SPRAWOZDANIE

Zajęcia: Analiza Procesów Uczenia

Prowadzący: prof. dr hab. inż. Vasyl Martsenyuk

|  |  |
| --- | --- |
| Laboratorium Nr 3  Data 02.12.2023  Temat: „Użycie sztucznych sieci neuronowych"  Wariant 6 | Adam Kubliński  Informatyka  II stopień, niestacjonarne, zaoczne,  I semestr, gr. 1A |

Repozytorium GitHub: <https://github.com/Adamadacho/APU-LAB3-NN-APP.git>

# Polecenie: wariant 6 zadania

* 1. Zadanie I

Zadanie dotyczy modelowania funkcji matematycznych przy użyciu sztucznej sieci neuronowej za pomocą pakietu neuralnet. Rozważamy zmienną niezależną x. Celem jest uzyskanie sieci neuronowej, zmieniając zarówno ilość warstw ukrytych, jak i ilość neuronów, spełniającej warunek Error < 0.01.

* 1. Zadanie II

Zadanie dotyczy prognozowania cen urządzeń RTV AGD (błąd ≤ 100 zł), określonych na Zjeździe 1. Przy użyciu metody sztucznych sieci neuronowych należy opracować plik w języku R z wykorzystaniem pakietu neuralnet.

# Opis opracowanego programu dla Zadanie I

## Zastosowany kod

Model matematyczny został stworzony na podstawie wzoru z wariantu 6 (wzór poniżej)



By przeprowadzić udane uruchomienie programu należy w pierwszej kolejności zainstalować i wczytać bibliotekę „neuralnet” i wprowadzić funkcję z postaci wzoru do kodu zrozumiałego dla programu RStudio.

#Zadanie wariant 6

#f(x) = cos(x^2); x zawiera się w [1; 3]

# Instalacja pakietu neuralnet, jeśli jeszcze nie jest zainstalowany

# install.packages("neuralnet")

# Załaduj potrzebne pakiety

library(neuralnet)

# Funkcja do generowania danych treningowych

generate\_data <- function(n) {

x <- seq(1, 3, length.out = n)

y <- cos(x^2)

data.frame(x = x, y = y)

}

Następnie ustawiono ziarno losowości, by była możliwość uzyskiwania tych samych wyników przy każdym uruchomieniu programu. Potem wprowadzono komendy dotyczące generowania danych treningowych w liczbie 100. Wprowadzono warunek z treści zadania dotyczący progu błędu, ustalono ilość warstw ukrytych oraz początkową liczbę neuronów w każdej warstwie ukrytej. Utworzono pętlę, która dostosowuje parametry sieci neuronowej.

# Generuj dane treningowe

n <- 100

training\_data <- generate\_data(n)

# Inicjalizacja zmiennych

error\_threshold <- 0.01

error <- Inf

hidden\_layers <- 1

neurons <- 5

# Pętla do dostosowywania parametrów sieci

while (error > error\_threshold) {

model <- neuralnet(

y ~ x,

data = training\_data,

hidden = rep(neurons, hidden\_layers),

linear.output = TRUE

)

# Obliczanie błędu

predictions <- predict(model, training\_data)

error <- sum((predictions - training\_data$y)^2) / n

# Dostosuj parametry

if (error > error\_threshold) {

if (neurons < 20) {

neurons <- neurons + 1

} else {

neurons <- 5

hidden\_layers <- hidden\_layers + 1

}

}

}

Na zakończenie zaprogramowano, by program generował wyniki takie jak: ostateczna struktura sieci, finałowy błąd treningowy sieci, oraz wykresy.

# Wydrukuj ostateczną strukturę sieci

print(model)

# Wydrukuj błąd

cat("Final training error:", error, "\n")

# Wykres danych treningowych i rzeczywistej funkcji

plot(training\_data$x, training\_data$y, col = "blue", pch = 20, main = "Neural Network Fit - Training Data and True Function")

lines(training\_data$x, cos(training\_data$x^2), col = "black", lwd = 2)

legend("topleft", legend = c("Training Data", "True Function"), col = c("blue", "black"), pch = c(20, NA), lwd = c(NA, 2))

# Wykres danych treningowych i predykcji modelu

plot(training\_data$x, training\_data$y, col = "blue", pch = 20, main = "Neural Network Fit - Training Data and Model Prediction")

lines(training\_data$x, predict(model, training\_data), col = "red", lwd = 2)

legend("topleft", legend = c("Training Data", "Model Prediction"), col = c("blue", "red"), pch = c(20, NA), lwd = c(NA, 2))

# Testowanie modelu na nowych danych

new\_data <- data.frame(x = seq(1, 3, length.out = 100))

predictions <- predict(model, newdata = new\_data)

# Wykres predykcji na nowych danych

plot(new\_data$x, predictions, col = "green", type = "l", lwd = 2, ylab = "Predicted f(x)", xlab = "x", main = "Model Prediction on New Data")

lines(new\_data$x, cos(new\_data$x^2), col = "black", lwd = 2)

legend("topleft", legend = c("Model Prediction", "True Function"), col = c("green", "black"), lwd = c(2, 2))

