



**KARADENİZ
TEKNİK ÜNİVERSİTESİ**
KARADENİZ TECHNICAL UNIVERSITY
1955

**T.C.
KARADENİZ TEKNİK ÜNİVERSİTESİ
Mühendislik Fakültesi**

Bilgisayar Mühendisliği Bölümü

Yapay Sinir Ağları Projesi

Elif Vişne 402557

Prof. Dr. MURAT EKİNCİ

1. Proje Özeti

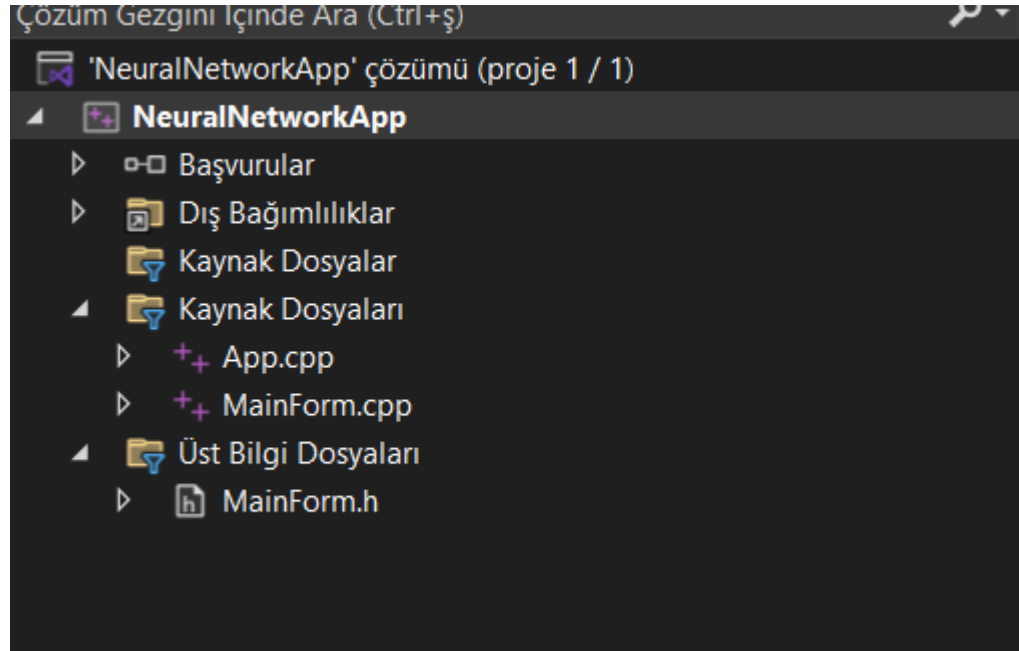
Bu proje, wxWidgets kullanarak geliştirdiğim bir yapay sinir ağı (YSA) uygulamasıdır ve farklı sınıflar arasında ayırım yapabilen bir model oluşturmayı hedefliyor. Visual Studio'da Windows Form uygulamasını doğrudan açamadığım için, wxWidgets kütüphanesini tercih ederek hem kullanıcı dostu hem de görsel bir arayüz geliştirme fırsatı buldum.

Uygulamada, kullanıcılar bir koordinat düzleminde noktalar seçip bu noktaları dört farklı sınıfa (Class 1, Class 2, Class 3, Class 4) atayabiliyor. Her sınıf farklı bir renk ile temsil ediliyor, bu da noktaların ayırt edilmesini kolaylaştırıyor. Eğitim verilerini oluşturduktan sonra, kullanıcı epoch ve learning rate değerlerini belirleyerek YSA modelini eğitebiliyor. Model, sigmoid aktivasyon fonksiyonu kullanarak ve ağırlık güncellemelerini manuel olarak yaparak çalışıyor.

