Zajęcie 4. Użycie sztucznych sieci neuronowych

Abstract

Celem jest uczenie maszynowe nadzorowane za pomocą procedury nauczania sztucznej sieci neuronowej

1. Wprowadzenie

Sztuczne sieci neuronowe są techniką inteligentną powstałą z inspiracji strukturą i możliwościami analitycznymi ludzkiego mózgu. Zastosowanie sieci neuronowych wiąże się z możliwością ich uczenia do konkretnego zadania, tak iż sieć taka staje się matematycznym modelem systemu czy procesu, który podlega analizie. Niezależnie od konkretnej struktury, każda sieć neuronowa zbudowana jest z pojedynczych modułów obliczeniowych, zwanych neuronami

Pojedynczy neuron realizuje obliczeniu sumy ważonej sygnałów wejściowych z nałożoną nieliniową funkcją aktywacji. Jego wyjście określone jest zależnością

$$a = f(X * W + b)$$

gdzie: X – wektor sygnałów wejściowych o rozmiarze $[n \times 1]$, W – wektor wag synaptycznych o rozmiarze $[1 \times n]$, $f(e)$ – funkcja aktywacji neuronu, zwykle nieliniowa, b – współczynnik przesunięcia (ang. bias).

Projektowanie neuronowego układu rozpoznającego lub decyzyjnego zarówno jedno-neuronowego, jak i wielowarstwowego, jest możliwe poprzez **uczenie sieci neuronowej** z wykorzystaniem odpowiednio przygotowanych wzorców uczących i stosownego algorytmu uczenia. W niniejszym skrypcie nie ma miejsca na większe szczegóły w tym zakresie, wystarczy tutaj stwierdzenie, że uczenie to polega na dostosowaniu/dostrojeniu współczynników wagowych (wag synaptycznych) neuronu/neuronów, aby uzyskać jak największą zgodność wyjścia neuronu/sieci a z wymaganym sygnałem wyjściowym t, Rys. 2 Metoda ta nosi nazwę uczenia z nauczycielem.

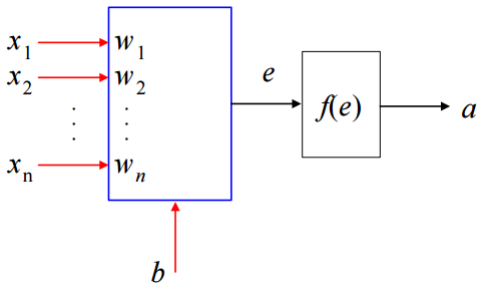

Figure 1: Model pojedynczego neuronu z nieliniową funkcją aktywacji

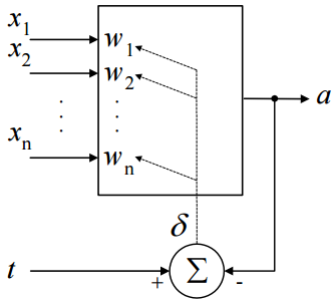

Figure 2: Ilustracja procesu uczenia pojedynczego neuronu

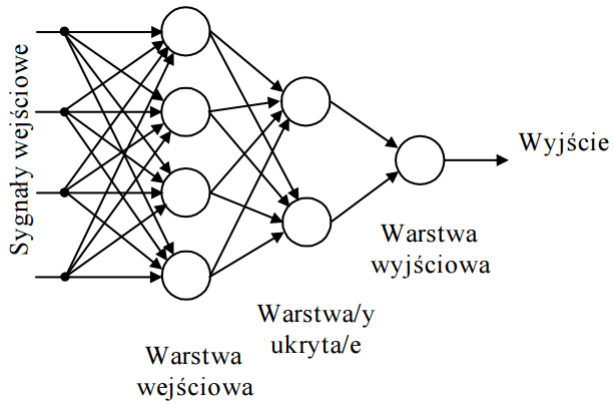

Figure 3: Struktura wielowarstwowego perceptronu

Istnieją również metody uczenia bez nauczyciela (tzn. bez określania i podawania żądanego wyjścia neuronu/sieci), oparte na idei samoorganizacji danych. Nie będą one jednak tutaj opisywane. Szczegółowe algorytmy uczenia można znaleźć w literaturze, jak również w pomocy paczki **neuralnet** programu R, gdzie można skorzystać z gotowych procedur uczenia i testowania sieci neuronowych. Ze względu na to, że możliwości obliczeniowe pojedynczego neuronu są ograniczone, praktyczne zastosowanie znalazły tzw. sieci neuronowe będące układami pojedynczych neuronów ułożonych w pewne struktury. Do najczęściej stosowanych sieci neuronowych należy zaliczyć:

- sieci typu wielowarstwowy perceptron, Rys. 3 – trój- lub czterowarstwowe sieci jednokierunkowe, sieci z radialną funkcją bazową,
- sieci typu Hoppfielda – maszyny Bolzmanna, Gaussa, sieci chaotyczne,
- sieci Kohonena – dwu- lub trójwymiarowe sieci kratowe,
- inne – rzadziej stosowane.

Przygotowanie efektywnego modułu decyzyjnego opartego na technice SSN wiąże się z rozwiązaniem zadań należących do dwóch grup problemowych:

- wybór optymalnej sieci neuronowej (rodzaj sieci, liczba warstw i neuronów w poszczególnych warstwach sieci, rodzaj funkcji aktywacji neuronów),

- uczenie sieci (algorytm uczenia, początkowe wartości wag synaptycznych i współczynników przesunięcia, dobór sygnałów uczących i testujących).

Istotnym problemem jest także wybór i przygotowanie zestawu reprezentatywnych „wzorców uczących", tj. sygnałów, z pomocą których dokonywane będzie uczenie sieci neuronowej. Sygnały wejściowe sieci neuronowych otrzymywane są z reguły w wyniku symulacji pracy obiektu, ponieważ przebiegi rzeczywiste (z rejestratorów zainstalowanych w systemie) są z jednej strony trudno dostępne, z drugiej zaś nie tworzą odpowiednio wyczerpującej bazy sygnałowej, pokrywającej całą różnorodność sytuacji awaryjnych i stanów pracy normalnej rozważanej sieci. Sygnały te mogą jedynie uzupełniać wzorce uczące wygenerowane w ramach symulacji, mogą być także wykorzystane do późniejszego testowania gotowego układu neuronowego.

Powszechnie przyjętą praktyką jest podział zbioru dostępnych sygnałów na dwa podzbiory (np. 50drugi zaś do weryfikacji działania układu. To ostatnie jest bardzo istotne, ponieważ poprawnie zaprojektowany układ neuronowy powinien także posiadać zdolność generalizacji zdobytej wiedzy, objawiającą się stabilnością wobec sygnałów wejściowych nieprezentowanych podczas uczenia. Oznacza to, że układ powinien generować wła- ściwe sygnały wyjściowe także dla nowych, niewidzianych wcześniej sygnałów wej- ściowych.

Dodatkowe trudności związane są z wyborem „najlepszych" sygnałów dla danego zadania klasyfikacji wiążą się z rozważeniem następujących kwestii:

- liczba i rodzaj sygnałów wejściowych sieci (składowych wektora uczącego i testującego) – niosących w miarę możliwości maksymalną ilość informacji o zjawiskach mających podlegać klasyfikacji,

- wstępne przetwarzanie sygnałów dostarczanych z systemu elektroenergetycznego (algorytmy cyfrowego pomiaru wielkości decyzyjnych),

- długość okna decyzyjnego (ilość próbek sygnałów w wektorze wejściowym sieci).