## Tabela zawierająca dane i wynik

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

## Model sieci neuronowej

Obraz zawierający szkic, diagram, linia

Opis wygenerowany automatycznie

## Wykresy

Obraz zawierający tekst, diagram, Wykres, linia

Opis wygenerowany automatycznie

Obraz zawierający tekst, Wykres, diagram, linia

Opis wygenerowany automatycznie

Obraz zawierający tekst, diagram, Wykres, linia

Opis wygenerowany automatycznie

# Opis opracowanego programu dla Zadanie II

## Zastosowany kod

Napisano kod, którego celem jest prognozowanie cen pralek w oparciu o dane z laboratorium 1. Z dostępnych danych stworzono dwie osobne grupy: dane wejściowe i dane treningowe. Wykorzystano do tego plik pralki.CSV. Zainstalowano biblioteki „Neuralnet” i „Caret” i wczytano plik z danymi początkowymi. Przeprowadzono operację normalizacji danych oraz wyłączono niektóre elementy danych, które wprowadzały błędne interpretacje, jak na przykład nazwa pralki. Zdefiniowano parametry sieci neuronowej oraz przeprowadzono symulację. W wyniku otrzymano błąd prognozy oraz schemat sieci neuronowej.

# Wczytanie danych z pliku CSV (przykładowy plik "pralki.csv")

data <- read.csv("pralki\_dane.csv")

# 4.2. Normalizacja danych

# Create Vector of Column Max and Min Values

maxs <- apply(data[, 2:3], 2, max)

mins <- apply(data[, 2:3], 2, min)

# Use scale() and convert the resulting matrix to a data frame

scaled\_data <- as.data.frame(scale(data[, 2:3], center = mins, scale = maxs - mins))

# Check out results

print(head(scaled\_data, 10))

# 4.3. Po łączenie danych wejściowych i wyjściowych

# Column bind the data into one variable

# Załóżmy, że dane wyjściowe znajdują się w kolumnie "cena"

training\_data <- cbind(scaled\_data, Cena = data$cena)

colnames(training\_data) <- c("Pojemnosc", "PredkoscWirowania", "Price")

print(training\_data)

# 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 ~ Pojemnosc + PredkoscWirowania, training\_data, hidden = c(3, 2), threshold = 0.01)

print(net\_price)

# 4.5. Wyświetlenie sieci neuronowej

# Plot the neural network

plot(net\_price)

# 4.6. Prognozowanie z pomocą̨ sieci neuronowej

# Test the neural network on some testing data

# Załóżmy, że testowe dane znajdują się w pliku "testowe\_dane.csv"

test\_data <- read.csv("testowe\_dane.csv")

scaled\_test\_data <- as.data.frame(scale(test\_data[, 2:3], center = mins, scale = maxs - mins))

net\_results <- compute(net\_price, scaled\_test\_data)

# Run them through the neural network

# Let's see what properties net\_price has

ls(net\_results)

# Let's see the results

print(net\_results$net.result)

## Tabela zawierająca dane i wyniki

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

## Model sieci neuronowej

Obraz zawierający linia, diagram

Opis wygenerowany automatycznie

# Wnioski

Laboratorium zostało przeprowadzone pomyślnie. Zarówno zadanie I jak i II zostało wykonane oraz osiągnięto zadowalające wyniki. Zaobserwowano wpływ działania różnych kombinacji parametrów sieci neuronowej w celu osiągnięcia jak najbardziej optymalnych modeli. Zapoznano się ze sposobami przygotowania i obróbki wstępnych danych taki jak normalizacja danych. Zauważono znaczący wpływ normalizacji na wielkość błędu prognozy.

Proces budowy sieci neuronowej to proces iteracyjny. Jest to ciągłe doskonalenie kombinacji parametrów sieci jak i dostarczanie kolejnych danych treningowych, dzięki którym model staje się coraz precyzyjniejszy.

# Załączniki

## **Pełny kod programu do zadania I**

#Zadanie wariant 6

#f(x) = cos(x^2); x zawiera się w [1; 3]

# install.packages("neuralnet")

# Załaduj potrzebne pakiety

library(neuralnet)

# Funkcja do generowania danych treningowych

generate\_data <- function(n) {

x <- seq(1, 3, length.out = n)

y <- cos(x^2)

data.frame(x = x, y = y)

}

# Ustaw ziarno losowości w celu reprodukowalności wyników

set.seed(123)

# Generuj dane treningowe

n <- 100

training\_data <- generate\_data(n)

# Inicjalizacja zmiennych

error\_threshold <- 0.01

error <- Inf

hidden\_layers <- 1

neurons <- 5

# Pętla do dostosowywania parametrów sieci

while (error > error\_threshold) {

model <- neuralnet(

y ~ x,

data = training\_data,

hidden = rep(neurons, hidden\_layers),

linear.output = TRUE

)

# Obliczanie błędu

predictions <- predict(model, training\_data)

error <- sum((predictions - training\_data$y)^2) / n

# Dostosuj parametry

if (error > error\_threshold) {

if (neurons < 20) {

neurons <- neurons + 1

} else {

neurons <- 5

hidden\_layers <- hidden\_layers + 1

}

}

}

# Wydrukuj ostateczną strukturę sieci

print(model)