Eğitim tamamlandıktan sonra, model test verilerini kullanarak bir noktanın hangi sınıfa ait olduğunu tahmin edebiliyor. Bunun yanı sıra, sınıflar arasında ayırım yapan çizgiler koordinat düzlemine görsel olarak ekleniyor.

2. Dosya Yapısı

Proje, NeuralNetworkApp adıyla düzenlenmiş ve içinde App.cpp, MainForm.cpp ve MainForm.h dosyaları bulunuyor. Bu dosyalar, uygulamanın işleyişi ve kullanıcı arayüzünü oluşturuyor. Ayrıca, proje dış bağımlılıklar ve başvurularla ilgili gerekli dosyaları da içeriyor. Bu yapı, projeyi daha düzenli hale getiriyor ve işlevlerin birbirinden ayrılmasını sağlıyor.



3. App.cpp dosyası

App.cpp dosyasında, wxWidgets uygulamasını başlatan MyApp sınıfı yer alır. OnInit fonksiyonunda, MainForm sınıfından bir nesne oluşturulup ekranda gösterilir. Bu dosya, uygulamanın başlangıcını kontrol eder.

```
MainForm.cpp  MainForm.h  App.cpp  + x
NeuralNetworkApp
1  #include "wx/wx.h"
2  #include "MainForm.h"
3
4  class MyApp : public wxApp {
5  public:
6      virtual bool OnInit();
7  };
8
9  wxIMPLEMENT_APP(MyApp);
10
11 bool MyApp::OnInit() {
12     MainForm* mainForm = new MainForm("Yapay Sinir Ağı");
13     mainForm->Show(true);
14     return true;
15 }
```

4. MainForm.h

MainForm.h dosyası, ana penceremizi ve kullanıcı etkileşimlerini yönetiyor. Burada, sınıfları seçmek için kutular, eğitim parametrelerini seçmek için seçenekler ve verileri saklamak için listeler var. Ayrıca, sinir ağını eğitmek, noktaları çizmek ve sınıfları ayıran çizgiyi göstermek gibi işlemleri yapan fonksiyonlar da bulunuyor.

```
1  #pragma once
2
3  #include <wx/wx.h>
4  #include <vector>
5  #include <utility>
6
7  class MainForm : public wxFrame
8  {
9  public:
10     MainForm(const wxString& title);
11     std::pair<int, double> max_selector(const std::vector<double>& netOutputs);
12
13 private:
14     wxCheckBox* class1CheckBox;
15     wxCheckBox* class2CheckBox;
16     wxCheckBox* class3CheckBox;
17     wxCheckBox* class4CheckBox;
18     wxButton* normalizeButton;
19     wxChoice* epochChoice;
20     wxChoice* learningRateChoice;
21
22     std::vector<std::pair<double, double>> class1Points;
23     std::vector<std::pair<double, double>> class2Points;
24     std::vector<std::pair<double, double>> class3Points;
25     std::vector<std::pair<double, double>> class4Points;
26
27     std::vector<std::vector<int>> training_outputs;
28
29     std::vector<std::vector<double>> weights;
30 }
```

```
28
29     std::vector<std::vector<double>> weights;
30
31     void OnPaint(wxPaintEvent& event);
32     void OnMouseClicked(wxMouseEvent& event);
33     void OnClass1Checked(wxCommandEvent& event);
34     void OnClass2Checked(wxCommandEvent& event);
35     void OnClass3Checked(wxCommandEvent& event);
36     void OnClass4Checked(wxCommandEvent& event);
37     void OnNormalize(wxCommandEvent& event);
38     void TrainNeuralNetwork(int epochLimit, double learningRate);
39     void DrawPoints(wxDC& dc);
40     void DrawSeparatingLine(wxDC& dc, int classIndex);
41     void test(double x, double y);
42
43     double sigmoid(double x);
44     double activation(double x);
45
46     bool isClass1Active;
47     bool isClass2Active;
48     bool isClass3Active;
49     bool isClass4Active;
50
51     wxDECLARE_EVENT_TABLE();
52 };
53
54
```

