Technical Implementation

# Code

## ./CodeToWord.py

#convenience tool for passting code into word doc.  
from docx import Document  
import os  
#from docx.shared import Inches  
  
  
def add\_heading(text):  
 global document  
 document.add\_heading(text, level=2)  
  
def read\_code(direct):  
 global document  
 f=open(direct,"r", encoding="utf8")  
 text=f.read()  
 document.add\_paragraph(text)  
  
  
# def tes(argj:iter):  
   
  
def save\_doc():  
 global document  
  
 document.save('code.docx')  
  
if \_\_name\_\_=="\_\_main\_\_":  
 document = Document()  
  
 document.add\_heading('Technical Implementation', 0)  
 document.add\_heading("Code", level=1)  
  
 initial=os.getcwd()  
 print(os.getcwd())  
 y=input("y or n")  
 if (y=="y"):  
 for (root,dirs,files) in os.walk('.', topdown=True):  
 print(root)  
 for file in files:  
 ys=input(f"include {file}?")  
 if(ys=="y"):  
 add\_heading(root+"/"+file)  
 read\_code(root+"/"+file)  
 save\_doc()  
 dirscopy=dirs.copy()  
 for d in dirs:  
 inc=input(f"include dir {d}?")  
 if(inc=="n"):  
 dirscopy.remove(d)  
 [dirs.remove(d) for d in list(dirs) if not d in dirscopy]  
 #dirs[:] = [d for d in dirs if (lambda : input(f"include dir {d}?")=="n") ]

## ./setttingggjsonnexample.txt

{"0":{"Name":"Symbol","IsRegex":false,"Regex":"","IsCategorical":false,"Categories":[], "IsNumeric":false, "BoundsAndStep":[]},"1":{"Name":"Date From","IsRegex":true,"Regex":"[0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]","IsCategorical":false,"Categories":[], "IsNumeric":false, "BoundsAndStep":[]}, "2":{"Name":"Date To","IsRegex":true,"Regex":"[0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]","IsCategorical":false,"Categories":[], "IsNumeric":false, "BoundsAndStep":[]}}  
  
{"0":{"Name":"Symbol","IsRegex":false,"Regex":"","IsCategorical":false,"Categories":[], "IsNumeric":false, "BoundsAndStep":[]},"1":{"Name":"Date From","IsRegex":true,"Regex":"[0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]","IsCategorical":false,"Categories":[], "IsNumeric":false, "BoundsAndStep":[]}, "2":{"Name":"Date To","IsRegex":true,"Regex":"[0-9][0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]","IsCategorical":false,"Categories":[], "IsNumeric":false, "BoundsAndStep":[]}}  
  
{"Name":"Input","IsRegex":false,"Regex":"","IsCategorical":false,"Categories":[], "IsNumeric":true, "BoundsAndStep":[1.0E+6,-1.0E+6, 1.0E-3]}  
  
"IsNumeric":false, "BoundsAndStep":[]  
  
["NeaLibrary.Data.CalculateRSI","NeaLibrary.Data.CalculateStochastic","NeaLibrary.Data.CalculateMacd","NeaLibrary.Data.CalculateSmi","NeaLibrary.Data.CalculateCci","NeaLibrary.Data.CalculateSma","NeaLibrary.Data.CalculateAdx","NeaLibrary.Data.CalculateMfi","NeaLibrary.Data.CalculateAroon","NeaLibrary.Data.CalculateEma"]

## .\NeaLibrary/settings.json

{  
 "db\_dir": {  
  
 "Value": "C:/Users/s\_mah/source/repos/Nea/Models/database.sqlite3",  
 "TypeName": "System.String",  
 "Categorical": false,  
 "Categories": [],  
 "PathSelector": true,  
 "FileExtension": "\*.sqlite3",  
 "IsNumeric": false,  
 "BoundsAndStep": [],  
 "IsRegex": false,  
 "Regex": ""  
 },  
 "Random\_Batch\_Items\_Count": {  
  
 "Value": "100",  
 "TypeName": "System.Int32",  
 "Categorical": false,  
 "Categories": [],  
 "PathSelector": false,  
 "FileExtension": "",  
 "IsNumeric": true,  
 "BoundsAndStep": [1,250,1],  
 "IsRegex": false,  
 "Regex": ""  
 }  
}

## .\NeaLibrary\Data/Calculator.cs

﻿using Newtonsoft.Json.Linq;  
using Skender.Stock.Indicators;  
using System.Globalization;  
using System.Numerics;  
  
  
namespace NeaLibrary.Data  
{  
  
  
 public static class Calculator  
 {  
 private static SQL\_Driver db = new SQL\_Driver(Tools.Tools.GetGlobalVar("db\_dir"));  
 public static void CalculateRSI(string table, int n = 14)  
 {  
 using (IEnumerator<string> date = SQL\_Driver.ReadColumn<string>(db.conn, table, "Date", "1=1 ORDER BY Date ASC").GetEnumerator())  
 {  
 using (IEnumerator<double> rsi = Tools.Tools.RSI(SQL\_Driver.ReadColumn<double>(db.conn, table, "close", "1=1 ORDER BY Date ASC")).GetEnumerator())  
 {  
 while (date.MoveNext() && rsi.MoveNext())  
 {  
 //Console.SetCursorPosition(0, Console.CursorTop - 1);  
  
 //Console.WriteLine($"{date.Current} {rsi.Current}");  
 Dictionary<string, string> dic = new Dictionary<string, string>  
 {  
 { "RSI", rsi.Current.ToString() }  
 };  
 SQL\_Driver.Update(db.conn, table, dic, $"Date = '{date.Current}'");  
 }  
 }  
 }  
 }  
  
 public static IEnumerable<Quote> GetQuotes(string table)  
 {  
 using (IEnumerator<string> date = SQL\_Driver.ReadColumn<string>(db.conn, table, "Date", "1=1 ORDER BY Date ASC").GetEnumerator())  
 {  
 using (IEnumerator<NeaLibrary.DataStructures.Vector> data = SQL\_Driver.ReadRow\_AsVector(db.conn, table, new string[] { "open","high","low","close","volume" }, "1=1 ORDER BY Date ASC").GetEnumerator())  
 {  
 while (date.MoveNext() && data.MoveNext())  
 {  
 Quote quote = new Quote();  
 quote.Date = DateTime.Parse(date.Current.ToString());  
 quote.Open = (decimal)data.Current[0];  
 quote.High = (decimal)data.Current[1];  
 quote.Low = (decimal)data.Current[2];  
 quote.Close = (decimal)data.Current[3];  
 quote.Volume = (decimal)data.Current[4];  
 yield return quote;  
 }  
 }  
 }  
 }  
  
  
  
 public static void CalculateStochastic(string table, int lookback = 14)  
 {  
 IEnumerable<StochResult> results = GetQuotes(table).GetStoch(lookback);  
 foreach (StochResult r in results)  
 {  
 Dictionary<string,string> v = new Dictionary<string,string>();  
  
 if (r.Oscillator == null)  
 {  
 v.Add("StochasticOscillator", "0.5"); //default  
 }  
 else  
 {  
 v.Add("StochasticOscillator", (r.Oscillator/100).ToString());  
 }  
 SQL\_Driver.Update(db.conn, table, v, $"Date = '{r.Date.ToString("yyyy-MM-dd")}'");  
 }  
 }  
 public static void CalculateMacd(string table, int fast = 12, int slow=26)  
 {  
 IEnumerable<MacdResult> results = GetQuotes(table).GetMacd(fast, slow);  
 foreach (MacdResult r in results)  
 {  
 Dictionary<string, string> v = new Dictionary<string, string>();  
  
 if (r.Signal == null)  
 {  
 v.Add("MACD", "0"); //default  
 }  
 else  
 {  
 v.Add("MACD", (r.Signal).ToString());  
 }  
 SQL\_Driver.Update(db.conn, table, v, $"Date = '{r.Date.ToString("yyyy-MM-dd")}'");  
 }  
 }  
 public static void CalculateSmi(string table, int lookback = 14, int firstsmoothing=3, int secondsmoothing=3)  
 {  
 IEnumerable<SmiResult> results = GetQuotes(table).GetSmi(lookback,firstsmoothing,secondsmoothing);  
 foreach (SmiResult r in results)  
 {  
 Dictionary<string, string> v = new Dictionary<string, string>();  
  
 if (r.Smi == null)  
 {  
 v.Add("SMI", "0"); //default  
 }  
 else  
 {  
 v.Add("SMI", (r.Signal).ToString());  
 }  
 SQL\_Driver.Update(db.conn, table, v, $"Date = '{r.Date.ToString("yyyy-MM-dd")}'");  
 }  
 }  
 public static void CalculateCci(string table, int lookback = 20)  
 {  
 IEnumerable<CciResult> results = GetQuotes(table).GetCci(lookback);  
 foreach (CciResult r in results)  
 {  
 Dictionary<string, string> v = new Dictionary<string, string>();  
  
 if (r.Cci == null)  
 {  
 v.Add("CCI", "0"); //default  
 }  
 else  
 {  
 v.Add("CCI", (r.Cci/100).ToString());  
 }  
 SQL\_Driver.Update(db.conn, table, v, $"Date = '{r.Date.ToString("yyyy-MM-dd")}'");  
 }  
 }  
 public static void CalculateSma(string table, int lookback = 14)  
 {  
 IEnumerator<Quote> qs = GetQuotes(table).GetEnumerator();  
 IEnumerable<SmaResult> results = GetQuotes(table).GetSma(lookback);  
 foreach (SmaResult r in results)  
 {  
 Dictionary<string, string> v = new Dictionary<string, string>();  
  
 if (r.Sma == null)  
 {  
 qs.MoveNext();  
 v.Add("SMA", qs.Current.Close.ToString()); //default  
 }  
 else  
 {  
 v.Add("SMA", (r.Sma).ToString());  
 }  
 SQL\_Driver.Update(db.conn, table, v, $"Date = '{r.Date.ToString("yyyy-MM-dd")}'");  
 }  
 }  
  
 public static void CalculateAdx(string table, int lookback = 14)  
 {  
 IEnumerable<AdxResult> results = GetQuotes(table).GetAdx(lookback);  
 foreach (AdxResult r in results)  
 {  
 Dictionary<string, string> v = new Dictionary<string, string>();  
  
 if (r.Adx == null)  
 {  
 v.Add("ADX", "0"); //default  
 }  
 else  
 {  
 v.Add("ADX", (r.Adx/100).ToString());  
 }  
 SQL\_Driver.Update(db.conn, table, v, $"Date = '{r.Date.ToString("yyyy-MM-dd")}'");  
 }  
 }  
  
 public static void CalculateMfi(string table, int lookback = 14)  
 {  
 IEnumerable<MfiResult> results = GetQuotes(table).GetMfi(lookback);  
 foreach (MfiResult r in results)  
 {  
 Dictionary<string, string> v = new Dictionary<string, string>();  
  
 if (r.Mfi == null)  
 {  
 v.Add("MFI", "0.5"); //default  
 }  
 else  
 {  
 v.Add("MFI", (r.Mfi / 100).ToString());  
 }  
 SQL\_Driver.Update(db.conn, table, v, $"Date = '{r.Date.ToString("yyyy-MM-dd")}'");  
 }  
 }  
 public static void CalculateAroon(string table, int lookback = 25)  
 {  
 IEnumerable<AroonResult> results = GetQuotes(table).GetAroon(lookback);  
 foreach (AroonResult r in results)  
 {  
 Dictionary<string, string> v = new Dictionary<string, string>();  
  
 if (r.Oscillator == null)  
 {  
 v.Add("Aroon", "0"); //default  
 }  
 else  
 {  
 v.Add("Aroon", (r.Oscillator/100).ToString());  
 }  
 SQL\_Driver.Update(db.conn, table, v, $"Date = '{r.Date.ToString("yyyy-MM-dd")}'");  
 }  
 }  
  