# Wydrukuj błąd

cat("Final training error:", error, "\n")

# Wykres danych treningowych i rzeczywistej funkcji

plot(training\_data$x, training\_data$y, col = "blue", pch = 20, main = "Neural Network Fit - Training Data and True Function")

lines(training\_data$x, cos(training\_data$x^2), col = "black", lwd = 2)

legend("topleft", legend = c("Training Data", "True Function"), col = c("blue", "black"), pch = c(20, NA), lwd = c(NA, 2))

# Wykres danych treningowych i predykcji modelu

plot(training\_data$x, training\_data$y, col = "blue", pch = 20, main = "Neural Network Fit - Training Data and Model Prediction")

lines(training\_data$x, predict(model, training\_data), col = "red", lwd = 2)

legend("topleft", legend = c("Training Data", "Model Prediction"), col = c("blue", "red"), pch = c(20, NA), lwd = c(NA, 2))

# Testowanie modelu na nowych danych

new\_data <- data.frame(x = seq(1, 3, length.out = 100))

predictions <- predict(model, newdata = new\_data)

# Wykres predykcji na nowych danych

plot(new\_data$x, predictions, col = "green", type = "l", lwd = 2, ylab = "Predicted f(x)", xlab = "x", main = "Model Prediction on New Data")

lines(new\_data$x, cos(new\_data$x^2), col = "black", lwd = 2)

legend("topleft", legend = c("Model Prediction", "True Function"), col = c("green", "black"), lwd = c(2, 2))

## **Wyciąg z konsoli z zadania I**

#Zadanie wariant 6

>

> #f(x) = cos(x^2); x zawiera się w [1; 3]

>

> # Instalacja pakietu neuralnet, jeśli jeszcze nie jest zainstalowany

> # install.packages("neuralnet")

>

> # Załaduj potrzebne pakiety

> library(neuralnet)

>

> # Funkcja do generowania danych treningowych

> generate\_data <- function(n) {

+ x <- seq(1, 3, length.out = n)

+ y <- cos(x^2)

+ data.frame(x = x, y = y)

+ }

>

> # Ustaw ziarno losowości w celu reprodukowalności wyników

> set.seed(123)

>

> # Generuj dane treningowe

> n <- 100

> training\_data <- generate\_data(n)

>

> # Inicjalizacja zmiennych

> error\_threshold <- 0.01

> error <- Inf

> hidden\_layers <- 1

> neurons <- 5

>

> # Pętla do dostosowywania parametrów sieci

> while (error > error\_threshold) {

+ model <- neuralnet(

+ y ~ x,

+ data = training\_data,

+ hidden = rep(neurons, hidden\_layers),

+ linear.output = TRUE

+ )

+

+ # Obliczanie błędu

+ predictions <- predict(model, training\_data)

+ error <- sum((predictions - training\_data$y)^2) / n

+

+ # Dostosuj parametry

+ if (error > error\_threshold) {

+ if (neurons < 20) {

+ neurons <- neurons + 1

+ } else {

+ neurons <- 5

+ hidden\_layers <- hidden\_layers + 1

+ }

+ }

+ }

>

> # Wydrukuj ostateczną strukturę sieci

> print(model)

$call

neuralnet(formula = y ~ x, data = training\_data, hidden = rep(neurons,

hidden\_layers), linear.output = TRUE)