2. Implementacja metody sieci neuronowej w R

Metoda sztucznej sieci neuronowych jest zrealizowana w paczce **neuralnet**
https://cran.r-project.org/web/packages/ahp/index.html
Dokumentacja znajduje się https://cran.r-project.org/web/packages/neuralnet/neuralnet.pdf
Oto przykłady stosowań http://gekkoquant.com/2012/05/26/neural-networks-with-r-simple-example
https://www.r-bloggers.com/fitting-a-neural-network-in-r-neuralnet-package/

3. Instrukcji do korzystania neuralnet

http://www.kdnuggets.com/2016/08/begineers-guide-neural-networks-r.html
https://www.r-bloggers.com/using-neural-networks-for-credit-scoring-a-simple-example/
https://journal.r-project.org/archive/2010-1/RJournal_2010-1_Guenther+Fritsch.pdf

4. Opracowanie pliku programowego w R

4.1. Ładowanie danych dla nauczania sieci

```
 #install.packages('neuralnet')
library("neuralnet")

#Going to create a neural network to perform prediction
#Type ?neuralnet for more information on the neuralnet library

#Generate training data
#And store them as a dataframe
traininginput <-  as.data.frame(matrix(c(32,1000,
                                         16,500,
                                       8,256,
                                       64,500,
                                       32,500,
                                       64,1000,
                                       16,500,
                                       8,256,
                                       64,500,
```

```
                                                32,500),nrow=10,ncol=2))
trainingoutput <- c(4000,3000,1100,2200,2500,4200)
```

4.2. Normalizacja danych

```
# Create Vector of Column Max and Min Values
maxs <- apply(traininginput[,1:2], 10, max)
mins <- apply(traininginput[,1:2], 10, min)

# Use scale() and convert the resulting matrix to a data frame
scaled.traininginput <- as.data.frame(scale(traininginput[,1:2],center =

# Check out results
print(head(scaled.traininginput,10))
```

4.3. Połączenie danych wejściowych i wyjściowych

```
 #Column bind the data into one variable
trainingdata <- cbind(scaled.traininginput,trainingoutput)
colnames(trainingdata) <- c("RAM","HDD","Price")

print(trainingdata)
```

4.4. Nauczanie sieci neuronowej

```
 #Train the neural network
#Going to have c(3,2) hidden layers
#Threshold is a numeric value specifying the threshold for the partial
#derivatives of the error function as stopping criteria.
net.price <- neuralnet(Price~RAM+HDD,trainingdata, hidden=c(3,2), thresh
print(net.price)
```

4.5. Wyświetlenie sieci neuronowej

```
 #Plot the neural network
plot(net.price)
```

4.6. Prognozowanie z pomocą sieci neuronowej

```
 #Test the neural network on some training data
testdata <- as.data.frame(matrix(c(64,1000,
                                   32,500,
```