 /// <summary>  
 /// standard 20 day short term trading ema  
 /// </summary>  
 /// <param name="table"></param>  
 /// <param name="lookback"></param>  
 public static void CalculateEma(string table, int lookback = 20)  
 {  
 IEnumerator<Quote> qs = GetQuotes(table).GetEnumerator();  
 IEnumerable<EmaResult> results = GetQuotes(table).GetEma(lookback);  
 foreach (EmaResult r in results)  
 {  
 Dictionary<string, string> v = new Dictionary<string, string>();  
  
 if (r.Ema == null)  
 {  
 qs.MoveNext();  
 v.Add("EMA", qs.Current.Close.ToString()); //default  
 }  
 else  
 {  
 v.Add("EMA", (r.Ema).ToString());  
 }  
 SQL\_Driver.Update(db.conn, table, v, $"Date = '{r.Date.ToString("yyyy-MM-dd")}'");  
 }  
 }  
  
 /// <summary>  
 /// Look up all the methods to use in calculation of the specified table  
 /// </summary>  
 public static void Calculate(string table)  
 {  
 //string type = "";  
 //using (var r = SQL\_Driver.Query(db.conn, $"SELECT AssetType FROM TokenNames WHERE Token='{table}'"))  
 //{  
 // type  
 //}  
 string json\_list\_commands = "";  
 using (var r = SQL\_Driver.Query(db.conn, $"SELECT CalculatorOperations FROM AssetType, TokenNames WHERE TokenNames.AssetType = AssetType.AssetType AND TokenNames.Token = '{table}';"))  
 {  
 r.Read();  
 json\_list\_commands = r.GetString(0);  
 }  
 JArray jArray = JArray.Parse(json\_list\_commands);  
 foreach (JValue jval in jArray)  
 {  
 string methodName = jval.ToString();  
 string className = String.Join(".",methodName.Split(".").SkipLast(1));  
 string funcName = methodName.Split(".").Last();  
 var type = Type.GetType(className);  
   
 try  
 {  
 var methodInfo = type.GetMethod(funcName);  
 // according to https://stackoverflow.com/questions/11908156/call-static-method-with-reflection  
 //https://stackoverflow.com/questions/2421994/invoking-methods-with-optional-parameters-through-reflection  
 int paramcount= methodInfo.GetParameters().Length;  
 var param =new object[paramcount];  
 param[0] = table;  
 for(int i=1;i<methodInfo.GetParameters().Length;i++) param[i]=Type.Missing;  
 methodInfo.Invoke(null, param);  
 }catch (Exception ex)  
 {  
 continue;//multiple methods of same name, do nothing  
 //or invalid method invoked  
 }  
   
 }  
 }  
  
 //public static void CalculateRSI(string table, int n = 14)  
 //{  
 // using (IEnumerator<string> date = SQL\_Driver.ReadColumn<string>(db.conn, table, "Date", "1=1 ORDER BY Date ASC").GetEnumerator())  
 // {  
 // using (IEnumerator<double> rsi = Tools.Tools.RSI(SQL\_Driver.ReadColumn<double>(db.conn, table, "close", "1=1 ORDER BY Date ASC")).GetEnumerator())  
 // {  
 // while (date.MoveNext() && rsi.MoveNext())  
 // {  
 // //Console.SetCursorPosition(0, Console.CursorTop - 1);  
  
 // //Console.WriteLine($"{date.Current} {rsi.Current}");  
 // Dictionary<string, string> dic = new Dictionary<string, string>  
 // {  
 // { "RSI", rsi.Current.ToString() }  
 // };  
 // SQL\_Driver.Update(db.conn, table, dic, $"Date = '{date.Current}'");  
 // }  
 // }  
 // }  
 //}  
 }   
}

## .\NeaLibrary\Data/DataHandler.cs

using Newtonsoft.Json;  
using Newtonsoft.Json.Linq;  
using System.Data;  
using System.Data.SQLite;  
  
using NeaLibrary.DataStructures;  
using System.Collections.Generic;  
using System;  
using System.IO;  
using NeaLibrary.WebRequests;  
using System.Configuration;  
using NeaLibrary.Tools;  
using Skender.Stock.Indicators;  
using System.Globalization;  
  
namespace NeaLibrary.Data  
{  
 public class Data\_Handler{  
  
 string? json\_doc;  
  
 SQL\_Driver DB;  
   
 public Data\_Handler(string db){  
  
 //string path = System.AppContext.BaseDirectory;  
 //#nullable enable  
 //path = Directory.GetParent(path).ToString();  
 //path = Directory.GetParent(path).ToString();  
  
 //path = Directory.GetParent(path).ToString();  
 //path = Directory.GetParent(path).ToString();  
 //#nullable disable  
 //path+=$"/{db}";  
 DB= new SQL\_Driver(db);  
  
 }  
 public Data\_Handler()  
 {  
  
 // string path = System.AppContext.BaseDirectory;  
 //#nullable enable  
 // path = Directory.GetParent(path).ToString();  
 // path = Directory.GetParent(path).ToString();  
  
 // path = Directory.GetParent(path).ToString();  
 // path = Directory.GetParent(path).ToString();  
 //#nullable disable  
 // path += $"/{Tools.Tools.GetGlobalVar("db\_dir")}";  
 string path = Tools.Tools.GetGlobalVar("db\_dir");  
 DB = new SQL\_Driver(path);  
  
 }  
 public (Vector[],Vector[]) produceTrainingData(){  
 List<Vector> ins= new List<Vector>();  
  
 using (SQLiteDataReader reader =SQL\_Driver.Query(DB.conn,$"SELECT \* FROM IBM WHERE Date>'2000-1-1'ORDER BY Date;")){  
 reader.Read();  
  
 double prev\_open = (double)reader.GetValue(1);  
 double prev\_ema = (double)reader.GetValue(6);  
 reader.Read();  
 double open = (double)reader.GetValue(1);  
 double ema = (double)reader.GetValue(6);  
 Vector v = new Vector(2);  
 v.vector = new double[]{open/prev\_open,ema/prev\_ema};  
 ins.Add(v);  
 prev\_open = open;   
 prev\_ema = ema;  
 while(reader.Read()){  
 open = (double)reader.GetValue(1);  
 ema = (double)reader.GetValue(6);  
 Vector V = new Vector(2);  
 V.vector = new double[]{open/prev\_open,ema/prev\_ema};  
 prev\_open = open;   
 prev\_ema = ema;  
 ins.Add(V);  
 }  
 }  
 Vector[] INS = new Vector[ins.Count-1];  
 Vector[] OUT = new Vector[ins.Count-1];  
 for(int i =0;i<ins.Count-1;i++){  
 INS[i] = ins[i];  
 OUT[i] = ins[i+1];  
 }  
   
 return (INS,OUT);  
   
  
 }  
  
 /// <summary>  
 /// Fetch data from API as described by Sources table in the database  
 /// </summary>  
 /// <param name="SourceID"></param>  
 /// <param name="args"></param>  
 /// <param name="upsert">Determine whether Upsert behaviour is used, if false the strict Insert</param>  
 public string Fetch(int SourceID,string[] args,bool upsert=true){  
 int fetched = 0;  
 string baseRequest;  
 string requestArgs;  
 long namePos;  
 string assetType;  
 string DDL\_table\_struct;  
 string responsemap;  
 string rootmap;  
 string dateFormat;  
 string culture;  
 using (SQLiteDataReader reader =SQL\_Driver.Query(DB.conn,$"SELECT Source, Parameters, AssetType, ResponseMap, ResponseRoot, NamePositionInParameters, DateFormat FROM Sources WHERE SourceID={SourceID}")){  
 reader.Read();  
 baseRequest = (string)reader.GetValue(0);  
 requestArgs = (string)reader.GetValue(1);  
 assetType = (string)reader.GetValue(2);  
 responsemap = (string)reader.GetValue(3);  
 rootmap = (string)reader.GetValue(4);  
 namePos = (long)reader.GetValue(5);  
 culture = (string)reader.GetValue(6);  
 using (SQLiteDataReader reader2 =SQL\_Driver.Query(DB.conn,$"SELECT Fields FROM AssetType WHERE AssetType='{assetType}'")){  
 reader2.Read();  
 DDL\_table\_struct = (string)reader2.GetValue(0);  
 }  
 using (SQLiteDataReader reader2 = SQL\_Driver.Query(DB.conn, $"SELECT DateFormat FROM Culture WHERE Culture='{culture}'"))  
 {  
 reader2.Read();  
 dateFormat = (string)reader2.GetValue(0);  
 }  
 }  
 requestArgs=String.Format(requestArgs,args);  
 string req = baseRequest+requestArgs;  
   
 string json\_doc = WebRequestsHandler.Req(req);  
   
 JToken json = JToken.Parse(json\_doc);  
  
 Console.WriteLine($"Fetched {args[namePos]} from source {SourceID}: {req}");  
 Console.WriteLine();  
 JArray RootMap = (JArray)JsonConvert.DeserializeObject(rootmap);  
 foreach(var d in RootMap){  
 json = json[d.Value<string>()];  
 }//json now reduced to root  
 //i coded this before i found out JObject.SelectToken() perhaps change in future  
   