$response

y

1 0.540302306

2 0.505519678

3 0.469174485

4 0.431280080

5 0.391857985

6 0.350938429

7 0.308560871

8 0.264774515

9 0.219638805

10 0.173223893

11 0.125611075

12 0.076893191

13 0.027174983

14 -0.023426609

15 -0.074782199

16 -0.126749783

17 -0.179174644

18 -0.231889332

19 -0.284713749

20 -0.337455330

21 -0.389909332

22 -0.441859242

23 -0.493077317

24 -0.543325250

25 -0.592354982

26 -0.639909658

27 -0.685724736

28 -0.729529256

29 -0.771047268

30 -0.809999421

31 -0.846104716

32 -0.879082426

33 -0.908654168

34 -0.934546134

35 -0.956491470

36 -0.974232787

37 -0.987524808

38 -0.996137117

39 -0.999857004

40 -0.998492391

41 -0.991874797

42 -0.979862338

43 -0.962342717

44 -0.939236179

45 -0.910498398

46 -0.876123251

47 -0.836145449

48 -0.790642974

49 -0.739739289

50 -0.683605257

51 -0.622460751

52 -0.556575874

53 -0.486271774

54 -0.411920985

55 -0.333947262

56 -0.252824855

57 -0.169077203

58 -0.083274983

59 0.003966483

60 0.091990526

61 0.180103112

62 0.267577557

63 0.353659940

64 0.437575246

65 0.518534180

66 0.595740656

67 0.668399904

68 0.735727142

69 0.796956744

70 0.851351835

71 0.898214203

72 0.936894433

73 0.966802135

74 0.987416134

75 0.998294485

76 0.999084141

77 0.989530132

78 0.969484063

79 0.938911773

80 0.897899964

81 0.846661623

82 0.785540077

83 0.715011493

84 0.635685677

85 0.548305031

86 0.453741531

87 0.352991641

88 0.247169064

89 0.137495300

90 0.025287988

91 -0.088052944

92 -0.201061255

93 -0.312222395

94 -0.419993372

95 -0.522824309

96 -0.619181421

97 -0.707571151

98 -0.786565109

99 -0.854825447

100 -0.911130262

$covariate

[1,] 1.000000

[2,] 1.020202

[3,] 1.040404

[4,] 1.060606

[5,] 1.080808

[6,] 1.101010

[7,] 1.121212

[8,] 1.141414

[9,] 1.161616

[10,] 1.181818

[11,] 1.202020

[12,] 1.222222

[13,] 1.242424

[14,] 1.262626

[15,] 1.282828

[16,] 1.303030

[17,] 1.323232

[18,] 1.343434

[19,] 1.363636

[20,] 1.383838

[21,] 1.404040

[22,] 1.424242

[23,] 1.444444

[24,] 1.464646

[25,] 1.484848

[26,] 1.505051

[27,] 1.525253

[28,] 1.545455

[29,] 1.565657

[30,] 1.585859

[31,] 1.606061

[32,] 1.626263

[33,] 1.646465

[34,] 1.666667

[35,] 1.686869

[36,] 1.707071

[37,] 1.727273

[38,] 1.747475

[39,] 1.767677

[40,] 1.787879

[41,] 1.808081

[42,] 1.828283

[43,] 1.848485

[44,] 1.868687

[45,] 1.888889

[46,] 1.909091

[47,] 1.929293

[48,] 1.949495

[49,] 1.969697

[50,] 1.989899

[51,] 2.010101

[52,] 2.030303

[53,] 2.050505

[54,] 2.070707

[55,] 2.090909

[56,] 2.111111

[57,] 2.131313

[58,] 2.151515

[59,] 2.171717

[60,] 2.191919

[61,] 2.212121

[62,] 2.232323

[63,] 2.252525

[64,] 2.272727

[65,] 2.292929

[66,] 2.313131

[67,] 2.333333

[68,] 2.353535

[69,] 2.373737

[70,] 2.393939

[71,] 2.414141

[72,] 2.434343

[73,] 2.454545

[74,] 2.474747

[75,] 2.494949

[76,] 2.515152

[77,] 2.535354

[78,] 2.555556

[79,] 2.575758

[80,] 2.595960

[81,] 2.616162

[82,] 2.636364

[83,] 2.656566

[84,] 2.676768

[85,] 2.696970

[86,] 2.717172

[87,] 2.737374

[88,] 2.757576

[89,] 2.777778

[90,] 2.797980

[91,] 2.818182

[92,] 2.838384

[93,] 2.858586

[94,] 2.878788

[95,] 2.898990

[96,] 2.919192

[97,] 2.939394

[98,] 2.959596

[99,] 2.979798

[100,] 3.000000

$model.list

$model.list$response

[1] "y"

$model.list$variables

[1] "x"

$err.fct

function (x, y)

{

1/2 \* (y - x)^2

}

<bytecode: 0x0000021ff2719f00>

<environment: 0x0000021f8085aa78>

attr(,"type")

[1] "sse"

$act.fct

function (x)

{

1/(1 + exp(-x))

}

<bytecode: 0x0000021ff28a0ef8>

<environment: 0x0000021f8085af10>

attr(,"type")

[1] "logistic"

