

Modelowanie i analiza systemów informatycznych

dokumentacja projektu:
System wspomagania decyzji

inż. Bartosz Ociepka

inż. Beniamin Stecuła

25 listopada 2020

Podział pracy

Podział pracy był płynny, lecz z większym naciskiem na:

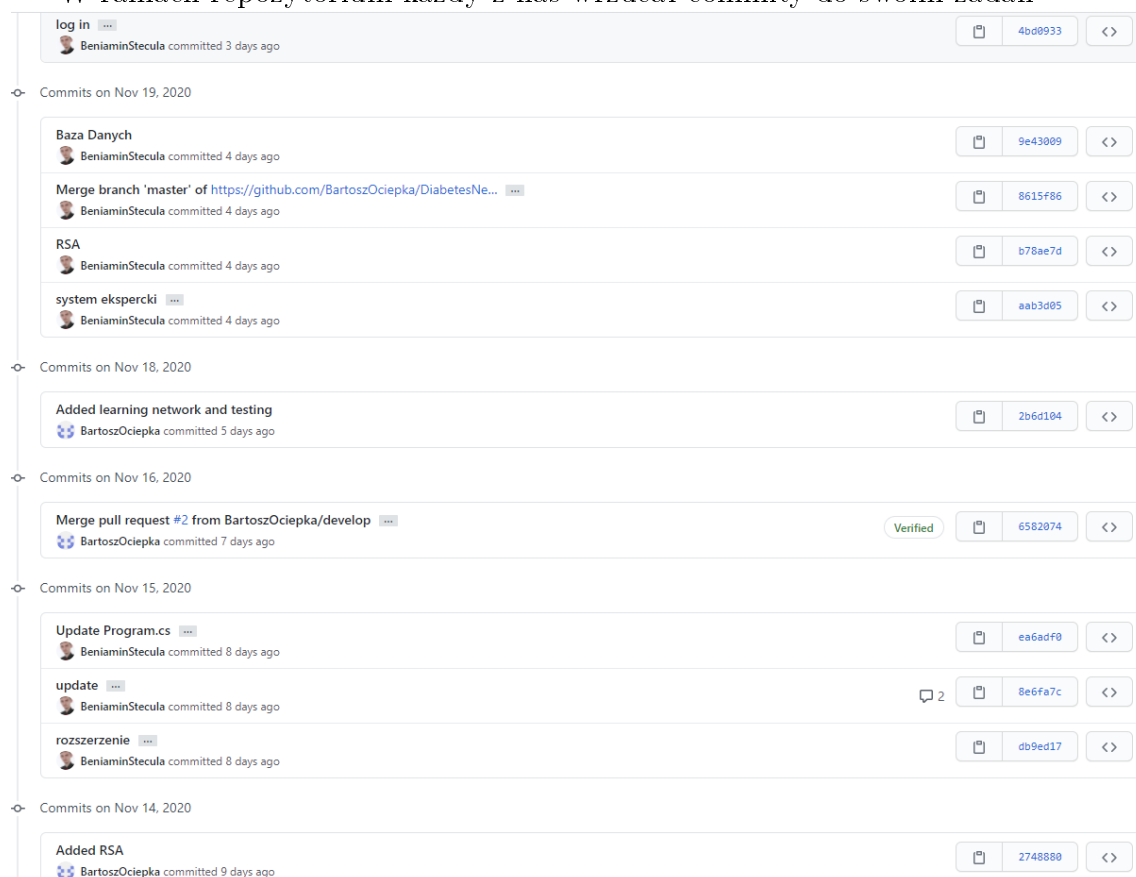
- inż. Bartosz Ociepka – backend, praktyka,
- inż. Beniamin Stecula – frontend, teoria, dokumentacja.

Udokumentowanie pracy

Dokumentowanie pracy odbyło się na kilka sposobów:

- utworzenie niniejszej dokumentacji,
- zarządzanie podziałem i wykonaniem zadań w serwisie Trello,
- przechowywanie kopii poprzednich wersji programu.