  
 SQL\_Driver.CreateTable(DB.conn,args[namePos],DDL\_table\_struct);  
 //if it doesnt exist for new tokens  
 Dictionary<string,string> token = new Dictionary<string,string>();  
 token.Add("Token", "'"+args[namePos]+"'");  
 token.Add("AssetType", "'" + assetType+"'");  
 SQL\_Driver.InserOnConflictUpdate\_Data(DB.conn, "TokenNames", "Token", "AssetType=excluded.AssetType",token);  
  
  
 string table = args[namePos];  
  
 // JObject values = (JObject)json;  
  
 //var values = (json is JArray)? (JArray)json:(JObject)json;  
  
 JObject ResponseMap = JObject.Parse(responsemap);  
   
 void AddData(Dictionary<string,string> data)  
 {  
 if (!upsert)  
 {  
 try  
 {  
 SQL\_Driver.InsertData(DB.conn, table, data);  
 fetched++;  
 Console.SetCursorPosition(0, Console.CursorTop - 1);  
 Console.WriteLine($"Fetched results: {fetched}");  
 }  
 catch  
 {  
 //log  
  
 }  
 }  
 else  
 {  
 SQL\_Driver.Upsert(DB.conn, table, data, "Date");  
 }  
 }  
 DateTime ParseDate(string date)  
 {  
 if (culture == "UnixMsec")  
 {  
 return DateTime.UnixEpoch.AddMilliseconds(Convert.ToDouble(date));  
 }  
 else  
 {  
 var dateculture = CultureInfo.CreateSpecificCulture(culture); //should be country code  
 return DateTime.ParseExact(date, dateFormat, dateculture);  
 }  
 }  
 if (json is JArray) {  
 var values = (JArray)json;  
 string date\_key = "";  
 foreach (KeyValuePair<string, JToken> maptoken in ResponseMap)  
 {  
 if ((string)maptoken.Value=="Date")  
 {  
 date\_key = maptoken.Key;  
 break;  
 }  
 }  
  
 foreach (JObject quote in values) {  
  
 DateTime t = ParseDate((string)quote[date\_key]);  
 Dictionary<string, string> dic = new Dictionary<string, string>();  
  
 dic.Add("Date", $"'{t.ToString("yyyy-MM-dd")}'"); //keep the quote marks? want always the format yyyy-MM-dd  
 //extra ' so text gets passed correctly  
 //pair.Value[];  
 foreach (KeyValuePair<string, JToken> map in ResponseMap)  
 {  
 if (map.Key == date\_key) continue;  
  
 //map.Key matches response. map.Value matches DB  
 string item\_value = (string)quote[map.Key];  
 dic.Add($"{(string)map.Value}", item\_value);  
 }  
 AddData(dic);  
 }  
  
 } else {  
  
 var values = (JObject)json;  
 foreach (KeyValuePair<string, JToken> quote in values)  
 {  
 //foreach (JProperty record in cur.Properties())  
 //{  
 //Console.WriteLine($"{pair.Key}");  
 DateTime t = ParseDate(quote.Key);  
 //if (culture=="UnixMsec") {  
 // t= DateTime.UnixEpoch.AddMilliseconds(Convert.ToDouble(quote.Key));  
 //}  
 //else  
 //{  
 // t = DateTime.Parse(quote.Key);  
 //}  
 Dictionary<string, string> dic = new Dictionary<string, string>();  
  
 dic.Add("Date", $"'{t.ToString("yyyy-MM-dd")}'"); //keep the quote marks? want always the format yyyy-MM-dd  
 //extra ' so text gets passed correctly  
 //pair.Value[];  
 foreach (KeyValuePair<string, JToken> map in ResponseMap)  
 {  
 //map.Key matches response. map.Value matches DB  
 string r = (string)quote.Value[map.Key];  
 dic.Add($"{(string)map.Value}", r);  
 }  
 AddData(dic);  
 }  
 }  
  
 //Console.WriteLine("Complete!");  
 return table;  
 }  
  
 }  
}

## .\NeaLibrary\Data/DB\_DataSet.cs

﻿using NeaLibrary.Data.Other;  
using NeaLibrary.DataStructures;  
using System.Data.SQLite;  
using System.Drawing;  
using System.Globalization;  
using System.Runtime.Serialization.Formatters.Binary;  
using System.Security.Policy;  
  
namespace NeaLibrary.Data  
{  
  
 public class DB\_DataSet : IDataSet  
 {  
 private string sql\_query;  
 private string from\_string;  
 private string cols\_string;  
 public List<string> tables;  
 public List<string> cols;  
 public List<string> normalise\_cols; private List<int> normalise\_indexes;  
 public List<string> relative\_cols; private List<int> relative\_indexes;  
 public List<string> smooth\_cols; private List<int> smooth\_indexes;  
  
 public List<string> ValueColumn; private List<int> ValueIndexes;  
  
 public List<int> GetValueIndexes() => ValueIndexes;  
  
 public double safety;  
  
 private int smoother\_steps;  
 private int INTERNAL\_DB\_OFFSET = 30; //used to make sure no NULL values are encountered  
  
 public int GetInternalDbOffset() => INTERNAL\_DB\_OFFSET;  
  
 private SortedDictionary<int, (Vector, Vector)> cache = new SortedDictionary<int, (Vector, Vector)>();  
 private object cacheLock = new object();  
  
 private Dictionary<string, int> table\_start = new Dictionary<string, int>();  
  
 private Func<double, double> Normaliser = Tools.Tools.Sigmoid;  
 private Func<IEnumerable<double>, double> Smoother = x => x.Sum() / x.Count(); // the smoother should return an average of the couple of values in the time series, first derivative doesnt really  
  
 private int cached\_size = -1;  
 public InputMapCache InputMapCache;  
 public void SetInputMapCache(InputMapCache inputMapCache)  
 {  
 this.InputMapCache = inputMapCache;  
 }  
 public InputMapCache GetInputMapCache()  
 {  
 return this.InputMapCache;  
 }  
  
 public double Compound { get; }  
  
 public object driver\_lock = new object();  
 public SQL\_Driver? driver;  
  
  
 [field: NonSerialized]  
 public event EventHandler<List<(Vector, Vector)>> ToListComplete;  
  
 //(Vector, Vector) IDataSet.this[int n] { get => Batch(1, n).First() ; set => throw new NotImplementedException(); }  
 (Vector, Vector) IDataSet.this[int n]  
 {  
 get  
 {  
 if (cache.ContainsKey(n)) return cache[n];  
 return Batch(1, n).First();  
 }  
 set => throw new NotImplementedException();  
 }  
  
 private void InitialiseSQL()  
 {  
 //this.sql\_query = sql\_query;  
 // string path = System.AppContext.BaseDirectory;  
 //#nullable enable  
 // path = Directory.GetParent(path).ToString();  
 // path = Directory.GetParent(path).ToString();  
  
 // path = Directory.GetParent(path).ToString();  
 // path = Directory.GetParent(path).ToString();  
 // path = Directory.GetParent(path).ToString();  
 //#nullable disable  
 if (driver != null) return;  
 string path = Tools.Tools.GetGlobalVar("db\_dir");  
 lock (driver\_lock) {  
 driver = new SQL\_Driver(path);  
 }  
  
 }  
  
 private void ReCalcTableStart()  
 {  
 table\_start = new Dictionary<string, int>();  
 int pos = 0;  
 tables.Sort(); // so would be deterministic in which tables come first. Allows for replication . is sort in place?  
 foreach (string table in tables)  
 {  
 if (driver == null) throw new Exception("Database not initialised");  
 table\_start.Add(table, pos);  
 SQLiteDataReader read = SQL\_Driver.Query(driver.conn, $"SELECT Count(\*) FROM {table}"); //No taking away the offset, this is CORRECT, bc iterator\_one\_table works so  
 read.Read();  
 //pos += read.GetInt32(0) + 1000; //wiggle room In the very rare case should the DB get more values while DB\_DataSet was still loaded so one table has more room than allocated?  
 //cache could be overwritten if more data than wiggle room is added (approx 1000 days worth) and RandomBatch will give wrong result  
  
 //wiggle room redundant. We are assuming data wont be added to database table while the DB\_DataSet is open. once closed more can be added and upon reoppening recalctablestart will do its job  
 }  
 }  
  
  
 public void Cache()  
 {  
 //generate wholy  
 foreach (string table in table\_start.Keys)  
 {  
 iterator\_one\_table(Count(), 0, table);  
 }  
 }  
 public void ClearCache()  
 {  
 cache.Clear();  
 }  
 private IEnumerable<(Vector, Vector)> iterator\_one\_table(int size, int start, string table) // size and start should be within this table  
 {  
 start += INTERNAL\_DB\_OFFSET;  
 int real\_item\_index = start;  
 bool Cached = true;  
  
  
 ////LOCK HERE? Read operation only so no need  
  
 // IEnumerable<Vector> input = SQL\_Driver.ReadRow\_AsVector(driver.conn, /\*"(" + from\_string + ")"\*/ table, cols, $"1=1 ORDER BY Date ASC LIMIT {size} OFFSET {start}");  
  
 // TrainingDataTransformer tr = new TrainingDataTransformer(input, //MAYBE MOCE THESE OUTSIDE? SO NOT SPAWNED EVERYTIME  
 // relative\_indexes,  
 // normalise\_indexes,  
 // Normaliser,  
 // smooth\_indexes,  
 // Smoother,  
 // smoother\_steps  
 // ); //$"SELECT \* FROM ({from\_string}) LIMIT {size} OFFSET {start};")  
  
 // OutputDataProducer output = new OutputDataProducer(safety, 3, input, ValueIndexes, ValueIndexes.Count() / 2); //TODO SAFETY  
 // IEnumerator<Vector> Tr = tr.GetEnumerator();  
 // IEnumerator<Vector> Output = output.GetEnumerator();  
 // for (int i = start; i < size + start; i++)  
 // {  
 // if (cache.ContainsKey(i))  
 // {  
 // yield return cache[i];  
 // }  
 // else  
 // {  
 // //IEnumerable<Vector> input = SQL\_Driver.ReadRow\_AsVector(driver.conn, "(" + from\_string + ")", cols, $"1=1 ORDER BY Date ASC LIMIT {size} OFFSET {start}");  
 // //TrainingDataTransformer tr = new TrainingDataTransformer(input, //MAYBE MOCE THESE OUTSIDE? SO NOT SPAWNED EVERYTIME  
 // // relative\_indexes,  
 // // normalise\_indexes,  
 // // Normaliser,  
 // // smooth\_indexes,  
 // // Smoother,  
 // // smoother\_steps  
 // // ); //$"SELECT \* FROM ({from\_string}) LIMIT {size} OFFSET {start};")  
  