$linear.output

[1] TRUE

$data

x y

1 1.000000 0.540302306

2 1.020202 0.505519678

3 1.040404 0.469174485

4 1.060606 0.431280080

5 1.080808 0.391857985

6 1.101010 0.350938429

7 1.121212 0.308560871

8 1.141414 0.264774515

9 1.161616 0.219638805

10 1.181818 0.173223893

11 1.202020 0.125611075

12 1.222222 0.076893191

13 1.242424 0.027174983

14 1.262626 -0.023426609

15 1.282828 -0.074782199

16 1.303030 -0.126749783

17 1.323232 -0.179174644

18 1.343434 -0.231889332

19 1.363636 -0.284713749

20 1.383838 -0.337455330

21 1.404040 -0.389909332

22 1.424242 -0.441859242

23 1.444444 -0.493077317

24 1.464646 -0.543325250

25 1.484848 -0.592354982

26 1.505051 -0.639909658

27 1.525253 -0.685724736

28 1.545455 -0.729529256

29 1.565657 -0.771047268

30 1.585859 -0.809999421

31 1.606061 -0.846104716

32 1.626263 -0.879082426

33 1.646465 -0.908654168

34 1.666667 -0.934546134

35 1.686869 -0.956491470

36 1.707071 -0.974232787

37 1.727273 -0.987524808

38 1.747475 -0.996137117

39 1.767677 -0.999857004

40 1.787879 -0.998492391

41 1.808081 -0.991874797

42 1.828283 -0.979862338

43 1.848485 -0.962342717

44 1.868687 -0.939236179

45 1.888889 -0.910498398

46 1.909091 -0.876123251

47 1.929293 -0.836145449

48 1.949495 -0.790642974

49 1.969697 -0.739739289

50 1.989899 -0.683605257

51 2.010101 -0.622460751

52 2.030303 -0.556575874

53 2.050505 -0.486271774

54 2.070707 -0.411920985

55 2.090909 -0.333947262

56 2.111111 -0.252824855

57 2.131313 -0.169077203

58 2.151515 -0.083274983

59 2.171717 0.003966483

60 2.191919 0.091990526

61 2.212121 0.180103112

62 2.232323 0.267577557

63 2.252525 0.353659940

64 2.272727 0.437575246

65 2.292929 0.518534180

66 2.313131 0.595740656

67 2.333333 0.668399904

68 2.353535 0.735727142

69 2.373737 0.796956744

70 2.393939 0.851351835

71 2.414141 0.898214203

72 2.434343 0.936894433

73 2.454545 0.966802135

74 2.474747 0.987416134

75 2.494949 0.998294485

76 2.515152 0.999084141

77 2.535354 0.989530132

78 2.555556 0.969484063

79 2.575758 0.938911773

80 2.595960 0.897899964

81 2.616162 0.846661623

82 2.636364 0.785540077

83 2.656566 0.715011493

84 2.676768 0.635685677

85 2.696970 0.548305031

86 2.717172 0.453741531

87 2.737374 0.352991641

88 2.757576 0.247169064

89 2.777778 0.137495300

90 2.797980 0.025287988

91 2.818182 -0.088052944

92 2.838384 -0.201061255

93 2.858586 -0.312222395

94 2.878788 -0.419993372

95 2.898990 -0.522824309

96 2.919192 -0.619181421

97 2.939394 -0.707571151

98 2.959596 -0.786565109

99 2.979798 -0.854825447

100 3.000000 -0.911130262

$exclude

NULL

$net.result

$net.result[[1]]

[,1]

[1,] 0.528574667

[2,] 0.498665797

[3,] 0.466312171

[4,] 0.431606588

[5,] 0.394646639

[6,] 0.355535335

[7,] 0.314381766

[8,] 0.271301748

[9,] 0.226418482

[10,] 0.179863184

[11,] 0.131775713

[12,] 0.082305145

[13,] 0.031610313

[14,] -0.020139730

[15,] -0.072765317

[16,] -0.126075892

[17,] -0.179869844

[18,] -0.233934509

[19,] -0.288046329

[20,] -0.341971226

[21,] -0.395465196

[22,] -0.448275163

[23,] -0.500140098

[24,] -0.550792449

[25,] -0.599959859

[26,] -0.647367196

[27,] -0.692738882

[28,] -0.735801470

[29,] -0.776286444

[30,] -0.813933140

[31,] -0.848491693

[32,] -0.879725880

[33,] -0.907415663

[34,] -0.931359278

[35,] -0.951374621

[36,] -0.967299754

[37,] -0.978992325

[38,] -0.986327779

[39,] -0.989196326

[40,] -0.987498757

[41,] -0.981141382

[42,] -0.970030563

[43,] -0.954067511

[44,] -0.933144220

[45,] -0.907141488

[46,] -0.875930013

[47,] -0.839375367

[48,] -0.797347366

[49,] -0.749733866

[50,] -0.696458433

[51,] -0.637500750

[52,] -0.572918081

[53,] -0.502865765

[54,] -0.427614617

[55,] -0.347563321

[56,] -0.263244331

[57,] -0.175322455

[58,] -0.084586043

[59,] 0.008068635

[60,] 0.101658603

[61,] 0.195139434

[62,] 0.287432758

[63,] 0.377454287

[64,] 0.464140791

[65,] 0.546474729

[66,] 0.623505597

[67,] 0.694367401

[68,] 0.758291962

[69,] 0.814618078

[70,] 0.862796738

[71,] 0.902392781

[72,] 0.933083450

[73,] 0.954654381

[74,] 0.966993545

[75,] 0.970083662

[76,] 0.963993559

[77,] 0.948868894

[78,] 0.924922597

[79,] 0.892425340

[80,] 0.851696262

[81,] 0.803094133

[82,] 0.747009099

[83,] 0.683855081

[84,] 0.614062887

[85,] 0.538074058

[86,] 0.456335445

[87,] 0.369294490

[88,] 0.277395179

[89,] 0.181074626

[90,] 0.080760234

[91,] -0.023132625

[92,] -0.130202455

[93,] -0.240063066

[94,] -0.352344639

[95,] -0.466694473

[96,] -0.582777457

[97,] -0.700276318

[98,] -0.818891661

[99,] -0.938341855

[100,] -1.058362785

$weights

$weights[[1]]

$weights[[1]][[1]]

[,1] [,2] [,3] [,4] [,5]

[1,] -0.9669898 4.668664 -13.282858 8.828453 4.361946

[2,] -2.6437510 -2.545189 6.627911 -4.165032 -1.733384

$weights[[1]][[2]]

[,1]

[1,] -3.881444

[2,] -18.935255

[3,] 5.114850

[4,] -2.892942

[5,] -18.099699

[6,] 19.593338

$generalized.weights

$generalized.weights[[1]]

[,1]