W ramach repozytorium każdy z nas wrzucał commity do swoim zadań



Całe repozytorium jest dostępne pod adresem
<https://github.com/BartoszOciepka/DiabetesNeuralNetwork>

Instrukcja obsługi

Niezałogowany użytkownik na start widzi menu, z którego może się zalogować, zarejestrować, sprawdzić próbkę lub wyjść.

```
Hello patient 1
Role: patient

MENU
1. Test data
2. Log out
3. Register
4. Exit program
```

Po zalogowaniu jako doktor do menu dochodzi opcja dodania próbki, trenowania zbioru lub uruchomienia programu testującego, który sprawdza i pokazuje wyniki uczenia dla różnych konfiguracji sieci.

```
Hello doctor 1
Role: doctor

MENU
1. Test data
2. Log out
3. Register
4. Exit program
5. Train dataset
6. Add data to dataset
7. Compare training parameters
```

Wchodząc w każdą opcję użytkownik jest prowadzony za rękę przez cały proces. W razie błędu lub podania niepoprawnych danych wyświetlany jest komunikat błędu.

Instrukcja wdrożenia

Aby wdrożyć projekt należy wykonać poniższą listę kroków:

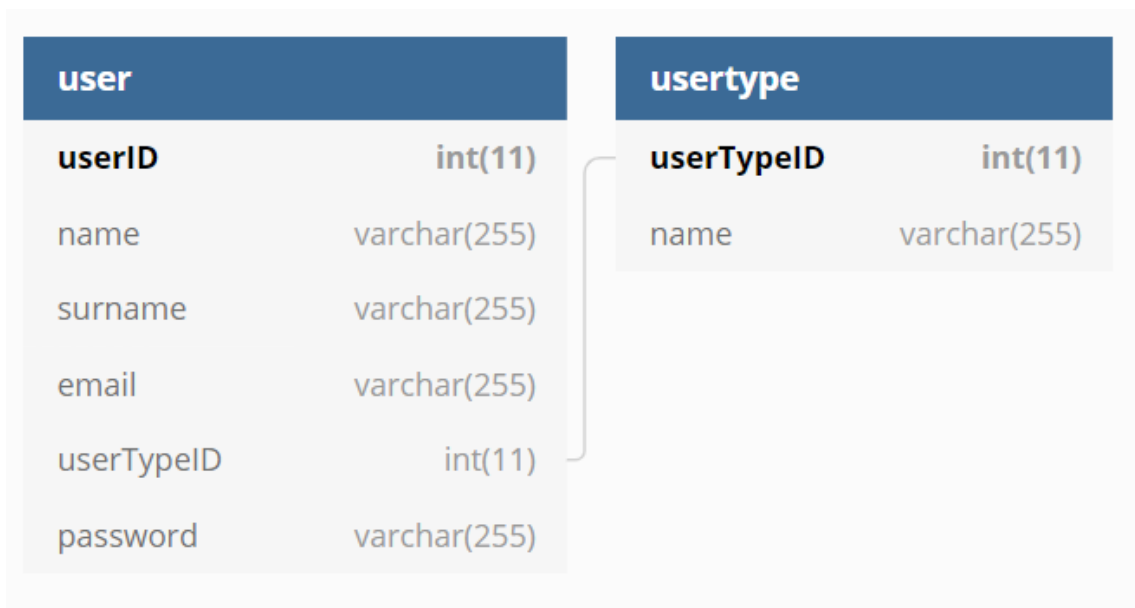
1. zaimportowanie projektu w Visual Studio 2015,
2. import danych do bazy danych MySql (dołączono plik dump.sql zawierający potrzebne tabele),
3. zmiana connectionString w kodzie na odpowiadające używanej bazie, danych (dokonanie zmiany klas gdy używana jest inna baza niż MySql),
4. uruchomienie projektu.

Baza danych

Łączenie z bazą danych ma na stałe ustawione właściwości, które mogą być w razie potrzeby edytowane w klasie *LinqToDbSettings.cs* we fragmencie:

```
1  new ConnectionStringSettings
2  {
3      Name = "DiabetesDatabase",
4      ProviderName = "MySQL",
5      ConnectionString = @"Server=localhost;Database=diabetes;Uid=;Pwd="
6  };
```

Diagram UML bazy danych został przedstawiony na rysunku 1.



Rysunek 1: Diagram klas.

Komponenty systemu

W naszym programie komponenty systemu uporządkowane są w odpowiednich folderach o nazwach pokazujących jakie pliki tam znajdziemy. I tak mamy:

- Core - z głównymi elementami systemu, w naszym wypadku sieć neuronową
- Authorization - pliki do logowania/rejestracji oraz trzymywania ich stanu
- Controllers - tutaj są kontrolery używane jako most między bazą danych a modelami

- Helpers - klasy pomocnicze do walidacji i szyfrowania
- Models - modele używane w programie
- ORM - konfiguracja ORMa, którego używamy do komunikacji z bazą

Użyliśmy wzorca projektowego Singleton, dzięki temu nie musimy tworzyć obiektu gdy chcemy zwalidować dane lub je zaszyfrować. Można też uznać, że w ramach używania biblioteki AForge do sieci neuronowej używamy wzorca Fasada.