 // //OutputDataProducer output = new OutputDataProducer(safety, 3, tr, ValueIndexes, ValueIndexes.Count() / 2); //TODO SAFETY  
 // //IEnumerator<Vector> Tr = tr.GetEnumerator();  
 // //IEnumerator<Vector> Output = output.GetEnumerator();  
 // //while (Tr.MoveNext() && Output.MoveNext())  
 // //{  
  
 // try{  
 // Tr.MoveNext(); Output.MoveNext();  
 // }  
 // catch { break; }  
  
 // if (!cache.ContainsKey(table\_start[table] + i )) { cache.Add(table\_start[table] + i, (Tr.Current, Output.Current)); }  
 // yield return (Tr.Current, Output.Current);  
 // real\_item\_index++;  
 // //}  
  
 // //break; //brak the loop  
 // }  
 // }  
  
 // yield break;  
  
  
 IEnumerable<Vector> input = SQL\_Driver.ReadRow\_AsVector(driver.conn, /\*"(" + from\_string + ")"\*/ table, cols, $"1=1 ORDER BY Date ASC LIMIT {size} OFFSET {start}");  
 //IEnumerable<string> d = SQL\_Driver.ReadColumn<string>(driver.conn, /\*"(" + from\_string + ")"\*/ table, "Date", $"1=1 ORDER BY Date ASC LIMIT {size} OFFSET {start}");  
 /\*IEnumerator<string> D = d.GetEnumerator();\*/ IEnumerator<Vector> Input = input.GetEnumerator();  
 TrainingDataTransformer data\_transformer = new TrainingDataTransformer(input, //MAYBE MOCE THESE OUTSIDE? SO NOT INITIATED EVERYTIME  
 relative\_indexes,  
 normalise\_indexes,  
 Normaliser,  
 smooth\_indexes,  
 Smoother,  
 smoother\_steps  
 ); //$"SELECT \* FROM ({from\_string}) LIMIT {size} OFFSET {start};")  
  
 OutputDataProducer output = new OutputDataProducer(safety, 3, input, ValueIndexes, ValueIndexes.Count() / 2); //TODO SAFETY  
 IEnumerator<Vector> TrainingDataTransformerEnumerator = data\_transformer.GetEnumerator();  
 IEnumerator<Vector> OutputEnumerator = output.GetEnumerator();  
 for (int i = start; i < size + start; i++)  
 {  
 if (cache.ContainsKey(table\_start[table] + i ))  
 {  
 yield return cache[i];  
 }  
 //IEnumerable<Vector> input = SQL\_Driver.ReadRow\_AsVector(driver.conn, "(" + from\_string + ")", cols, $"1=1 ORDER BY Date ASC LIMIT {size} OFFSET {start}");  
 //TrainingDataTransformer tr = new TrainingDataTransformer(input, //MAYBE MOCE THESE OUTSIDE? SO NOT SPAWNED EVERYTIME  
 // relative\_indexes,  
 // normalise\_indexes,  
 // Normaliser,  
 // smooth\_indexes,  
 // Smoother,  
 // smoother\_steps  
 // ); //$"SELECT \* FROM ({from\_string}) LIMIT {size} OFFSET {start};")  
  
 //OutputDataProducer output = new OutputDataProducer(safety, 3, tr, ValueIndexes, ValueIndexes.Count() / 2); //TODO SAFETY  
 //IEnumerator<Vector> Tr = tr.GetEnumerator();  
 //IEnumerator<Vector> Output = output.GetEnumerator();  
 //while (Tr.MoveNext() && Output.MoveNext())  
 //{  
  
  
 try  
 {  
 TrainingDataTransformerEnumerator.MoveNext();  
 OutputEnumerator.MoveNext();  
 }  
 catch { break;/\*ran out of elements to yield\*/ }  
  
 if (!cache.ContainsKey(table\_start[table] + i)) {  
 lock (cacheLock)  
 {  
 if (!cache.ContainsKey(table\_start[table] + i)) //a second check required as 2 threads could havereached this state waiting for lock realease to add same value to the cache  
 {  
 cache.Add(table\_start[table] + i, (TrainingDataTransformerEnumerator.Current, OutputEnumerator.Current));  
 }  
 //sufficient as only 1 thread can have a lock at this time, so no race conditions, and no 2 threads can add the same item to add  
 }  
 }  
 yield return (TrainingDataTransformerEnumerator.Current, OutputEnumerator.Current);  
  
 //}  
  
 //break; //brak the loop  
  
 }  
 yield break;  
  
 }  
  
 public IEnumerable<(DateTime, Vector, Vector, Vector)> Debug(int size, int start, string table)  
 {  
 IEnumerable<Vector> input = SQL\_Driver.ReadRow\_AsVector(driver.conn, /\*"(" + from\_string + ")"\*/ table, cols, $"1=1 ORDER BY Date ASC LIMIT {size} OFFSET {start}");  
 IEnumerable<string> d = SQL\_Driver.ReadColumn<string>(driver.conn, /\*"(" + from\_string + ")"\*/ table, "Date", $"1=1 ORDER BY Date ASC LIMIT {size} OFFSET {start}");  
 IEnumerator<string> D = d.GetEnumerator(); IEnumerator<Vector> Input = input.GetEnumerator();  
 TrainingDataTransformer tr = new TrainingDataTransformer(input, //MAYBE MOCE THESE OUTSIDE? SO NOT SPAWNED EVERYTIME  
 relative\_indexes,  
 normalise\_indexes,  
 Normaliser,  
 smooth\_indexes,  
 Smoother,  
 smoother\_steps  
 ); //$"SELECT \* FROM ({from\_string}) LIMIT {size} OFFSET {start};")  
  
 OutputDataProducer output = new OutputDataProducer(safety, 3, input, ValueIndexes, ValueIndexes.Count() / 2); //TODO SAFETY  
 IEnumerator<Vector> Tr = tr.GetEnumerator();  
 IEnumerator<Vector> Output = output.GetEnumerator();  
 for (int i = start; i < size + start; i++)  
 {  
  
 //IEnumerable<Vector> input = SQL\_Driver.ReadRow\_AsVector(driver.conn, "(" + from\_string + ")", cols, $"1=1 ORDER BY Date ASC LIMIT {size} OFFSET {start}");  
 //TrainingDataTransformer tr = new TrainingDataTransformer(input, //MAYBE MOCE THESE OUTSIDE? SO NOT SPAWNED EVERYTIME  
 // relative\_indexes,  
 // normalise\_indexes,  
 // Normaliser,  
 // smooth\_indexes,  
 // Smoother,  
 // smoother\_steps  
 // ); //$"SELECT \* FROM ({from\_string}) LIMIT {size} OFFSET {start};")  
  
 //OutputDataProducer output = new OutputDataProducer(safety, 3, tr, ValueIndexes, ValueIndexes.Count() / 2); //TODO SAFETY  
 //IEnumerator<Vector> Tr = tr.GetEnumerator();  
 //IEnumerator<Vector> Output = output.GetEnumerator();  
 //while (Tr.MoveNext() && Output.MoveNext())  
 //{  
  
 try  
 {  
 D.MoveNext(); Input.MoveNext(); Tr.MoveNext(); Output.MoveNext();  
 }  
 catch { break; }  
  
 //if (!cache.ContainsKey(table\_start[table] + i)) { cache.Add(table\_start[table] + i, (Tr.Current, Output.Current)); }  
 yield return (DateTime.Parse( D.Current), Input.Current ,Tr.Current, Output.Current);  
  
 //}  
  
 //break; //brak the loop  
  
 }  
 }  
  
 //public IEnumerable<(DateTime, (Vector,Vector))> GetAllWithDateTime()  
 //{  
 // yield break;  
 //} redundant  
  
 //public IEnumerable<(Vector, Vector)> CACHE()  
 //{  
 // int start = INTERNAL\_DB\_OFFSET;  
 // int index = start;  
 // if (driver == null) InitialiseSQL();  
 // IEnumerable<Vector> input = SQL\_Driver.ReadRow\_AsVector(driver!.conn, "(" + from\_string + ")", cols, $"1=1 ORDER BY Date ASC LIMIT -1 OFFSET {start};");  
 // TrainingDataTransformer tr = new TrainingDataTransformer(input,  
 // relative\_indexes,  
 // normalise\_indexes,  
 // Normaliser,  
 // smooth\_indexes,  
 // Smoother,  
 // smoother\_steps  
 // ); //$"SELECT \* FROM ({from\_string}) LIMIT {size} OFFSET {start};")  
  
 // OutputDataProducer output = new OutputDataProducer(safety, 3, input, ValueIndexes, ValueIndexes.Count() / 2); //TODO SAFETY  
 // IEnumerator<Vector> Tr = tr.GetEnumerator();  
 // IEnumerator<Vector> Output = output.GetEnumerator();  
 // while (Tr.MoveNext() && Output.MoveNext())  
 // {  
 // if (!cache.ContainsKey(index)) { cache.Add(index, (Tr.Current, Output.Current)); }  
 // yield return (Tr.Current, Output.Current);  
 // index++;  
 // }  
  
 //}  
   
  
 /// <summary>  
 /// start and size only include items that can be enumerated. Which means those that are past the DB OFFSET.  
 /// This makes it the same as those accounted by .Count()  
 /// </summary>  
 /// <param name="size"></param>  
 /// <param name="start"></param>  
 /// <returns></returns>  
 public IEnumerable<(Vector, Vector)> Batch(int size, int start)  
 {  
  
 List<string> tables\_spanned = new List<string>();  
 int yielded = 0;  
 int items\_count\_from\_tables = 0;  
 foreach (KeyValuePair<string, int> v in table\_start)  
 {  
 if (v.Value >= start) { tables\_spanned.Add(v.Key); items\_count\_from\_tables+=Count(v.Key); if (items\_count\_from\_tables>=size) { break; } }//this includes subtracting DB OFFSET for each table  
 }  
 //bool complete=false;  
  
 bool first\_table=true;  
  
 foreach (string table in tables\_spanned) {// V determines the "virtual address"  
 foreach((Vector,Vector) val in iterator\_one\_table(size, (first\_table)? start - table\_start[table] : 0 , table))  
 //if we wanna start a 1000 in, first table starts from 600, then we wanna start first table from 400  
 //ik i ignored db offset -> accounted now. each table will start from 0 other than first  
 //all other tables after that are contiguous and thus start from 0. ik i need to fix the size requirement  
 //but it doesnt really matter in reality bc we have the "yielded" counter to stop us  
 //size >= (size of individual table) so this is ok too  
 {  
 yield return val;  
 yielded++;  
 if (yielded >= size) break;  
 }  
 first\_table = false;  
 if (yielded >= size) break;  
 }  
 }  
  
