na zbiorach uczącym i testowym. Wprowadź nowe punkty uczące niespełniające kryterium liniowej separowalności. Jak przebiegł proces uczenia i testowania perceptronu?
2. Uruchom w środowisku Python „Program 2. Modelowanie zagadnienia dopasowania za pomocą sieci typu feed-forward". Zamodeluj uczenie sieci dwóch funkcji zaimplementowanych w programie. Oblicz błąd średniokwadratowy dla następujących parametrów sieci:

Algorytm metody największego spadku (train_gd)

Liczba neuronów w warstwie ukrytej	3	5	10	20	30	50
Średniokwadratowy błąd odwzorowania						

Algorytm metody train_gdm

Liczba neuronów w warstwie ukrytej	3	5	10	20	30	50
Średniokwadratowy błąd odwzorowania						

Algorytm metody train_gda

Liczba neuronów w warstwie ukrytej	3	5	10	20	30	50
Średniokwadratowy błąd odwzorowania						

Wyjaśnić na podstawie strony www producenta biblioteki neurolab, co oznaczają nazwy netod uczenia „train_gd", „train_gdm" i „train_gda".

Takie samo ćwiczenie przeprowadzić na większej liczbie próbek, np. 130 zamiast 30
Odpowiedzieć na pytania:

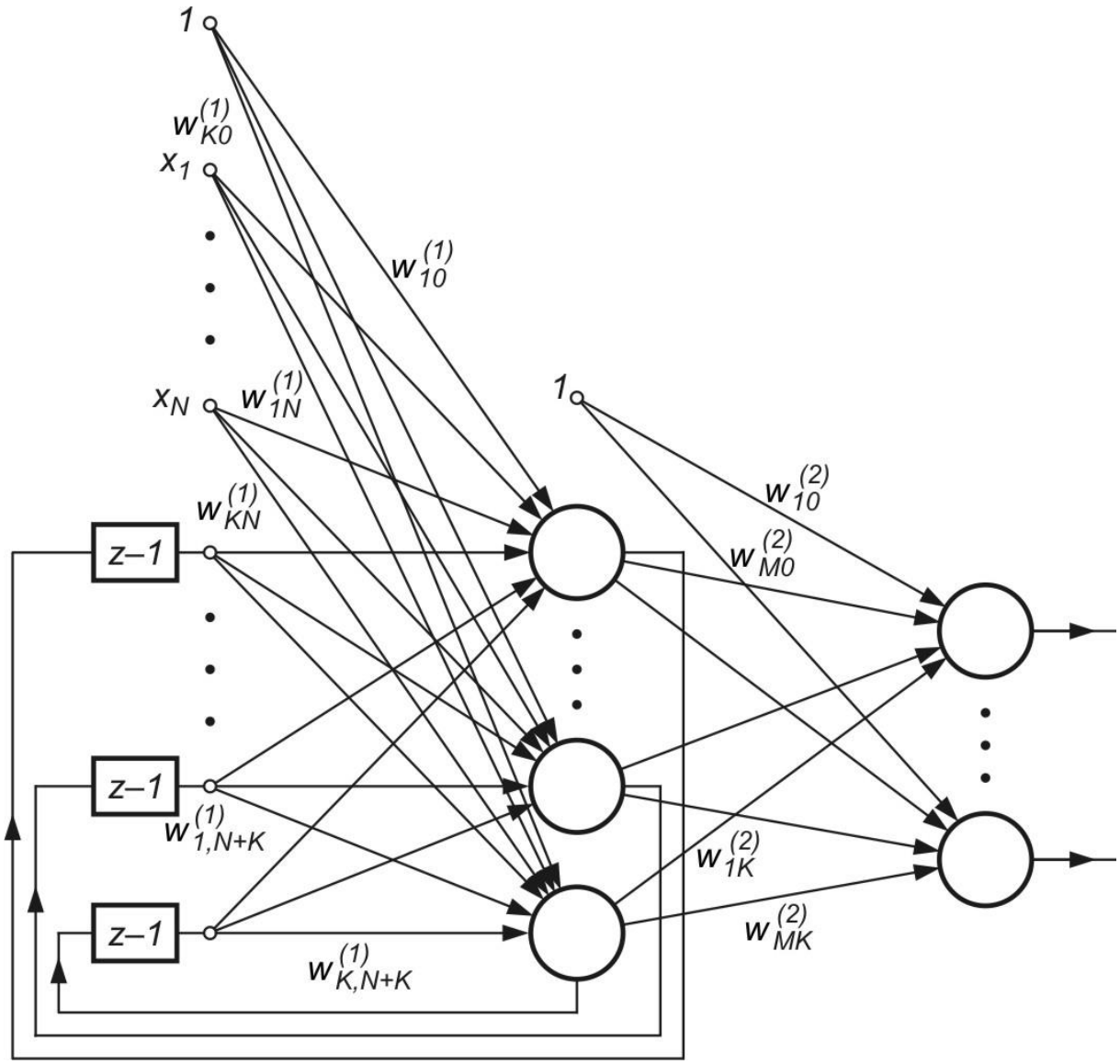
1. Co dla uczenia SSN jest bardziej korzystne: duża czy mała liczba próbek uczących?
2. Czy istnieje optymalna liczba neuronów w warstwie ukrytej dla dużej i małej liczby próbek?
3. Jak metody przetestowane metody doboru wag wpływają na proces uczenia?