Nasza sieć neuronowa jest utworzona przy wykorzystaniu biblioteki AForge (<http://www.aforget.net.com/framework/docs/>). Nie jest to zbyt znana biblioteka, ale do jej wykorzystania przekonała nas bardzo dokładna dokumentacja. Biblioteka udostępnia nam obiekt *ActivationNetwork*, który w konstruktorze bierze funkcję aktywacyjną, ilość wejść sieci oraz tablicę neuronów w poszczególnych warstwach. W naszym wypadku za funkcję aktywacyjną wzięliśmy funkcję sigmoidalną z alfą równą 1.8 (taką wartość uznaliśmy za najlepszą w ramach testowania). Funkcja ta ma postać:

$$f(x) = \frac{1}{1 + e^{-\alpha * x}} \quad (1)$$

Gdy już mamy utworzony obiekt sieci musimy ją nauczyć. Służy do tego klasa *BackPropagationLearning* (jak i inne, ale my będziemy uczyć sieć używając propagacji wstecznej).

Algorytm szyfrowania

W naszym programie do rejestracji i logowania użyliśmy stworzonej przez siebie klasy *AuthorizationManager*, która zawiera metody *LogIn()*, *Register()*, *LogOut()*. Szyfrowanie zaimplementowane zostało w klasie *RSAHelper*, w której znajduje się cała logika szyfrowania opierająca się na klasie *RSACryptoServiceProvider*. Do szyfrowania użyty został algorytm RSA (definicja oraz dwa wywołania):

```

1      using (var rsa = new RSACryptoServiceProvider(2048))
2      ...
3      User user = UserController.FindUserByEmailAndPassword(email,
        RSAHelper.encrypt(password));
4      ...
5      User user = new User(name, surname, email, RSAHelper.encrypt(
        password), userTypeID);

```

Algorytm Rivesta-Shamira-Adlemana (RSA) to jeden z pierwszych i najpopularniejszych asymetrycznych algorytmów kryptograficznych o kluczu publicznym. Może być stosowany i do szyfrowania, i cyfrowego podpisywania plików.

Polega on na liczeniu funkcji Eulera dla dużych liczb pierwszych, a jego bezpieczeństwo opiera się na trudności faktoryzacji dużych liczb złożonych.

Każdy z rozmówców posiada parę kluczy: prywatny i publiczny. Pierwszy z nich służy do deszyfrowania wiadomości przychodzącej, a drugi do szyfrowania wychodzącej. Aby nawiązać komunikację rozmówcy muszą wymienić się swoimi kluczami publicznymi. Klucze prywatne nigdy nie są ujawniane.

System ekspercki

W dziale sztucznej inteligencji systemem eksperckim (Expert System – ES) nazywa się system komputerowy, który emuluje podejmowanie decyzji przez ludzkiego specjalistę z danej dziedziny. Odzwierciedla procesy myślowe na zasadzie działania ludzkiego mózgu.

Do rozwiązań wykorzystujących ES zaliczamy:

- systemy agentowe,
- agenty programowe,
- eksplorację danych,
- wspomaganie twórczego myślenia,
- ontologie systematyzujące wiedzę.

ES powinien zawierać trzy charakterystyczne, podstawowe cechy:

- bazy wiedzy – schematycznie zapisane informacje uzyskane od ludzkich ekspertów w danej dziedzinie,
- procedury wnioskujące – odzwierciedlające wnioskowanie symboliczne, które jest charakterystyczne dla funkcjonowania ludzkiego mózgu,
- zdolność do poszerzania wiedzy – umożliwienie ciągłego rozwoju systemu w przypadku stale napływających nowych informacji koniecznych do uwzględnienia aby otrzymywać stale aktualne wyniki zgodne z nowym stanem wiedzy w dziedzinie.