 /// <summary>  
 /// SPOILER: doesnt acctually yield next batch. risk of thread safety was taken into consideration  
 /// </summary>  
 /// <param name="size"></param>  
 /// <param name="start"></param>  
 /// <returns></returns>  
 public IEnumerable<(Vector, Vector)> NextBatch(int size, int start)  
 {  
 int returned = 0;  
  
 foreach (var v in Batch(Count(), start)) //no need to men tion offset here it is accounted in the normal batch method  
 {  
 if (returned > size) yield break;  
 yield return v;  
 returned++;  
 }  
 }  
 /// <summary>  
 /// Only cached items can be selected. So I Cache() first  
 /// </summary>  
 /// <param name="size"></param>  
 /// <returns></returns>  
 public IEnumerable<(Vector, Vector)> RandomBatch(int size)  
 {  
 for (int i = 0; i < size; i++)  
 {  
 if (cache.Count == 0)  
 {  
 yield return NextBatch(size, 0).ElementAt(Tools.Tools.RandomInt(0,size-1)); //as RandomInt is inclusive  
 continue;  
 }  
 //yield return (cache[Tools.Tools.RandomInt(0, Count() - 1)]); //-1 cuz 0 indexed and random inculsive  
 yield return cache.ElementAt(Tools.Tools.RandomInt(0, cache.Count() - 1)).Value;  
 }  
 yield break;  
 //yield return cache.ElementAt(Tools.Tools.RandomInt(0, cache.Count()-1)).Value;  
 }  
 public int Count()  
 {  
 if (cached\_size != -1) return cached\_size;  
 //using (SQLiteDataReader r = SQL\_Driver.Query(driver.conn, $"SELECT COUNT(\*) FROM ({from\_string});"))  
 //{  
 // r.Read();  
  
 // cached\_size = r.GetInt32(0) /\*- INTERNAL\_DB\_OFFSET\*/ - INTERNAL\_DB\_OFFSET\*table\_start.Count(); //Hm?  
 // return cached\_size;  
 //}  
 int size = 0;  
 foreach(string t in tables) { size += Count(t); }  
 cached\_size = size;  
 return size;  
 }  
 public int Count(string table)  
 {  
 //if (cached\_size != -1) return cached\_size;  
 using (SQLiteDataReader r = SQL\_Driver.Query(driver.conn, $"SELECT COUNT(\*) FROM ({table});"))  
 {  
 r.Read();  
 //cached\_size = r.GetInt32(0) - INTERNAL\_DB\_OFFSET;  
 return r.GetInt32(0) - INTERNAL\_DB\_OFFSET;  
 }  
 }  
  
 IEnumerator<(Vector, Vector)> IEnumerable<(Vector, Vector)>.GetEnumerator()  
 {  
  
 return Batch(Count(), 0).GetEnumerator();  
 ////return IDataSet.NextBatch().GetEnumerator();  
  
 //IEnumerable<Vector> input = SQL\_Driver.ReadRow\_AsVector(driver.conn, "(" + from\_string + ")", cols, $"1=1 ORDER BY Date ASC LIMIT -1 OFFSET {INTERNAL\_DB\_OFFSET}");  
 //TrainingDataTransformer tr = new TrainingDataTransformer(input,  
 // relative\_indexes,  
 // normalise\_indexes,  
 // Normaliser,  
 // smooth\_indexes,  
 // Smoother,  
 // smoother\_steps  
 // ); //$"SELECT \* FROM ({from\_string}) LIMIT {size} OFFSET {start};")  
  
 //OutputDataProducer output = new OutputDataProducer(safety, 3, input, ValueIndexes, ValueIndexes.Count()/2);  
 //IEnumerator<Vector> Tr = tr.GetEnumerator();  
 //IEnumerator<Vector> Output = output.GetEnumerator();  
 //while (Tr.MoveNext() && Output.MoveNext())  
 //{  
 // yield return (Tr.Current, Output.Current);  
 //}  
 //yield break;  
 }  
  
 /// <summary>  
 /// Instatiate Db\_dataset, specify parameters by name of columns. Value column advised to be only 1 string   
 /// </summary>  
 /// <param name="tables"></param>  
 /// <param name="cols"></param>  
 /// <param name="normalise\_cols"></param>  
 /// <param name="relative\_cols"></param>  
 /// <param name="smooth\_cols"></param>  
 /// <param name="value\_cols"></param>  
 /// <param name="safety"></param>  
 /// <param name="smoother\_steps"></param>  
 /// <param name="cond"></param>  
 /// <param name="offset"></param>  
 public DB\_DataSet(List<string> tables, List<string> cols, List<string> normalise\_cols,  
 List<string> relative\_cols,  
 List<string> smooth\_cols,  
 List<string> value\_cols,  
 double safety,  
 int smoother\_steps,  
 string cond = "1=1",  
 int offset = 30)  
 {  
 INTERNAL\_DB\_OFFSET = offset;  
 from\_string = "";  
 foreach (string t in tables)  
 {  
 from\_string += "SELECT \* FROM ";  
 from\_string += t;  
 from\_string += " UNION ";  
 }  
 from\_string = from\_string.Substring(0, from\_string.Length - 6);  
  
  
 cols\_string = from\_string;  
   
 this.tables = tables;  
 this.cols = cols;  
 sql\_query = $"SELECT \* FROM ({from\_string}) WHERE {cond};";//in future work as desriptor, resolved with input map cache  
 this.cols = cols;  
 this.normalise\_cols = normalise\_cols;  
 this.relative\_cols = relative\_cols;  
 this.smooth\_cols = smooth\_cols;  
 this.smoother\_steps = smoother\_steps;  
 ValueColumn = value\_cols;  
 this.safety = safety;  
 normalise\_indexes = new List<int>();  
 relative\_indexes = new List<int>();  
 smooth\_indexes = new List<int>();  
 ValueIndexes = new List<int>();   
  
 Dictionary<int,string> for\_inputmapcache= new Dictionary<int,string>();  
  
 for (int i = 0; i < cols.Count; i++)  
 {  
 string col = cols[i];  
 for\_inputmapcache.Add(i, "");  
 if (normalise\_cols.Contains(col))  
 {  
 normalise\_indexes.Add(i);  
 for\_inputmapcache[i] += "Normalised ";  
 }  
 if (relative\_cols.Contains(col))  
 {  
 relative\_indexes.Add(i);  
 for\_inputmapcache[i] += "Relativised ";  
 }  
 if (smooth\_cols.Contains(col))  
 {  
 smooth\_indexes.Add(i);  
 for\_inputmapcache[i] += "Smoothed ";  
 }  
 if (ValueColumn.Contains(col))  
 {  
 for\_inputmapcache[i] += "Value Column ";  
 ValueIndexes.Add(i);  
 }  
 for\_inputmapcache[i] += $"{cols[i]}";  
 }  
 InputMapCache=new InputMapCache(cols.Count,for\_inputmapcache,normalise\_cols,relative\_cols,value\_cols.First());  
 //InputMapCache.Normalise = normalise\_cols;  
 //InputMapCache.Relative = relative\_cols;  
 //InputMapCache.Value = value\_cols.First();  
 //InputMapCache.Normalise = normalise\_cols;  
 InitialiseSQL();  
 ReCalcTableStart();  
 //this = DB\_DataSet(sql\_query);  
 }  
  
 public void SetInputMapCacheDescription(int i, string description)  
 {  
 InputMapCache.InputDescription[i]= description;  
 }  
  
 public DB\_DataSet(DB\_DataSet\_Context c)  
 {  
 sql\_query = c.sql\_query;  
 from\_string = c.from\_string;  
 cols\_string = c.cols\_string;  
 tables = c.tables;  
 cols=c.cols;  
 normalise\_cols = c.normalise\_cols; normalise\_indexes = c.normalise;  
 relative\_cols=c.relative\_cols; relative\_indexes = c.relative;  
 smooth\_cols=c.smooth\_cols; smooth\_indexes = c.smooth;  
 ValueIndexes = c.ValueIndexes; ValueColumn=c.ValueColumn;  
 smoother\_steps=c.smoother\_steps;  
 INTERNAL\_DB\_OFFSET = c.INTERNAL\_DB\_OFFSET; //used to make sure no NULL values are encountered  
 cache = c.cache;  
 InputMapCache = c.InputMapCache;  
 safety = c.safety;  
 Compound = c.Compound;  
 InitialiseSQL();  
 ReCalcTableStart();  
 }  
  
  
  
 public NeaLibrary.DataStructures.DataSet ToDataSet()  
 {  
 NeaLibrary.DataStructures.DataSet ds = new NeaLibrary.DataStructures.DataSet();  
 foreach ((Vector, Vector) data in Batch(Count(), 0))  
 {  
 ds.Add(data.Item1, data.Item2);  
 }  
 ds.SetInputMapCache(InputMapCache);  
 return ds;  
 }  
  
 public List<(Vector,Vector)> ToList()  
 {  
 List<(Vector,Vector)> l=this.Batch(Count(), 0).ToList();  
 ToListComplete(this, l);  
 return l;  
 }  
  
  
 public void Save(string path)  
 {  
 DB\_DataSet\_Context context = new DB\_DataSet\_Context(  
 sql\_query,  
 from\_string,  
 cols\_string,  
 tables,  
 cols,  
 normalise\_cols, normalise\_indexes,  
 relative\_cols, relative\_indexes,  
 smooth\_cols, smooth\_indexes,  
 ValueColumn,ValueIndexes,  
 safety,  
 smoother\_steps,  
 INTERNAL\_DB\_OFFSET,  
 cache,  
 InputMapCache,  
 Compound  
 );  
 Tools.Tools.Serialize(path, context);  
 }  
 public static IDataSet Load(string path)  
 {  
 //TODO null handling  
 DB\_DataSet\_Context? n = null;  
 FileStream fs = new FileStream(path, FileMode.Open);  
 BinaryFormatter formatter = new BinaryFormatter();  
 #pragma warning disable  
 n = (DB\_DataSet\_Context)formatter.Deserialize(fs);  
 fs.Close();  
 return new DB\_DataSet(n);  
 }  
  