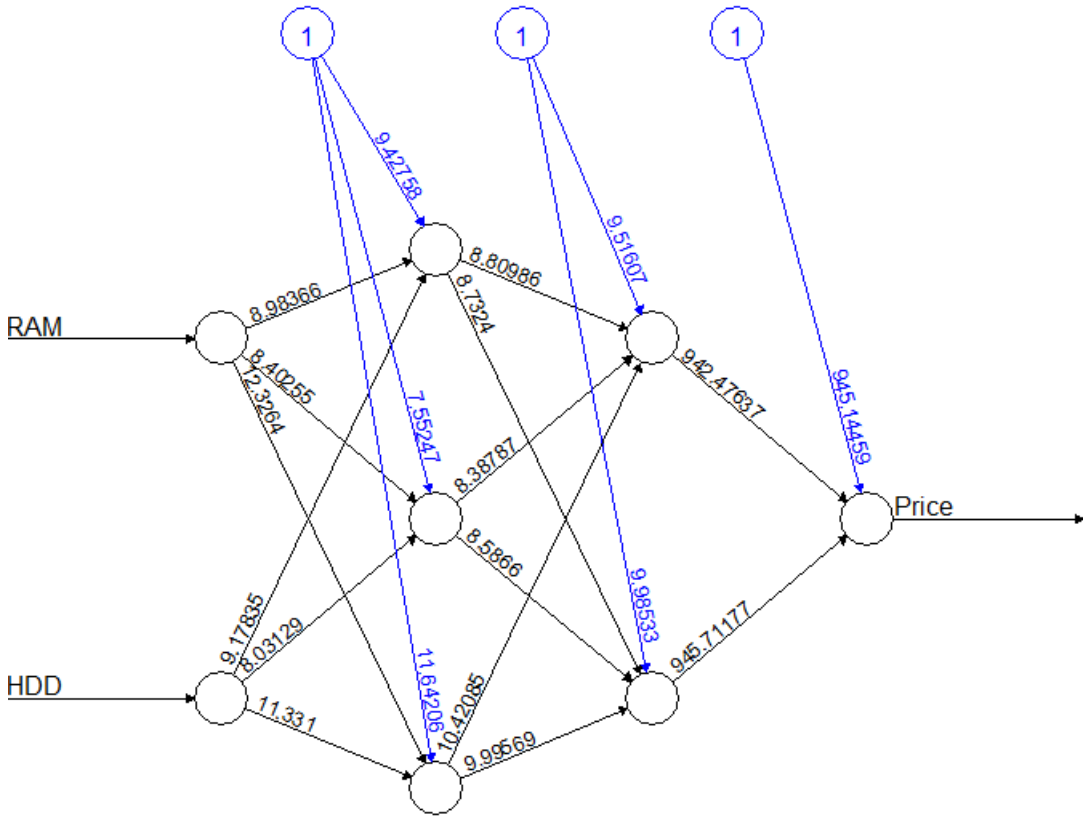

Figure 4: Sieć neuronowa dla danych wejściowych znormalizowanych

```
                                                    8,100),nrow=3,ncol=2)) #Generate son

scaled.testdata <- as.data.frame(scale(testdata[,1:2],center = mins, sca

net.results <- compute(net.price, scaled.testdata) #Run them through the

#Lets see what properties net.price has
ls(net.results)

#Lets see the results
print(net.results$net.result)
```

5. Warianty Zadania

Uwaga! Sprawozdania muszą być sporządzane zgodnie ze wzorem. Oprócz tego pliki źródłowe oraz obrazy muszą być zachowane w zdalnym repozytorium.

Zadanie 1. Zadanie dotyczy modelowania funkcji matematycznych za pomocą sztucznej sieci neuronowej używając paczkę neuralnet. Rozważamy źmienną niezależną x. Celem jest uzyskanie sieci neuronowej (zmieniając zarówno ilość warstw ukrytych jak i ilość neuronów) wypełniającej warunek $Error < 0.01$. Sprawozdania w postaci pliku R oraz obrazu sieci neuronowej zachować w zdalnym repozytorium (np Github), link na który wysłać w mailu z tematem **APU_4_Gr_numer_grupy** na adres mailowy **vmartsenyuk@ath.bielsko.pl**

1. $f(x) = x^3 + 2 * x, \quad x \in [1; 100]$
2. $f(x) = \cos(x)^{\sin(x)}, \quad x \in [0; 2\pi]$
3. $f(x) = e^{\sqrt{x}}, \quad x \in [1; 16]$
4. $f(x) = \log x^2, \quad x \in [1; 10]$
5. $f(x) = x^2 + e^{-x}, \quad x \in [1; 10]$
6. $f(x) = \cos(x^2), x \in [1, 3]$

7. $f(x) = x^3 - 2 * x, \quad x \in [1; 100]$

8. $f(x) = \sin(x)^{\cos(x)}, \quad x \in [0; 2\pi]$

9. $f(x) = \sin(x^2), x \in [1, 3]$

10. $f(x) = x^2 - e^{-x}, \quad x \in [1; 10]$

11. $f(x) = x^{-1/4}, \quad x \in [1; 10]$

12. $f(x) = \frac{1}{\sqrt{x}}, \quad x \in [1; 10]$

Zadanie 2. Zadanie dotyczy prognozowania ceny urządzeń RTV AGD (error ≤ 100 zł), określonych na Zajęciu 1. Używając metody sztucznych sieci neuronowych opracować plik w języku R z wykorzystaniem paczki *neuralnet*. Sprawozdania w postaci pliku R, obrazu sieci neuronowej oraz wyników z konsolu (dowolny plik tekstowy) zachować w zdalnym repozytorium (np Github) link na który wysłać w mailu z tematem **APU_4_Gr_numer_grupy** na adres mailowy **vmartsenyuk@ath.bielsko.pl**

References