Sieci rekurencyjne na przykładzie sieci Elmana

Dotychczas rozważane nieliniowe sieci należały do sieci jednokierunkowych, w których przepływ sygnału odbywał się od wejścia do wyjścia. Ustalanie się wartości sygnałów w takich sieciach odbywało się w tej samej chwili. Odmienny typ stanowią **sieci rekurencyjne**,

w których istnieje **sprzężenie zwrotne** od wyjścia sieci w kierunku wejścia. Zasadniczą cechą sieci rekurencyjnych jest istnienie sprzężenia zwrotnego między neuronami. Sprzężenie to może wychodzić od neuronów wyjściowych lub neuronów warstwy ukrytej i być przeniesione do wejścia sieci lub do warstwy uprzedniej. Sprzężenie zwrotne powoduje powstanie pewnego stanu nieustalonego w procesie przesyłania sygnałów, stanowiąc istotną różnicę w stosunku do sieci rozważanych dotychczas.
Sprzężenie zwrotne może być wyprowadzone bądź z warstwy wyjściowej neuronów, bądź z warstwy ukrytej. W każdym torze takiego sprzężenia umieszcza się blok opóźnienia jednostkowego, pozwalający rozpatrywać przepływ sygnałów jako jednokierunkowy (sygnał wyjściowy z poprzedniego cyklu zegarowego stanowi sygnał znany, powiększający jedynie rozmiar wektora wejściowego **x** sieci). Tak rozumiana sieć rekurencyjna ze względu na sposób tworzenia sygnału wyjściowego działa podobnie jak sieć jednokierunkowa (perceptronowa). Tym niemniej algorytm uczący takiej sieci, adaptujący wartości wag synaptycznych, jest bardziej złożony, Ze względu na zależność wartości sygnałów w chwili *t* od ich wartości w chwilach poprzedzających i wynikającą z tego bardziej rozbudowaną formułę wektora gradientu. Spośród wielu sieci rekurencyjnych tutaj zostaną przedstawione dwie: **sieć Elmana**, stanowiąca rozwiązanie płytkie oraz **sieć LSTM** stanowiąca strukturę głęboką.
Zasadniczą zaletą sieci rekurencyjnych jest uwzględnienie związków między aktualnymi stanami neuronów a ich stanami w chwilach poprzednich. Stąd sieci rekurencyjne są skuteczniejsze w zadaniach predykcji szeregów czasowych, przetwarzaniu mowy czy tekstu.

Sieć rekurencyjna Elmana charakteryzuje się częściową rekurencją w postaci sprzężenia zwrotnego między warstwą ukrytą a warstwą wejściową, realizowaną za pośrednictwem jednostkowych opóźnień oznaczanych poprzez blok opisany funkcją z^{-1}. Pełną strukturę tej sieci przedstawiono na rysunku niżej:



Rys. x. Struktura sieci Elmana.