 }  
  
  
  
 [Serializable]  
 public class DB\_DataSet\_Context  
 {  
 public string sql\_query;  
 public string from\_string;  
 public string cols\_string;  
 public List<string> tables;  
 public List<string> cols;  
 public List<string> normalise\_cols; public List<int> normalise;  
 public List<string> relative\_cols; public List<int> relative;  
 public List<string> smooth\_cols; public List<int> smooth;  
 public List<string> ValueColumn; public List<int> ValueIndexes;  
 public double safety;  
 public int smoother\_steps;  
 public int INTERNAL\_DB\_OFFSET = 30; //used to make sure no NULL values are encountered  
 public SortedDictionary<int, (Vector, Vector)> cache = new SortedDictionary<int, (Vector, Vector)>();  
 public InputMapCache InputMapCache;  
 public double Compound;  
  
 public DB\_DataSet\_Context(  
 string sql\_query,  
 string from\_string,  
 string cols\_string,  
 List<string> tables,  
 List<string> cols,  
 List<string> normalise\_cols, List<int> normalise,  
 List<string> relative\_cols, List<int> relative,  
 List<string> smooth\_cols, List<int> smooth,  
 List<string> ValueColumn, List<int> ValueIndexes,  
 double safety,  
 int smoother\_steps,  
 int INTERNAL\_DB\_OFFSET, //used to make sure no NULL values are encountered  
 SortedDictionary<int, (Vector, Vector)> cache,  
 InputMapCache inputMapCache,  
 double Compound  
  
 )  
 {  
 this.sql\_query = sql\_query;  
 this.from\_string = from\_string;  
 this.cols\_string = cols\_string;  
 this.tables = tables;  
 this.cols = cols;  
 this.normalise = normalise;  
 this.normalise\_cols = normalise\_cols;  
 this.relative\_cols = relative\_cols;  
 this.relative = relative;  
 this.smooth = smooth;  
 this.smooth\_cols = smooth\_cols;  
 this.smoother\_steps = smoother\_steps;  
 this.INTERNAL\_DB\_OFFSET = INTERNAL\_DB\_OFFSET;  
 this.cache = cache;  
 this.ValueColumn = ValueColumn;  
 this.ValueIndexes = ValueIndexes;  
 this.safety = safety;  
 this.InputMapCache=inputMapCache;  
 this.Compound = Compound;  
 }  
 }  
}

## .\NeaLibrary\Data/OutputDataProducer.cs

﻿  
using NeaLibrary.DataStructures;  
using System.Collections;  
using System.Reflection;  
using System.Security.Cryptography.X509Certificates;  
  
namespace NeaLibrary.Data  
{  
 public class OutputDataProducer : IEnumerable<Vector>  
 {  
 private double safety = 0.01;  
 private int look\_forward;  
 private int look\_at;  
 private IEnumerable<Vector> input;  
 private int[] ValueColumns;  
 private int maj\_voting;  
 private int CHUNK\_SIZE = 5;  
  
 Dictionary<int, List<double>> valuable\_window = new Dictionary<int, List<double>>();  
 List<(Vector,Vector)> window = new List<(Vector,Vector)>();  
 Vector prev\_bought;  
 Vector profit;  
  
 /// <summary>  
 /// Initiates a dictionary, the key is which entry in the vector it is. and the list represents the previous values of the time series of vectors  
 /// List is more convenient in this case just because it has more function than c# Queue implements  
 /// </summary>  
  
 void InitiateValueWindow()  
 {  
 valuable\_window.Clear();  
 foreach (int i in ValueColumns)  
 {  
 valuable\_window[i] = new List<double>();  
 }  
 }  
  
 /// <summary>  
 /// This implements the required Queue functionality. Items too far back in time series are discarded  
 /// Newest items arrive at the end  
 /// </summary>  
 /// <param name="v"></param>  
 void MoveWindow(Vector v)  
 {  
 window.Add((v,new Vector(1)));  
 window.RemoveAt(0);  
 foreach (int i in ValueColumns)  
 {  
 valuable\_window[i].Add(v[i]);  
 valuable\_window[i].RemoveAt(0);  
 }  
 }  
   
 /// <summary>  
 /// Finds peaks and throughs. then labels the corresponding item with a +1 to mean buy (through)  
 /// -1 to sell (peak)  
 /// 0 neither  
 /// </summary>  
 void FindPeaks()  
 {  
 foreach (int i in ValueColumns)  
 {  
 foreach ((int, double, int) tp in Tools.Tools.ExtractMinimaMaxima(valuable\_window[i]))  
 {  
 if (tp.Item1 == 1)  
 {//peak so sell  
 window[tp.Item3].Item2[0] = -1;  
 if (prev\_bought[0] != 0) { profit += window.Last().Item1 - prev\_bought; }  
 }else if (tp.Item1 == -1)  
 {//buy  
 window[tp.Item3].Item2[0] = 1;  
 prev\_bought = window[tp.Item3].Item1;  
 }  
 else  
 {  
 //if (window[tp.Item3].Item2[0]==0&& window[tp.Item3].Item2[1]==0)  
 //{  
  
 //}  
 //leave as is as the initial value is 0 anyway  
 }  
 }  
 }  
 }  
  
 void FillWindow(IEnumerator<Vector> iter)  
 {  
 prev\_bought = new Vector(input.First().dimension);  
 profit = new Vector(input.First().dimension);  
 if (window.Count==0)  
 {  
 //IEnumerator<Vector> iter = input.GetEnumerator();  
 for (int i = 0; i<CHUNK\_SIZE; i++)  
 {  
 try  
 {  
 iter.MoveNext();  
 window.Add((iter.Current ,new Vector(1)));  
 foreach (int v in ValueColumns)  
 {  
 valuable\_window[v].Add(iter.Current[v]);  
 }  
 }catch { }  
 }  
 }  
 else if (window.Count == CHUNK\_SIZE)  
 {  
 // if the window is already filled do nothing   
 }  
 else  
 {  
 //window is bigger than it should be. this wouldnt happen? so do nothing  
 throw new Exception("Window for OutputDataProducer overfilled, this cant have happened naturally");  
 //Console.WriteLine();  
 }  
 }  
  
 public IEnumerator<Vector> GetEnumerator()   
 {  
 //List<Vector> all\_inputs= new List<Vector>();  
 //List<Vector> all\_outputs= new List<Vector>();  
 //Dictionary<int,List<double>> all\_valuable = new Dictionary<int,List<double>>();  
 //foreach (int i in ValueColumns)  
 //{  
 // all\_valuable.Add(i, new List<double>());  
 //}  
 //foreach (Vector v in input)  
 //{  
 // all\_inputs.Add(v);  
 // foreach (int i in ValueColumns)  
 // {  
 // all\_valuable[i].Add(v[i]);  
 // }  
 //}  
 //foreach (int i in ValueColumns)  
 //{//should run just ONCE  
  
 // foreach ((int, double, int) tp in Tools.Tools.ExtractMinimaMaxima(all\_valuable[i]))  
 // {  
 // Vector ret=new Vector(3);  
 // if (tp.Item1==1)  
 // {  
 // //maximum so sell  
 // ret[0] = 1;  
 // }  
 // else if(tp.Item1==-1){  
 // //minimum so buy  
 // ret[1] = 1;  
 // }  
 // else  
 // {  
 // ret[2] = 1; //hold  
 // }  
 // all\_outputs.Add(ret);  
 // }  
 //}  
 //return all\_outputs.GetEnumerator();  
  
   
  
 IEnumerator<Vector> iter = input.GetEnumerator();  
 InitiateValueWindow();  
 FillWindow(iter);  
 FindPeaks();  
 //foreach ((Vector,Vector) v in window)  
 //{  
  
 // yield return v.Item2;   
 //}  
  
 for (int i=0;i<window.Count-1;i++) //yield all items upto penultimate  
 {  
 yield return window[i].Item2;  
 }  
  
 //budged is there so we can move the window twice. This way we always yield return the penultimate item in the window. This allows us to determine whether it is a max or min  
  
 //case indicates what happens at the end of the time series  
  
 int case\_= 0;  
 //bool budged=false;  
 while (true)  
 {  
 try  
 {   
 iter.MoveNext();  
 MoveWindow(iter.Current);  
 }  
 catch { case\_=1; break; }  
 //try  
 //{  
 // if (!budged)  
 // {  
 // iter.MoveNext();  
 // MoveWindow(iter.Current);  
 // }  
 //}  
 //catch { case\_ = 2;break; }  
  
 FindPeaks(); //determines whether to buy or sell  
  
 yield return window[CHUNK\_SIZE - 1 - 1].Item2; //return penultimate item  
  
  
 //if (!budged)  
 //{  
 // yield return window[CHUNK\_SIZE - 1 - 1].Item2; //second to last  
 // budged = true;  
 //}  
 //else  
 //{  
 // yield return window[CHUNK\_SIZE - 1-1].Item2; //2nd to last but budged????! what this is the same as above.. still return penultimate item, just dont set that it wasw budged  
 //}  
  
 }  
 //if (case\_ == 2)  
 //{  
 // FindPeaks();  
 // yield return window[CHUNK\_SIZE - 1 - 1].Item2; //second to last  
 //}  
 yield return window.Last().Item2; //case 2 can not happen anymore. redundant code. Now we just return last item (going to be 0 for sure since at the end)  
 //Console.WriteLine(profit);  
  
 //Queue<Vector> sliding\_window = new Queue<Vector>(look\_forward);  
 //Queue<Vector> output = new Queue<Vector>(look\_forward);  
  
 //Dictionary<int, List<double>> sliding\_window\_=new Dictionary<int, List<double>>();  
 //foreach (int i in ValueColumns)  
 //{  
 // sliding\_window\_.Add(i,new List<double>());  
 //}  
  
 //Vector profit = new Vector(input.First().dimension);  
 //List<Vector> bought = new List<Vector>();  
 //List<Vector> sold = new List<Vector>();  
 //Vector prev\_bought = new Vector(input.First().dimension);  
 //Vector prev\_sold = new Vector(input.First().dimension);  
 //int countr = 0;  
  
 //foreach (Vector v in input)  
 //{  
 // Vector ret = new Vector(3);  
 // output.Enqueue(ret);  
 // int state = 2;//2  
 // sliding\_window.Enqueue(v);  
  
  
 // if (sliding\_window.Count == look\_forward)  
 // {  
 // Vector avg = sliding\_window.Aggregate((x, y) => x + y) / look\_forward;  
  