[1,] -5.69260483

[2,] -6.16702070

[3,] -6.67228761

[4,] -7.23353737

[5,] -7.88460479

[6,] -8.67403924

[7,] -9.67666946

[8,] -11.01755973

[9,] -12.92709619

[10,] -15.88747208

[11,] -21.11539449

[12,] -32.84009491

[13,] -82.87675139

[14,] 125.81070869

[15,] 33.60919644

[16,] 18.68371446

[17,] 12.58723404

[18,] 9.28170791

[19,] 7.21228920

[20,] 5.79773137

[21,] 4.77141873

[22,] 3.99381367

[23,] 3.38477948

[24,] 2.89499302

[25,] 2.49241752

[26,] 2.15532544

[27,] 1.86844617

[28,] 1.62071828

[29,] 1.40391589

[30,] 1.21177523

[31,] 1.03941932

[32,] 0.88296608

[33,] 0.73925223

[34,] 0.60563142

[35,] 0.47981992

[36,] 0.35977219

[37,] 0.24357376

[38,] 0.12934220

[39,] 0.01512854

[40,] -0.10118765

[41,] -0.22201693

[42,] -0.35020103

[43,] -0.48918885

[44,] -0.64326609

[45,] -0.81786903

[46,] -1.02003270

[47,] -1.25905916

[48,] -1.54755534

[49,] -1.90311075

[50,] -2.35112706

[51,] -2.92982601

[52,] -3.69964678

[53,] -4.76223061

[54,] -6.30265144

[55,] -8.69647420

[56,] -12.83740813

[57,] -21.49259243

[58,] -49.56048196

[59,] 576.99055052

[60,] 50.79495623

[61,] 29.33195060

[62,] 22.07450826

[63,] 18.64881137

[64,] 16.85230545

[65,] 15.94494774

[66,] 15.62155420

[67,] 15.74710114

[68,] 16.26536610

[69,] 17.15422255

[70,] 18.37787371

[71,] 19.77523001

[72,] 20.73728368

[73,] 19.39954909

[74,] 11.95747720

[75,] -2.58947480

[76,] -15.17029323

[77,] -19.97284326

[78,] -20.15279463

[79,] -18.90747751

[80,] -17.52873360

[81,] -16.40593981

[82,] -15.63424757

[83,] -15.23620610

[84,] -15.24013046

[85,] -15.72095994

[86,] -16.85243331

[87,] -19.03007932

[88,] -23.25806726

[89,] -32.84311954

[90,] -68.12549425

[91,] 220.74676842

[92,] 36.50670068

[93,] 18.47874867

[94,] 11.77768432

[95,] 8.33589267

[96,] 6.27021936

[97,] 4.91005580

[98,] 3.95734865

[99,] 3.25980255

[100,] 2.73171084

$startweights

$startweights[[1]]

$startweights[[1]][[1]]

[,1] [,2] [,3] [,4] [,5]

[1,] -0.5604756 1.55870831 0.1292877 0.4609162 -0.6868529

[2,] -0.2301775 0.07050839 1.7150650 -1.2650612 -0.4456620

$startweights[[1]][[2]]

[,1]

[1,] 1.2240818

[2,] 0.3598138

[3,] 0.4007715

[4,] 0.1106827

[5,] -0.5558411

[6,] 1.7869131

$result.matrix

[,1]

error 4.175820e-02

reached.threshold 9.973199e-03

steps 8.759200e+04

Intercept.to.1layhid1 -9.669898e-01

x.to.1layhid1 -2.643751e+00

Intercept.to.1layhid2 4.668664e+00

x.to.1layhid2 -2.545189e+00

Intercept.to.1layhid3 -1.328286e+01

x.to.1layhid3 6.627911e+00

Intercept.to.1layhid4 8.828453e+00

x.to.1layhid4 -4.165032e+00

Intercept.to.1layhid5 4.361946e+00

x.to.1layhid5 -1.733384e+00

Intercept.to.y -3.881444e+00

1layhid1.to.y -1.893525e+01

1layhid2.to.y 5.114850e+00

1layhid3.to.y -2.892942e+00

1layhid4.to.y -1.809970e+01

1layhid5.to.y 1.959334e+01

attr(,"class")

[1] "nn"

>

> # Wydrukuj błąd

> cat("Final training error:", error, "\n")

Final training error: 0.000835164

>

> # Wykres danych treningowych i rzeczywistej funkcji

> plot(training\_data$x, training\_data$y, col = "blue", pch = 20, main = "Neural Network Fit - Training Data and True Function")

> lines(training\_data$x, cos(training\_data$x^2), col = "black", lwd = 2)

> legend("topleft", legend = c("Training Data", "True Function"), col = c("blue", "black"), pch = c(20, NA), lwd = c(NA, 2))

>

> # Wykres danych treningowych i predykcji modelu

> plot(training\_data$x, training\_data$y, col = "blue", pch = 20, main = "Neural Network Fit - Training Data and Model Prediction")

> lines(training\_data$x, predict(model, training\_data), col = "red", lwd = 2)

> legend("topleft", legend = c("Training Data", "Model Prediction"), col = c("blue", "red"), pch = c(20, NA), lwd = c(NA, 2))