Struktura ES składa się z sześciu podstawowych elementów:

- baza wiedzy – składająca się ze zbioru reguł,
- baza danych – zawierająca dane obiektów, wartości, przypadki i hipotezy,
- procedury wnioskowania – jest to maszyna wnioskująca,
- procedury objaśniania – tłumaczą strategię wnioskowania,
- procedury sterowania dialogiem – funkcje wejścia/wyjścia sterujące programem i wyznaczające zadania, które ma wykonać
- procedury zarządzania wiedzą – pozwalają na modyfikację oraz rozszerzanie, pozyskiwanie wiedzy.

Implementując kod programu i tworząc samemu kod ekspertowy doszliśmy do pewnych wniosków na temat zalet i wad systemów eksperckich:

Zalety to na pewno szybkość przetwarzania danych. Człowiekowi zajęło by to nieporównywalnie więcej czasu niż komputerowi. Do tego się potrafi zauważyć powiązanie, które człowiek pominie albo nie zauważy.

Wady to przede wszystkim brak zdrowego rozsądku. Dla komputera liczby to liczby więc błędne dane czy dane na logikę wskazujące na pewne rozwiązanie nie zwrócą jego uwagi. Dochodzi do tego możliwość przeuczenia sieci, błędne dopasowanie parametrów sieci lub zbyt mały zbiór danych i mamy rozwiązanie, które jest bardzo potężne, ale wymaga uwagi i wiedzy aby poprawnie zaimplementować.

Przykładowy kod z aplikacji z testami

RSAHelper.cs

```
1 using System;
2 using System.Security.Cryptography;
3 using System.Text;
4
5 namespace DiabetesNeuralNetwork
6 {
7     public class RSAHelper
8     {
9         public static string containerName = "defaultContainerName";
10        public static string encrypt(string message)
11        {
12            var bytesToEncrypt = Encoding.UTF8.GetBytes(message);
13
14            using (var rsa = new RSACryptoServiceProvider(2048))
15            {
16                try
17                {
18                    rsa.FromXmlString(RSAHelper.getPublicKeyFromContainer());
19                    var encryptedData = rsa.Encrypt(bytesToEncrypt, true);
20                    var base64Encrypted = Convert.ToBase64String(encryptedData);
21                    return base64Encrypted;
22                }
23                finally
24                {
25                    rsa.PersistKeyInCsp = false;
26                }
27            }
28        }
29
30        public static string decrypt(string message)
31        {
32            var bytesToDecrypt = Encoding.UTF8.GetBytes(message);
33
34            using (var rsa = new RSACryptoServiceProvider(2048))
35            {
36                try
37                {
```

```

38         rsa.FromXmlString(RSAHelper.getPrivateKeyFromContainer());
39
40         var resultBytes = Convert.FromBase64String(message);
41         var decryptedBytes = rsa.Decrypt(resultBytes, true);
42         var decryptedData = Encoding.UTF8.GetString(decryptedBytes);
43         return decryptedData.ToString();
44     }
45     finally
46     {
47         rsa.PersistKeyInCsp = false;
48     }
49 }
50 }
51
52 // Run this method before any encryption.
53 // This method adds a new key container and saves it on the machine.
54 // That way the public and private keys are the same each run.
55 public static void addKeysToContainer()
56 {
57     var parameters = new CspParameters
58     {
59         KeyContainerName = RSAHelper.containerName
60     };
61
62     var rsa = new RSACryptoServiceProvider(parameters);
63
64     Console.WriteLine("Added new key provider to default container");
65 }
66
67 public static string getPublicKeyFromContainer()
68 {
69     var parameters = new CspParameters
70     {
71         KeyContainerName = containerName
72     };
73
74     var rsa = new RSACryptoServiceProvider(parameters);
75     var publicKey = rsa.ExportParameters(false); //Generowanie klucza
76     //publiczny
77     string publicKeyString = RSAHelper.GetKeyString(publicKey);
78
79     // Display the key information to the console.
80     Console.WriteLine($"Public key retrieved from container");
81
82     return publicKeyString;
83 }
84
85 public static string getPrivateKeyFromContainer()
86 {
87     var parameters = new CspParameters
88     {
89         KeyContainerName = containerName
90     };
91
92     var rsa = new RSACryptoServiceProvider(parameters);

```



```

91         var privateKey = rsa.ExportParameters(true); //Generowanie klucza
           publiczny
92         string privateKeyString = RSAHelper.GetKeyString(privateKey);
93
94         Console.WriteLine("Private key retrieved from container");
95
96         return privateKeyString;
97     }
98     public static string GetKeyString(RSAParameters key)
99     {
100         var stringWriter = new System.IO.StringWriter();
101         var xmlSerializer = new System.Xml.Serialization.XmlSerializer(
           typeof(RSAParameters));
102         xmlSerializer.Serialize(stringWriter, key);
103         return stringWriter.ToString();
104     }
105 }
106 }