5. MainForm.cpp

Bu C++ programında wxWidgets kullanılarak bir sinir ağı uygulaması oluşturulmuş. Programda temel olarak kullanıcı arayüzü (GUI) ele alınmış. MainForm sınıfı, kullanıcıdan gelen verileri alıp işleyerek sonuçları ekranda gösteriyor. Bu sayede sinir ağı eğitimi yapılabilir. Programın bazı kısa fonksiyonları şu şekilde çalışıyor:

OnPaint: Ekranda eksenler ve tıklanan sınıf noktalarını çiziyor. Kullanıcı, farklı sınıfları temsil eden noktalara tıkladıkça, bu noktalar ekranda farklı renklerde görünür.

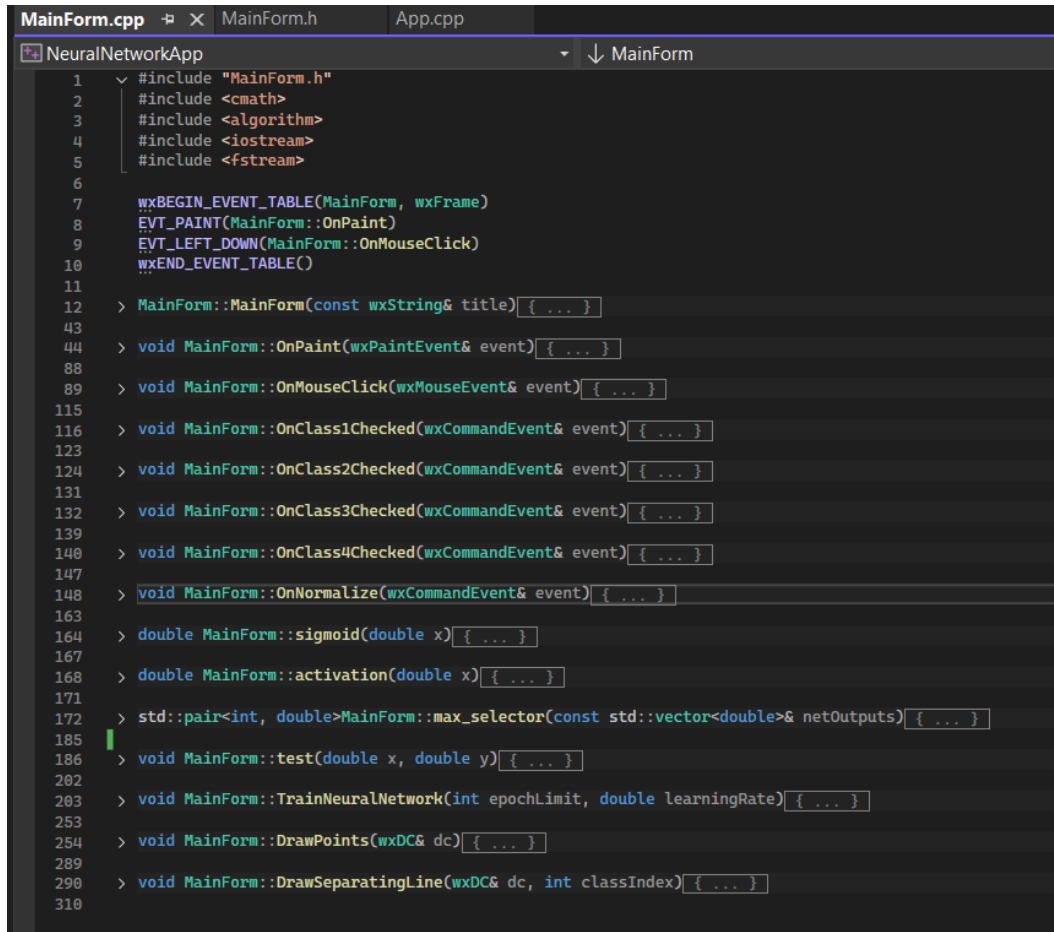
OnMouseClicked: Kullanıcı tıkladığında, tıklanan noktanın hangi sınıfa ait olduğunu belirleyip bu veriyi kaydediyor. Her sınıfın farklı bir rengi var ve bu renkler GUT'de yansıtılıyor.