 // Vector older = sliding\_window.Dequeue();  
 // Vector newer = sliding\_window.Peek();  
  
 // int voting\_increased = 0;  
 // int voting\_decreased = 0;  
  
  
  
  
 // foreach (int i in ValueColumns)  
 // {  
 // sliding\_window\_[i].Add(v[i]);  
 // if (sliding\_window\_[i].Count>CHUNK\_SIZE) sliding\_window\_[i].RemoveAt(0);  
  
  
  
 // //if (look\_forward>1) {  
 // // if (/\*avg[i] / older[i] > newer[i]/older[i] &&\*/ avg[i] / older[i] > (1+safety) ) //some future period upward gain bigger than 1 day gain?  
 // // {  
 // // voting\_increased++;  
 // // }  
 // // if (/\*avg[i] / older[i] > newer[i]/older[i] &&\*/ avg[i] / older[i] > (1 + safety)) //some future period upward gain bigger than 1 day gain?  
 // // {  
 // // voting\_decreased++;  
 // // }  
 // //}  
 // //else  
 // //{  
  
 // foreach((bool,double,int) vvals in Tools.Tools.ExtractMinimaMaxima(sliding\_window\_[i])){  
 // if (vvals.Item1)  
 // {  
  
 // }  
 // else  
 // {  
  
 // }  
 // }  
  
 // if (/\*newer[i] / older[i]>(1+safety) &&\*/newer[i] / older[i] > 1 && newer[i] > prev\_bought[i] \* (1 + safety)) //if value appreciated  
 // {  
 // voting\_increased++;  
  
 // }  
 // else if (newer[i] / older[i] < (1 - safety / 2) || newer[i] < prev\_bought[i] \* (1 - safety / 2)) //if value depreciated since last bought  
 // {  
 // voting\_decreased++;  
 // }  
 // //}  
 // }  
  
 // if (voting\_increased > maj\_voting && voting\_increased >  
 // voting\_decreased)  
 // {  
 // state = 0; //buy  
 // prev\_bought = newer;  
 // bought.Add(newer);  
 // }  
 // else if (voting\_decreased > maj\_voting || voting\_increased > voting\_decreased)//CHANGE HERE maybe a seprate maj voting for decrease  
 // {  
 // state = 1; //sell  
 // prev\_sold = newer;  
 // List<Vector> soldd = new List<Vector>();  
  
 // foreach (Vector selling\_at in bought)  
 // {  
 // if (selling\_at[1] < newer[1]) { profit += newer - selling\_at; soldd.Add(selling\_at); }  
 // }  
 // foreach (Vector vsold in soldd)  
 // {  
 // bought.Remove(vsold);  
 // }  
 // }  
 // else  
 // {  
 // state = 2; //hold  
 // }  
  
  
 // //if (newer[look\_at] > (older[look\_at]\*(1+safety) ))  
 // //{  
 // // state = 0; //buy  
 // //}else if (newer[look\_at] < (older[look\_at] \* (1 - safety/2)))  
 // //{  
 // // state=2;//sell  
 // //}  
  
 // }  
  
 // ret[state] = 1;  
 // yield return ret;  
  
 // //yield return ret; why twice? accident?  
 //}  
 ////List<Vector> all = input.ToList();  
  
 ////int index = 0;  
  
 ////List<Vector> window = new List<Vector>();  
  
  
  
 //yield break;  
 }  
  
  
  
 IEnumerator IEnumerable.GetEnumerator()  
 {  
 throw new NotImplementedException();  
 }  
 public OutputDataProducer(double safety, int look\_forward,IEnumerable<Vector> input, IEnumerable<int> ValueColumns, int maj\_voting) // input old to new  
 {  
 this.safety = safety;  
 this.look\_forward=look\_forward;  
 this.input = input;  
 this.ValueColumns = ValueColumns.ToArray();  
 this.maj\_voting=maj\_voting;  
 }  
 }  
}

## .\NeaLibrary\Data/SQL.cs

using System;  
using System.Collections.Generic;  
using System.Data;  
using System.Data.SQLite;  
using System.Linq;  
//using System.Numerics;  
using System.Text;  
using System.Threading.Tasks;  
  
using NeaLibrary.Tools;  
using NeaLibrary.DataStructures;  
using System.Xml.Linq;  
using Skender.Stock.Indicators;  
using System.Net;  
  
namespace NeaLibrary.Data  
{  
 public class SQL\_Driver  
 {  
 public SQLiteConnection conn;  
   
 public SQL\_Driver(string source){  
   
 conn = new SQLiteConnection($"Data Source={source};Version=3;", true);  
 // Open the connection:  
 try  
 {  
 conn.Open();  
 //Console.WriteLine($"Opened db {source} connection at {conn}");  
 }  
 catch (Exception ex)  
 {  
 Console.WriteLine(ex.Message);  
 }  
  
  
 }  
  
  
 /// <summary>  
 /// Initiate a connection to the specified data base file.  
 /// </summary>  
 /// <param name="source"></param>  
 /// <returns></returns>  
 public static SQLiteConnection CreateConnection(string source)  
 {  
  
 SQLiteConnection sqlite\_conn;  
 // Create a new database connection:  
 sqlite\_conn = new SQLiteConnection($"Data Source={source};");  
 // Open the connection:  
  
 try  
 {  
 sqlite\_conn.Open();  
 }  
 catch (Exception ex)  
 {  
  
 }  
 return sqlite\_conn;  
 }  
  
 public static void CreateTable(SQLiteConnection conn,string name, Dictionary<string,string> columns)  
 {  
  
 SQLiteCommand sqlite\_cmd;  
   
 string col = "";  
 foreach(KeyValuePair<string,string> entry in columns){  
 col +=entry.Key+" "+ entry.Value +",";  
 }  
 string Createsql = $"CREATE TABLE [IF NOT EXISTS] {name}({col.Substring(0,col.Length-1)})";  
 //string Createsql1 = "CREATE TABLE SampleTable1(Col1 VARCHAR(20), Col2 INT)";  
   
 sqlite\_cmd = conn.CreateCommand();  
 sqlite\_cmd.CommandText = Createsql;  
 sqlite\_cmd.ExecuteNonQuery();  
 }  
  
 /// <summary>  
 /// create a table. specify the DDL art of the CREATE TABLE command. by default uses if not exists  
 /// specify brackets in the ddl  
 ///   
 /// e.g.  
 ///(  
 ///colum1 INTEGER,  
 ///column2 REAL,  
 ///column3 TEXT  
 ///)  
 /// </summary>  
 /// <param name="conn"></param>  
 /// <param name="name"></param>  
 /// <param name="DDL"></param>  
 public static void CreateTable(SQLiteConnection conn,string name, string DDL)  
 {  
  
 SQLiteCommand sqlite\_cmd;  
 string Createsql = $"CREATE TABLE IF NOT EXISTS \'{name}\' {DDL}";  
 //string Createsql1 = "CREATE TABLE SampleTable1(Col1 VARCHAR(20), Col2 INT)";  
   
 sqlite\_cmd = conn.CreateCommand();  
 sqlite\_cmd.CommandText = Createsql;  
 sqlite\_cmd.ExecuteNonQuery();  
 }  
  
 public static void DropTable(SQLiteConnection conn,string name)  
 {  
  
 SQLiteCommand sqlite\_cmd;  
   
 string Createsql = $"DROP TABLE '{name}';";  
 //string Createsql1 = "CREATE TABLE SampleTable1(Col1 VARCHAR(20), Col2 INT)";  
   
 sqlite\_cmd = conn.CreateCommand();  
 sqlite\_cmd.CommandText = Createsql;  
 Console.WriteLine($"dropped {name}, affected {sqlite\_cmd.ExecuteNonQuery()}");  
 }  
 public static void ClearTable(SQLiteConnection conn,string name)  
 {  
  
 SQLiteCommand sqlite\_cmd;  
   
 string Createsql = $"DELETE FROM {name};";  
 //string Createsql1 = "CREATE TABLE SampleTable1(Col1 VARCHAR(20), Col2 INT)";  
   
 sqlite\_cmd = conn.CreateCommand();  
 sqlite\_cmd.CommandText = Createsql;  
 Console.WriteLine($"cleared {name}, affected {sqlite\_cmd.ExecuteNonQuery()}");  
 }  
  
 public static void ExecCommand(SQLiteConnection conn, string com){  
 SQLiteCommand command = conn.CreateCommand();  
 command.CommandText = com;  
 int n = command.ExecuteNonQuery();  
 Console.WriteLine($"Executed: {com}\nAffected {n}");  
 }  
 public static SQLiteDataReader Query(SQLiteConnection conn, string q){  
  
 SQLiteCommand cmd = conn.CreateCommand();  
 cmd.CommandText = q;  
 SQLiteDataReader reader = cmd.ExecuteReader();  
  
 return reader;  
 }  
 public static void InserOnConflictUpdate\_Data(SQLiteConnection conn, string table, string conflict, string on\_conflict,Dictionary<string, string> ins) {  
 string col = Tools.Tools.Join(ins.Keys);  
 string val = Tools.Tools.Join(ins.Values);  
 string com = $"INSERT INTO {table}({col}) VALUES({val}) " +  
 $"ON CONFLICT({conflict}) DO UPDATE SET {on\_conflict};";  
 SQLiteCommand sqlite\_cmd;  
 sqlite\_cmd = conn.CreateCommand();  
 sqlite\_cmd.CommandText = com;  
 sqlite\_cmd.ExecuteNonQuery();  
 }  
  
 /// <summary>  
 /// The key is which column, the value is the data  
 /// </summary>  
 /// <param name="conn"></param>  
 /// <param name="table"></param>  
 /// <param name="ins"></param>  
 public static void InsertData(SQLiteConnection conn,string table ,Dictionary<string,string> ins)  
 {  
 SQLiteCommand sqlite\_cmd;  
 sqlite\_cmd = conn.CreateCommand();  
  
 string col = "";  
 string val="";  
 //foreach(KeyValuePair<string,string> entry in ins){  
 //col +=entry.Key+",";  
 //val +=entry.Value+",";  
 //}  
 col= Tools.Tools.Join(ins.Keys);  
 val= Tools.Tools.Join(ins.Values);  
  
 //Console.WriteLine($"({col}) ({val})");  
 //sqlite\_cmd.CommandText = $"INSERT INTO {table}({col.Substring(0,col.Length-1)}) VALUES ({val.Substring(0,val.Length-1)});";  
 sqlite\_cmd.CommandText = $"INSERT INTO {table} ({col}) VALUES ({val});";  
  
 sqlite\_cmd.ExecuteNonQuery();  
   