```

UserType.cs

```

1 using System.Collections.Generic;
2 using LinqToDB.Mapping;
3
4 namespace DiabetesNeuralNetwork.Models
5 {
6     [Table(Name = "UserType")]
7     public class UserType
8     {
9         [PrimaryKey, Identity]
10        public int userTypeID;
11        [Column(Name = "name"), NotNull]
12        public string name { get; set; }
13        List<User> users { get; set; }
14
15        public override string ToString()
16        {
17            return this.userTypeID + " " + this.name;
18        }
19    }
20 }

```

User.cs

```

1 using LinqToDB.Mapping;
2
3 namespace DiabetesNeuralNetwork
4 {
5     [Table(Name = "User")]
6     public class User
7     {
8         [PrimaryKey, Identity]
9         public int userID;
10        [Column(Name = "name"), NotNull]
11        public string name;
12        [Column(Name = "surname"), NotNull]

```

```

13     public string surname;
14     [Column(Name = "email"), NotNull]
15     public string email;
16     [Column(Name = "userID"), NotNull]
17     public int userID;
18     [Column(Name = "password"), NotNull]
19     public string password;
20
21     public User()
22     {
23
24     }
25     public User(string name, string surname, string email, string
        password, int userID)
26     {
27         this.name = name;
28         this.surname = surname;
29         this.email = email;
30         this.password = password;
31         this.userID = userID;
32     }
33
34     public override string ToString()
35     {
36         return this.name + " " + this.surname + " " + this.email;
37     }
38 }
39 }

```

AuthorizationManager.cs

```

1 using DiabetesNeuralNetwork.Controllers;
2 using System;
3
4 namespace DiabetesNeuralNetwork
5 {
6     class AuthorizationManager
7     {
8         public static void LogIn()
9         {
10             Console.WriteLine("Email:");
11             string email = Console.ReadLine();
12             Console.WriteLine("Password:");
13
14             var password = readPassword();
15
16             User user = UserController.FindUserByEmailAndPassword(email,
                RSAHelper.encrypt(password));
17             if (user == null) {
18                 Console.WriteLine("Incorrect credentials");
19             }
20             else
21             {
22                 LogInUser(user);
23                 Console.WriteLine("Hi " + user.name + "! You were successfully

```

```

        logged in.");
    }
}

public static void LogInUser(User user)
{
    LoginStatus.IsLoggedIn = true;
    LoginStatus.LoggedInUser = user;
}

public static void Register()
{
    try
    {
        Console.WriteLine("What user type to register:");
        UserTypeController.GetAll().ForEach(i => Console.WriteLine("{0}\n"
            t", i));
        int userID = Int32.Parse(Console.ReadLine());
        Console.WriteLine("Name:");
        string name = Console.ReadLine();
        Console.WriteLine("Surname:");
        string surname = Console.ReadLine();
        Console.WriteLine("Email:");
        string email = Console.ReadLine();
        Console.WriteLine("Password:");
        string password = readPassword();

        Validator.ValidateUser(name, surname, email, password,
            userID);
        User user = new User(name, surname, email, RSAHelper.encrypt(
            password), userID);
        UserController.Insert(user);
        LogInUser(user);
        Console.WriteLine("Successfully registered new user");
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

private static string readPassword()
{
    var password = string.Empty;
    ConsoleKey key;
    do
    {
        var keyInfo = Console.ReadKey(intercept: true);
        key = keyInfo.Key;

        if (key == ConsoleKey.Backspace && password.Length > 0)
        {
            Console.Write("\b \b");
            password = password.Remove(password.Length - 1);
        }
    }
}

```

```

75     }
76     else if (!char.IsControl(keyInfo.KeyChar))
77     {
78         Console.Write("*");
79         password += keyInfo.KeyChar;
80     }
81     } while (key != ConsoleKey.Enter);
82     Console.WriteLine();
83
84     return password;
85 }
86
87     public static void LogOut()
88 {
89     LoginStatus.IsLoggedIn = false;
90     LoginStatus.LoggedInUser = null;
91     Console.WriteLine("Successfully logged out");
92 }
93
94
95
96 }
97 }