OnClassChecked: Bu fonksiyon, kullanıcı bir sınıfı işaretlediğinde aktif sınıfı belirliyor. Hangi sınıf aktifse, kullanıcı o sınıfa ait noktaları ekleyebiliyor.

OnNormalize: Sinir ağına ait ağırlıkları, kullanıcı tarafından belirlenen "epoch" ve "learning rate" değerlerine göre güncellemeye başlıyor. Bu işlem, sinir ağı eğitimi için çok önemli.

TrainNeuralNetwork: Sinir ağı modelini, kullanıcı tarafından seçilen parametrelere göre eğiten bir fonksiyon. Bu fonksiyon, eğitim için gerekli hesaplamaları yaparak ağırlıkları güncelliyor.

sigmoid: Bu fonksiyon, sinir ağında kullanılan sigmoid aktivasyon fonksiyonunu hesaplıyor. Bu sayede, modelin doğruluğu ve çıktısı belirleniyor.



```
1  #include "MainForm.h"
2  #include <cmath>
3  #include <algorithm>
4  #include <iostream>
5  #include <fstream>
6
7  wxBEGIN_EVENT_TABLE(MainForm, wxFrame)
8  EVT_PAINT(MainForm::OnPaint)
9  EVT_LEFT_DOWN(MainForm::OnMouseClicked)
10 wxEND_EVENT_TABLE()
11
12 > MainForm::MainForm(const wxString& title) { ... }
13
14 > void MainForm::OnPaint(wxPaintEvent& event) { ... }
15
16 > void MainForm::OnMouseClicked(wxMouseEvent& event) { ... }
17
18 > void MainForm::OnClass1Checked(wxCommandEvent& event) { ... }
19
20 > void MainForm::OnClass2Checked(wxCommandEvent& event) { ... }
21
22 > void MainForm::OnClass3Checked(wxCommandEvent& event) { ... }
23
24 > void MainForm::OnClass4Checked(wxCommandEvent& event) { ... }
25
26 > void MainForm::OnNormalize(wxCommandEvent& event) { ... }
27
28 > double MainForm::sigmoid(double x) { ... }
29
30 > double MainForm::activation(double x) { ... }
31
32 > std::pair<int, double> MainForm::max_selector(const std::vector<double>& netOutputs) { ... }
33
34 > void MainForm::test(double x, double y) { ... }
35
36 > void MainForm::TrainNeuralNetwork(int epochLimit, double learningRate) { ... }
37
38 > void MainForm::DrawPoints(wxDC& dc) { ... }
39
40 > void MainForm::DrawSeparatingLine(wxDC& dc, int classIndex) { ... }
```