>

> # Testowanie modelu na nowych danych

> new\_data <- data.frame(x = seq(1, 3, length.out = 100))

> predictions <- predict(model, newdata = new\_data)

>

> # Wykres predykcji na nowych danych

> plot(new\_data$x, predictions, col = "green", type = "l", lwd = 2, ylab = "Predicted f(x)", xlab = "x", main = "Model Prediction on New Data")

> lines(new\_data$x, cos(new\_data$x^2), col = "black", lwd = 2)

> legend("topleft", legend = c("Model Prediction", "True Function"), col = c("green", "black"), lwd = c(2, 2))

> View(new\_data)

## **Pełny kod programu do zadania II**

# Wczytanie danych z pliku CSV (przykładowy plik "pralki.csv")

data <- read.csv("pralki\_dane.csv")

# 4.2. Normalizacja danych

# Create Vector of Column Max and Min Values

maxs <- apply(data[, 2:3], 2, max)

mins <- apply(data[, 2:3], 2, min)

# Use scale() and convert the resulting matrix to a data frame

scaled\_data <- as.data.frame(scale(data[, 2:3], center = mins, scale = maxs - mins))

# Check out results

print(head(scaled\_data, 10))

# 4.3. Po łączenie danych wejściowych i wyjściowych

# Column bind the data into one variable

# Załóżmy, że dane wyjściowe znajdują się w kolumnie "cena"

training\_data <- cbind(scaled\_data, Cena = data$cena)

colnames(training\_data) <- c("Pojemnosc", "PredkoscWirowania", "Price")

print(training\_data)

# 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 ~ Pojemnosc + PredkoscWirowania, training\_data, hidden = c(3, 2), threshold = 0.01)

print(net\_price)

# 4.5. Wyświetlenie sieci neuronowej

# Plot the neural network

plot(net\_price)

# 4.6. Prognozowanie z pomocą̨ sieci neuronowej

# Test the neural network on some testing data

# Załóżmy, że testowe dane znajdują się w pliku "testowe\_dane.csv"

test\_data <- read.csv("testowe\_dane.csv")

scaled\_test\_data <- as.data.frame(scale(test\_data[, 2:3], center = mins, scale = maxs - mins))

net\_results <- compute(net\_price, scaled\_test\_data)

# Run them through the neural network

# Let's see what properties net\_price has

ls(net\_results)

# Let's see the results

print(net\_results$net.result)

## **Wyciąg z konsoli z zadania II**

# Wczytanie danych z pliku CSV (przykładowy plik "pralki.csv")

> data <- read.csv("pralki\_dane.csv")

>

> # 4.2. Normalizacja danych

> # Create Vector of Column Max and Min Values

> maxs <- apply(data[, 2:3], 2, max)

> mins <- apply(data[, 2:3], 2, min)

>

> # Use scale() and convert the resulting matrix to a data frame

> scaled\_data <- as.data.frame(scale(data[, 2:3], center = mins, scale = maxs - mins))

>

> # Check out results

> print(head(scaled\_data, 10))

pojemnosc predkoscWirowania

1 0.00 0.0

2 0.50 0.5

3 1.00 1.0

4 0.25 0.0

5 0.50 0.5

6 1.00 1.0

7 0.00 0.0

8 0.75 0.5

9 0.25 1.0

10 0.50 0.0

>

> # 4.3. Po łączenie danych wejściowych i wyjściowych

> # Column bind the data into one variable

>

> # Załóżmy, że dane wyjściowe znajdują się w kolumnie "cena"

> training\_data <- cbind(scaled\_data, Cena = data$cena)

> colnames(training\_data) <- c("Pojemnosc", "PredkoscWirowania", "Price")

> print(training\_data)

Pojemnosc PredkoscWirowania Price

1 0.00 0.0 400

2 0.50 0.5 500

3 1.00 1.0 600

4 0.25 0.0 450

5 0.50 0.5 550

6 1.00 1.0 650

7 0.00 0.0 420

8 0.75 0.5 520

9 0.25 1.0 620

10 0.50 0.0 430

11 0.75 0.5 530

12 1.00 1.0 630

13 0.00 0.0 410

14 0.50 0.5 510

15 0.75 1.0 610

>

> # 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 ~ Pojemnosc + PredkoscWirowania, training\_data, hidden = c(3, 2), threshold = 0.01)

> print(net\_price)

$call

neuralnet(formula = Price ~ Pojemnosc + PredkoscWirowania, data = training\_data,

hidden = c(3, 2), threshold = 0.01)

$response

Price

1 400

2 500

3 600

4 450

5 550

6 650

7 420

8 520

9 620

10 430

11 530

12 630

13 410

14 510

15 610

$covariate

Pojemnosc PredkoscWirowania

[1,] 0.00 0.0

[2,] 0.50 0.5

[3,] 1.00 1.0

[4,] 0.25 0.0

[5,] 0.50 0.5

[6,] 1.00 1.0

[7,] 0.00 0.0

[8,] 0.75 0.5

[9,] 0.25 1.0

[10,] 0.50 0.0

[11,] 0.75 0.5

[12,] 1.00 1.0

[13,] 0.00 0.0

[14,] 0.50 0.5

[15,] 0.75 1.0

$model.list

$model.list$response

[1] "Price"