```

Przykładowe testy:

```

1     [TestMethod]
2     public void TestRSACorrectlyEncryptsAndDecrypts()
3     {
4         string testString = "Ala ma kota";
5
6         string encryptedText = RSAHelper.encrypt(testString);
7
8         Assert.AreEqual(testString, RSAHelper.decrypt(encryptedText),
9             false, "Original text and same text after encrypt and decrypt
10             are not the same");
11     }
12     [TestMethod]
13     public void TestRSAEncryptsTheSameEachTime()
14     {
15         string testString = "Ala ma kota";
16
17         string encryptedText = RSAHelper.encrypt(testString);
18         string anotherEncryptedText = RSAHelper.encrypt(testString);
19
20         Assert.AreEqual(RSAHelper.decrypt(encryptedText), RSAHelper.
21             decrypt(anotherEncryptedText), false, "Not the same results
22             when encrypting and decrypting the same string");
23     }
24     [TestMethod]
25     public void testUserValidationCorrect()
26     {
27         string name = "test";
28         string surname = "test";
29         string password = "test";

```

```

26
27     Validator.ValidateName(name);
28     Validator.ValidateSurname(surname);
29     Validator.ValidatePassword(password);
30 }
31
32 [TestMethod]
33 [ExpectedException(typeof(Exception), "Empty name should not pass
    validation")]
34 public void testUserValidationIncorrect()
35 {
36     Validator.ValidateName("");
37 }

```

Wyglad programu

The image displays four screenshots of a program's terminal output, arranged in a 2x2 grid. The top-left screenshot shows the initial greeting 'Hello stranger' followed by a menu with four options: '1. Test data', '2. Log in', '3. Register', and '4. Exit program'. The top-right screenshot shows the program's response to the 'Log in' option, displaying 'Hello doctor 1' and 'Role: doctor', followed by a menu with five options: '1. Test data', '2. Log out', '3. Register', '4. Exit program', and '5. Train dataset'. The bottom-left screenshot shows the program's response to the 'Train dataset' option, displaying 'Error: 13,7619553827776'. The bottom-right screenshot shows the program's response to the 'Log out' option, displaying 'Hello stranger' followed by a menu with four options: '1. Test data', '2. Log in', '3. Register', and '4. Exit program'.

```

Hello stranger

MENU
1. Test data
2. Log in
3. Register
4. Exit program

Hello doctor 1
Role: doctor

MENU
1. Test data
2. Log out
3. Register
4. Exit program
5. Train dataset
6. Add data to dataset
7. Compare training parameters
5
Error: 13,7619553827776

Press enter...

Hello stranger

MENU
1. Test data
2. Log in
3. Register
4. Exit program
2
Email:
doctor@doctor
Password:
*****
Public key retrieved from container
Private key retrieved from container
Private key retrieved from container
Hi doctor! You were successfully logged in.

Press enter...

Hello doctor 1
Role: doctor

MENU
1. Test data
2. Log out
3. Register
4. Exit program
5. Train dataset
6. Add data to dataset
7. Compare training parameters
5
Error: 13,7619553827776

Press enter...

```

MENU	
1. Test data	
2. Log out	
3. Register	
4. Exit program	
5. Train dataset	
6. Add data to dataset	
7. Compare training parameters	
6	Testing: 8 1 Error too much
Pregnancies:	Testing: 9 1 Error too much
1	Testing: 10 1 Error too much
Glucose:	Testing: 11 1 Error too much
123	Testing: 12 1 Error too much
Blood pressure:	Testing: 13 1 Error too much
123	Testing: 14 1 Error: 44,2429932380331
Skin thickness:	Testing: 15 1 Error: 47,3197320024758
3	Testing: 16 1 Error: 42,6153842096591
Insulin:	
23	
BMI:	Current number of layers: 3
0,65	Testing: 1 1 1 Error too much
Diabetes pedigree function:	
0,5	Testing: 2 1 1 Error too much
Age:	Testing: 2 2 1 Error too much
12	Testing: 3 1 1 Error too much
	Testing: 3 2 1 Error too much
	Testing: 3 3 1 Error too much
	Testing: 4 1 1 Error too much
	Testing: 4 2 1 Error too much
	Testing: 4 3 1 Error too much
	Testing: 4 4 1 Error too much
	Testing: 5 1 1 Error too much
	Testing: 5 2 1 Error too much
	Testing: 5 3 1 Error too much
	Testing: 5 4 1