5.1. OnPaint

```
void MainForm::OnPaint(wxPaintEvent& event) {
    wxPaintDC dc(this);
    dc.SetPen(wxPen(wxColour(0, 0, 0), 2));

    dc.DrawLine(400, 0, 400, 600); // Y eksenini
    dc.DrawLine(0, 300, 800, 300); // X eksenini

    DrawPoints(dc);

    if (!weights.empty()) {
        wxPen pens[] = { wxPen(wxColour(0, 0, 255), 2), wxPen(wxColour(255, 0, 0), 2),
                        wxPen(wxColour(0, 255, 0), 2), wxPen(wxColour(255, 165, 0), 2) };

        if (class1Points.size() > 0 && class2Points.size() > 0 && class3Points.size() == 0 && class4Points.size() == 0) {
            // 2 sınıf
            for (int i = 0; i < 2; ++i) {
                dc.SetPen(pens[i]);
                DrawSeparatingLine(dc, i);
            }
        }
        else {
            // 4 sınıf
            for (size_t i = 0; i < weights.size(); ++i) {
                if (weights[i][1] == 0 && weights[i][2] == 0) continue;

                double x1 = -400 / 50.0;
                double y1 = (-weights[i][0] - weights[i][1] * x1) / weights[i][2];

                double x2 = 400 / 50.0;
                double y2 = (-weights[i][0] - weights[i][1] * x2) / weights[i][2];

                int screenX1 = 400 + x1 * 50;
                int screenY1 = 300 - y1 * 50;
                int screenX2 = 400 + x2 * 50;
                int screenY2 = 300 - y2 * 50;

                dc.SetPen(pens[i]);
                dc.DrawLine(screenX1, screenY1, screenX2, screenY2);
            }
        }
    }
    test(50, 50); //manuel verilen test için örnek
}
```

5.2. OnMouseClicked

```
void MainForm::OnMouseClicked(wxMouseEvent& event) {
    wxPoint mousePos = event.GetPosition();
    double x = (mousePos.x - 400) / 50.0;
    double y = (300 - mousePos.y) / 50.0;

    if (isClass1Active) {
        class1Points.emplace_back(x, y);
        training_outputs.push_back({ 1, 0, 0, 0 });
    }
    else if (isClass2Active) {
        class2Points.emplace_back(x, y);
        training_outputs.push_back({ 0, 1, 0, 0 });
    }
    else if (isClass3Active) {
        class3Points.emplace_back(x, y);
        training_outputs.push_back({ 0, 0, 1, 0 });
    }
    else if (isClass4Active) {
        class4Points.emplace_back(x, y);
        training_outputs.push_back({ 0, 0, 0, 1 });
    }

    Refresh();
}
```

5.3. OnClass1Checked

```
void MainForm::OnClass1Checked(wxCommandEvent& event) {  
    isClass1Active = class1CheckBox->IsChecked();  
    isClass2Active = isClass3Active = isClass4Active = false;  
    class2CheckBox->SetValue(false);  
    class3CheckBox->SetValue(false);  
    class4CheckBox->SetValue(false);  
}
```

Class2, Class3 ve Class4 de buna benzer şekildedir.

5.4. OnNormalize

```
void MainForm::OnNormalize(wxCommandEvent& event) {  
    if (epochChoice->GetSelection() == wxNOT_FOUND || learningRateChoice->GetSelection() == wxNOT_FOUND) {  
        wxMessageBox("Please select both Epoch and Learning Rate before training.", "Error", wxICON_ERROR);  
        return;  
    }  
  
    wxString epochStr = epochChoice->GetStringSelection();  
    wxString learningRateStr = learningRateChoice->GetStringSelection();  
  
    int epochLimit = wxAtoi(epochStr);  
    double learningRate = wxAtof(learningRateStr);  
  
    TrainNeuralNetwork(epochLimit, learningRate);  
    Refresh();  
}
```

5.5. Sigmoid ve activation

```
double MainForm::sigmoid(double x) {  
    return 1.0 / (1.0 + exp(-x));  
}  
  
double MainForm::activation(double x) {  
    return (x > 0 ? 1 : 0);  
}
```

5.6. Test

```
std::pair<int, double> MainForm::max_selector(const std::vector<double>& netOutputs) {  
    int maxIndex = 0;  
    double maxValue = netOutputs[0];  
  
    for (int i = 1; i < netOutputs.size(); ++i) {  
        if (netOutputs[i] > maxValue) {  
            maxValue = netOutputs[i];  
            maxIndex = i;  
        }  
    }  
  
    return { maxIndex, maxValue };  
}  
  
void MainForm::test(double x, double y) {  
    std::vector<double> netOutputs(4, 0);  
    for (int j = 0; j < 4; ++j) {  
        netOutputs[j] = sigmoid(weights[j][0] + weights[j][1] * x + weights[j][2] * y);  
    }  
  
    int idx;  
    double value;  
  
    std::pair<int, double> result = max_selector(netOutputs);  
  
    wxString message;  
    message.Printf("The class to which the point belongs: %d", result.first+1); //Noktaya ait sınıf  
    wxMessageBox(message, "Sonuç", wxOK | wxICON_INFORMATION);  
}
```