$model.list$variables

[1] "Pojemnosc" "PredkoscWirowania"

$err.fct

function (x, y)

{

1/2 \* (y - x)^2

}

<bytecode: 0x0000018989408008>

<environment: 0x000001898a8b27f0>

attr(,"type")

[1] "sse"

$act.fct

function (x)

{

1/(1 + exp(-x))

}

<bytecode: 0x0000018989409aa0>

<environment: 0x000001898a8b2da0>

attr(,"type")

[1] "logistic"

$linear.output

[1] TRUE

$data

Pojemnosc PredkoscWirowania Price

1 0.00 0.0 400

2 0.50 0.5 500

3 1.00 1.0 600

4 0.25 0.0 450

5 0.50 0.5 550

6 1.00 1.0 650

7 0.00 0.0 420

8 0.75 0.5 520

9 0.25 1.0 620

10 0.50 0.0 430

11 0.75 0.5 530

12 1.00 1.0 630

13 0.00 0.0 410

14 0.50 0.5 510

15 0.75 1.0 610

$exclude

NULL

$net.result

$net.result[[1]]

[,1]

[1,] 522

[2,] 522

[3,] 522

[4,] 522

[5,] 522

[6,] 522

[7,] 522

[8,] 522

[9,] 522

[10,] 522

[11,] 522

[12,] 522

[13,] 522

[14,] 522

[15,] 522

$weights

$weights[[1]]

$weights[[1]][[1]]

[,1] [,2] [,3]

[1,] 8.786959 9.577506 9.015703

[2,] 8.537569 10.023035 10.779346

[3,] 9.394682 10.445411 8.432529

$weights[[1]][[2]]

[,1] [,2]

[1,] 9.611780 7.373112

[2,] 9.284223 9.657527

[3,] 8.401455 10.007245

[4,] 9.702894 10.011827

$weights[[1]][[3]]

[,1]

[1,] 173.4392

[2,] 174.1949

[3,] 174.3658

$generalized.weights

$generalized.weights[[1]]

[,1] [,2]

[1,] 0 0

[2,] 0 0

[3,] 0 0

[4,] 0 0

[5,] 0 0

[6,] 0 0

[7,] 0 0

[8,] 0 0

[9,] 0 0

[10,] 0 0

[11,] 0 0

[12,] 0 0

[13,] 0 0

[14,] 0 0

[15,] 0 0

$startweights

$startweights[[1]]

$startweights[[1]][[1]]

[,1] [,2] [,3]

[1,] -0.58144118 0.0091059 -0.5526966

[2,] -0.83083095 0.4546348 1.2109464

[3,] 0.02628232 0.8770108 -1.1358712

$startweights[[1]][[2]]

[,1] [,2]

[1,] 0.7433801 -2.5268876

[2,] 0.4158233 -0.2424731

[3,] -0.4669446 0.1072455

[4,] 0.8344940 0.1118270

$startweights[[1]][[3]]

[,1]

[1,] -0.85010352

[2,] -0.09442104

[3,] 0.07649794

$result.matrix

[,1]

error 5.222000e+04

reached.threshold 2.117881e-04

steps 1.765000e+03

Intercept.to.1layhid1 8.786959e+00

Pojemnosc.to.1layhid1 8.537569e+00

PredkoscWirowania.to.1layhid1 9.394682e+00

Intercept.to.1layhid2 9.577506e+00

Pojemnosc.to.1layhid2 1.002303e+01

PredkoscWirowania.to.1layhid2 1.044541e+01

Intercept.to.1layhid3 9.015703e+00

Pojemnosc.to.1layhid3 1.077935e+01

PredkoscWirowania.to.1layhid3 8.432529e+00

Intercept.to.2layhid1 9.611780e+00

1layhid1.to.2layhid1 9.284223e+00

1layhid2.to.2layhid1 8.401455e+00

1layhid3.to.2layhid1 9.702894e+00

Intercept.to.2layhid2 7.373112e+00

1layhid1.to.2layhid2 9.657527e+00

1layhid2.to.2layhid2 1.000725e+01

1layhid3.to.2layhid2 1.001183e+01

Intercept.to.Price 1.734392e+02

2layhid1.to.Price 1.741949e+02

2layhid2.to.Price 1.743658e+02

attr(,"class")

[1] "nn"

>

> # 4.5. Wyświetlenie sieci neuronowej

> # Plot the neural network

> plot(net\_price)

>

> # 4.6. Prognozowanie z pomocą̨ sieci neuronowej

> # Test the neural network on some testing data

> # Załóżmy, że testowe dane znajdują się w pliku "testowe\_dane.csv"

> test\_data <- read.csv("testowe\_dane.csv")

>

> scaled\_test\_data <- as.data.frame(scale(test\_data[, 2:3], center = mins, scale = maxs - mins))

> net\_results <- compute(net\_price, scaled\_test\_data)

>

> # Run them through the neural network

> # Let's see what properties net\_price has

> ls(net\_results)

[1] "net.result" "neurons"

>

> # Let's see the results

> print(net\_results$net.result)

[,1]

[1,] 522

[2,] 522

[3,] 522