Każdy neuron ukryty ma swój odpowiednik w tak zwanej warstwie kontekstowej, stanowiącej wspólnie z wejściami zewnętrznymi sieci rozszerzoną warstwę wejściową. Warstwę wyjściową tworzą neurony połączone jednokierunkowo tylko z neuronami warstwy ukrytej. Oznaczmy zewnętrzny wektor wymuszający sieci przez **x** (do jego składu należy także sygnał jednostkowy polaryzacji $x_0=1$, stany neuronów ukrytych przez **v**, a sygnały wyjściowe sieci przez **y**.

Załóżmy, że wymuszenie **x** w chwili **k** podane na sieć generuje w warstwie ukrytej sygnały $\mathbf{v_i(k)}$ a w warstwie wyjściowej sygnały $\mathbf{y_i(k)}$ odpowiadające danej chwili. Biorąc pod uwagę bloki opóźniające w warstwie kontekstowej pełny wektor wejściowy sieci w chwili **k** przyjmie postać $\mathbf{x}(k)=[x_0(k),x_1(k),\ldots,x_N(k),v_1(k-1),v_2(k-1),\ldots v_K(k-1)]$

Wagi połączeń synaptycznych warstwy pierwszej (ukrytej) oznaczymy przez $w_{i,j}^{(1)}$ a warstwy drugiej (wyjściowej) przez $w_{i,j}^{(2)}$. Przy oznaczeniu sumy wagowej ii-tego neuronu warstwy ukrytej przez u_i, a jego sygnału wyjściowego przez v_i, otrzymuje się:

$$u_i(k) = \sum_{j=0}^{N+K} w_{ij}^{(1)} x_j(k)$$

$$v_i(k) = f_1\left(u_i(k)\right)$$

Wagi $w_{i,j}^{(2)}$ tworzą z kolei macierz $\mathbf{W}^{(2)}$ opisującą wagi połączeń synaptycznych neuronów warstwy wyjściowej, a $f_2(g_i)$ jest funkcją aktywacji i-tego neuronu tej warstwy.
Algorytm uczenia sieci Elmana wykorzystuje metodę gradientową.

Wykorzystanie sieci Elmana pokazane jest na przykładzie programu 3.

```
import neurolab as nl
import numpy as np

# Create train samples
i1 = np.sin(np.arange(0, 20))
i2 = np.sin(np.arange(0, 20)) * 2

t1 = np.ones([1, 20])
t2 = np.ones([1, 20]) * 2

input = np.array([i1, i2, i1, i2]).reshape(20 * 4, 1)
target = np.array([t1, t2, t1, t2]).reshape(20 * 4, 1)

# Create network with 2 layers
net = nl.net.newelm([[-2, 2]], [20, 1], [nl.trans.TanSig(), nl.trans.PureLin()])
# Set initialized functions and init
net.layers[0].initf = nl.init.InitRand([-0.1, 0.1], 'wb')
net.layers[1].initf= nl.init.InitRand([-0.1, 0.1], 'wb')
net.init()
# Train network
error = net.train(input, target, epochs=500, show=100, goal=0.01)
# Simulate network
output = net.sim(input)

# Plot result
import pylab as pl
pl.subplot(211)
pl.plot(error)
pl.xlabel('Epoch number')
pl.ylabel('Train error (default MSE)')

pl.subplot(212)
pl.plot(target.reshape(80))
pl.plot(output.reshape(80))
pl.legend(['train target', 'net output'])
pl.show()
```

Wykorzystując powyższy program sprawdzić na ile dokładnie sieć Elmana odwzorowuje sygnał prostokątny [1 2] dla różnej liczby neuronów
Wyniki zapisać w tabeli:

Liczba neuronów w warstwie ukrytej	3	5	20	50	100
Średniokwadratowy błąd odwzorowania					

Na podstawie danych z tabeli wyżej sformułować wnioski jak liczba neuronów w warstwie ukrytej wpływa na jakość uczenia sieci.