5.7. TrainNeuralNetwork

```
void MainForm::TrainNeuralNetwork(int epochLimit, double learningRate) {
    weights.clear();
    weights.resize(4, std::vector<double>(3, 0)); // [bias, weight_x, weight_y]

    for (int epoch = 0; epoch < epochLimit; ++epoch) {
        bool allClassified = true;

        for (size_t i = 0; i < training_outputs.size(); ++i) {
            // Noktanın koordinatlarını ve hedef çıktıları al
            double x = 0, y = 0;
            if (i < class1Points.size()) {
                x = class1Points[i].first;
                y = class1Points[i].second;
            }
            else if (i < class1Points.size() + class2Points.size()) {
                x = class2Points[i - class1Points.size()].first;
                y = class2Points[i - class1Points.size()].second;
            }
            else if (i < class1Points.size() + class2Points.size() + class3Points.size()) {
                x = class3Points[i - class1Points.size() - class2Points.size()].first;
                y = class3Points[i - class1Points.size() - class2Points.size()].second;
            }
            else {
                x = class4Points[i - class1Points.size() - class2Points.size() - class3Points.size()].first;
                y = class4Points[i - class1Points.size() - class2Points.size() - class3Points.size()].second;
            }

            // Çıktıları hesapla
            std::vector<double> netOutputs(4, 0);
            for (int j = 0; j < 4; ++j) {
                netOutputs[j] = sigmoid(weights[j][0] + weights[j][1] * x + weights[j][2] * y);
            }

            // Hata ve ağırlık güncelleme
            for (int j = 0; j < 4; ++j) {
                double error = training_outputs[i][j] - netOutputs[j];
                if (std::abs(error) > 0.01) {
                    allClassified = false;
                    weights[j][0] += learningRate * error; // Bias güncelleme
                    weights[j][1] += learningRate * error * x; // x ağırlığı güncelleme
                    weights[j][2] += learningRate * error * y; // y ağırlığı güncelleme
                }
            }
        }

        if (allClassified) break;
    }
}
```

5.8. DrawPoints

```
void MainForm::DrawPoints(wxDC& dc) {  
    wxBrush brushes[] = {  
        *wxBLUE_BRUSH,  
        *wxRED_BRUSH,  
        *wxGREEN_BRUSH,  
        wxBrush(wxColour(255, 165, 0))  
    };  
  
    for (const auto& point : class1Points) {  
        double x = point.first;  
        double y = point.second;  
        dc.SetBrush(brushes[0]); // Class 1 renk  
        dc.DrawCircle(400 + x * 50, 300 - y * 50, 5);  
    }  
  
    for (const auto& point : class2Points) {  
        double x = point.first;  
        double y = point.second;  
        dc.SetBrush(brushes[1]); // Class 2 renk  
        dc.DrawCircle(400 + x * 50, 300 - y * 50, 5);  
    }  
  
    for (const auto& point : class3Points) {  
        double x = point.first;  
        double y = point.second;  
        dc.SetBrush(brushes[2]); // Class 3 renk  
        dc.DrawCircle(400 + x * 50, 300 - y * 50, 5);  
    }  
  
    for (const auto& point : class4Points) {  
        double x = point.first;  
        double y = point.second;  
        dc.SetBrush(brushes[3]); // Class 4 renk  
        dc.DrawCircle(400 + x * 50, 300 - y * 50, 5);  
    }  
}
```

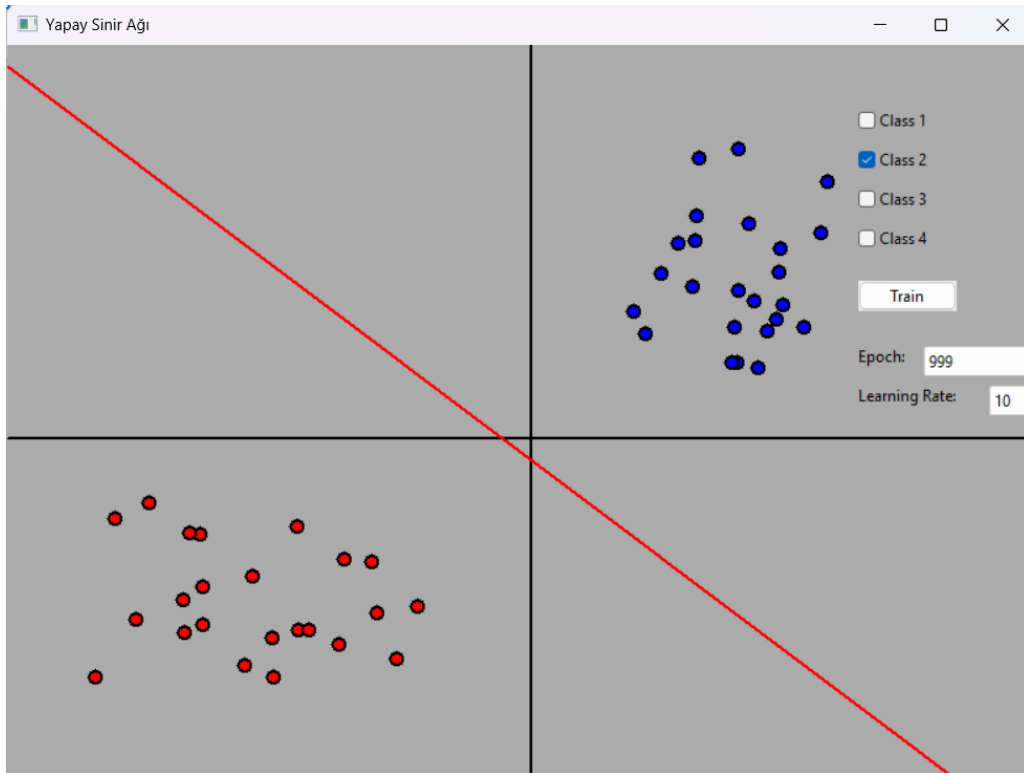
5.9. DrawSeparatingLine

```
void MainForm::DrawSeparatingLine(wxDC& dc, int classIndex) {  
    if (classIndex >= 0 && classIndex < weights.size()) {  
        double x1 = -400 / 50.0;  
        double y1 = (-weights[classIndex][0] - weights[classIndex][1] * x1) / weights[classIndex][2];  
  
        double x2 = 400 / 50.0;  
        double y2 = (-weights[classIndex][0] - weights[classIndex][1] * x2) / weights[classIndex][2];  
  
        int screenX1 = 400 + x1 * 50;  
        int screenY1 = 300 - y1 * 50;  
        int screenX2 = 400 + x2 * 50;  
        int screenY2 = 300 - y2 * 50;  
  
        wxPen pens[] = { wxPen(wxColour(0, 0, 255), 2), wxPen(wxColour(255, 0, 0), 2),  
                        wxPen(wxColour(0, 255, 0), 2), wxPen(wxColour(255, 165, 0), 2) };  
  
        dc.SetPen(pens[classIndex]);  
        dc.DrawLine(screenX1, screenY1, screenX2, screenY2);  
    }  
}
```


5.10. Ekran Çıktıları



Epoch ve learning rate girmezsek hata mesajı veriyor.





Girilen test değerlerine göre sınıf belirlemesini yapıp ekrana çıktısını veriyor.