  
 /\*  
 sqlite\_cmd.CommandText = "INSERT INTO SampleTable(Col1, Col2) VALUES ('Test1 Text1 ', 2);";  
 sqlite\_cmd.ExecuteNonQuery();  
 sqlite\_cmd.CommandText = "INSERT INTO SampleTable(Col1, Col2) VALUES ('Test2 Text2 ', 3);";  
 sqlite\_cmd.ExecuteNonQuery();  
   
  
 sqlite\_cmd.CommandText = "INSERT INTO SampleTable1(Col1, Col2) VALUES ('Test3 Text3 ', 3);";  
 sqlite\_cmd.ExecuteNonQuery();  
 \*/  
 }  
  
 public static void Update(SQLiteConnection conn, string table, Dictionary<string,string> vals, string condition){  
 string val="";  
 foreach(KeyValuePair<string,string> r in vals){  
 val += r.Key+"="+r.Value+",";  
 }  
 string sql = $"UPDATE \'{table}\' SET {val.Substring(0,val.Length-1)} WHERE {condition};";  
 SQLiteCommand sqlite\_cmd;  
 sqlite\_cmd = conn.CreateCommand();  
 sqlite\_cmd.CommandText = sql;  
 sqlite\_cmd.ExecuteNonQuery();  
 }  
  
  
  
  
 /// <summary>  
 /// If record doesnt exist (based on PK) then Insert,  
 /// Else Update  
 /// </summary>  
 /// <param name="conn"></param>  
 /// <param name="table"></param>  
 /// <param name="ins"></param>  
 public static void Upsert(SQLiteConnection conn, string table, Dictionary<string, string> vals, string conflict)  
 {  
 string update\_string = "";  
 foreach (KeyValuePair<string, string> r in vals)  
 {  
 update\_string += r.Key + "=" + r.Value + ",";  
 }  
 string insert\_col = Tools.Tools.Join(vals.Keys);  
 string insert\_val = Tools.Tools.Join(vals.Values);  
 string sql = $"INSERT INTO {table} ({insert\_col}) VALUES ({insert\_val}) ON CONFLICT({conflict}) DO UPDATE SET {update\_string.Substring(0,update\_string.Length-1)};";  
 SQLiteCommand sqlite\_cmd;  
 sqlite\_cmd = conn.CreateCommand();  
 sqlite\_cmd.CommandText = sql;  
 sqlite\_cmd.ExecuteNonQuery();  
 }  
  
 // public static DataTable ReadData(SQLiteConnection conn,string query)  
 //{  
 // DataTable table=new DataTable();  
 // SqliteDataAdapter adapter = new SqliteDataAdapter(query,conn);  
 // adapter.Fill(table);  
 // return table;  
 //}  
  
 public static IEnumerable<T> ReadColumn<T>(SQLiteConnection conn, string table, string col, string cond="1=1")  
 {  
 using (SQLiteDataReader r = Query(conn, $"SELECT {col} FROM {table} WHERE {cond};"))  
 {  
 while (r.Read())  
 {  
 T value;  
 try  
 {  
 value = r.GetFieldValue<T>(0);  
 }  
 catch (Exception e)  
 {  
 yield break;  
 }  
 yield return value;  
 }  
 }  
 }  
  
 public static IEnumerable<Vector> ReadRow\_AsVector(SQLiteConnection conn, string table, IEnumerable<string> cols, string cond = "1=1", bool DB\_Null\_Ignore=false, int DB\_Null\_Default\_Index=-1)  
 {  
  
 using (SQLiteDataReader r = Query(conn, $"SELECT {string.Join(',',cols)} FROM {table} WHERE {cond};"))  
 {  
 while (r.Read())  
 {  
 Vector v = new Vector(r.FieldCount);  
 int I=-1;  
 try  
 {  
 for (int i=0;i<r.FieldCount;i++)  
 {  
 I = i;  
 double d = r.GetFieldValue<double>(i);  
 v[i] = d;  
 }  
 }  
 catch (InvalidCastException e)  
 {  
 if(!DB\_Null\_Ignore) { yield break; }  
 v[I] = r.GetFieldValue<double>(DB\_Null\_Default\_Index);  
 }  
 catch (Exception e)  
 {  
 Console.WriteLine(e.Message); yield break;  
 }  
 yield return v;  
 }  
 }  
 }  
  
 /// <summary>  
 /// SPECIFY COLUMNS IN THIS ORDER  
 /// DATE  
 /// OPEN  
 /// HIGH  
 /// LOW  
 /// CLOSE  
 /// VOLUME  
 /// </summary>  
 /// <param name="conn"></param>  
 /// <param name="table"></param>  
 /// <param name="cols"></param>  
 /// <returns></returns>  
 public static IEnumerable<Quote> ReadRow\_AsQuote(SQLiteConnection conn, string table, IEnumerable<string> cols, int offset=0)  
 {  
 string c = String.Join(", ", cols);  
 using (SQLiteDataReader r = Query(conn, $"SELECT {c} FROM {table} ORDER BY Date ASC LIMIT -1 OFFSET {offset};"))  
 {  
 while (r.Read())  
 {  
 Quote q = new Quote();  
 try  
 {  
 q.Date = DateTime.Parse(r.GetFieldValue<string>(0));  
 q.Open = Convert.ToDecimal( r.GetFieldValue<double>(1) );  
 q.High = Convert.ToDecimal( r.GetFieldValue<double>(2) );  
 q.Low = Convert.ToDecimal( r.GetFieldValue<double>(3) );  
 q.Close = Convert.ToDecimal( r.GetFieldValue<double>(4) );  
 q.Volume = Convert.ToDecimal( r.GetFieldValue<double>(5) );  
 }  
 catch (InvalidCastException e)  
 {  
 continue;  
 }  
 catch (Exception e)  
 {  
 Console.WriteLine(e.Message); yield break;  
 }  
 yield return q;  
 }  
 }  
 }  
 }  
}

## .\NeaLibrary\Data/TrainingDataTransformer.cs

﻿using System;  
using System.Collections.Generic;  
using System.Linq;  
using NeaLibrary.DataStructures;  
using System.Text;  
using System.Threading.Tasks;  
using System.Collections;  
  
namespace NeaLibrary.Data  
{  
 public class TrainingDataTransformer : IEnumerable<Vector>  
 {  
 IEnumerable<Vector> input;  
 List<int> normalise\_fields;  
 Func<double, double> Normaliser;  
 List<int> smooth\_fields;  
 //int[] smoother\_steps;  
 const int SMOOTHER\_STEPS = 3;  
  
 Func<IEnumerable<double>, double> smoother;  
 List<int> relative;  
 public TrainingDataTransformer(IEnumerable<Vector> input, List<int> relative , List<int> normalise\_fields, Func<double, double> Normaliser, List<int> smooth\_fields, Func<IEnumerable<double>, double> smoother, int smoother\_steps)  
 {  
 this.input = input;  
 this.normalise\_fields = normalise\_fields;  
 this.Normaliser = Normaliser;  
 this.smooth\_fields = smooth\_fields;  
 this.smoother = smoother;  
 this.relative = relative;  
 }  
  
 public IEnumerator<Vector> GetEnumerator()  
 {  
 Vector previous = new Vector(input.First().dimension);  
 int count = 0;  
 Dictionary<int, Queue<double>> for\_smoother = new Dictionary<int, Queue<double>>();  
 foreach (Vector v in input)  
 {  
 Vector r = new Vector(v.dimension);  
   
 for (int n=0;n<v.dimension;n++)  
 {  
 r[n] = v[n]; //default values  
 if (normalise\_fields.Contains(n))  
 {  
 r[n] = Normaliser(v[n]);  
 }  
 if (smooth\_fields.Contains(n))  
 {  
 if(!for\_smoother.ContainsKey(n)) for\_smoother[n] = new Queue<double>();  
 r[n] = smoother(for\_smoother[n]);  
 if (for\_smoother[n].Count >= SMOOTHER\_STEPS) {  
 for\_smoother[n].Dequeue();  
 }  
 for\_smoother[n].Enqueue(v[n]);  
 }  
 if (relative.Contains(n))  
 {  
 if (count != 0)  
 {  
 r[n] = v[n] / previous[n];  
 }  
 else  
 {  
 r[n] = 1; //SENSIBLE DEFAULT, since this is first value there is nothing before to compare to  
 }  
 }  
 }  
 count++;  
 previous = v;  
 yield return r;  
 }  
 yield break;  
 }  
  
 IEnumerator IEnumerable.GetEnumerator()  
 {  
 return GetEnumerator();  
 }  
 }  
}

## .\NeaLibrary\Data\Other/InputMapCache.cs

﻿using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace NeaLibrary.Data.Other  
{  
 [Serializable]  
 public class InputMapCache  
 {  
 public int InputDimension;  
 public Dictionary<int, string> InputDescription;  
 public List<string> Normalise;  
 public List<string> Relative;  
 public string Value;  
 public InputMapCache(int inputDimension, Dictionary<int, string> inputDescription, List<string> normalise, List<string> relative, string value)  
 {  
 InputDimension = inputDimension;  
 InputDescription = inputDescription;  
 Normalise = normalise;  
 Relative = relative;  
 Value = value;  
 }  
 public override string ToString()  
 {  
 string r = "";  
 foreach(KeyValuePair<int,string> kvp in InputDescription)  
 {  
 r += $"{kvp.Key.ToString()} {kvp.Value} ({((Normalise.Contains(kvp.Value))?"Normalised":"")},{((Relative.Contains(kvp.Value)) ? "Relativised" : "")})\n";  
 }return r;  
 }  
 }  
}

## .\NeaLibrary\Data\Other/NNSpecification.cs

﻿using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace NeaLibrary.Data.Other  
{  
 [Serializable]  
 public class NNSpecification  
 {  
 public int InputDimension;  
 public int OutputDimension;  
 public string? LastTrainedOn;  
 public InputMapCache? InputMapCacheDescription;  
 public NNSpecification(int inputDimension, int outputDimension, string? lastTrainedOn, InputMapCache? inputDescription)  
 {  
 InputDimension = inputDimension;  
 OutputDimension = outputDimension;  
 LastTrainedOn = lastTrainedOn;  
 InputMapCacheDescription = inputDescription;  
 }  
 public override string ToString()  
 {  
 return String.Format("Input Dimension {0}\n" +  
 "Output Dimension {1}\n" +  
 "Last Trained On {2}\n" +  
 "InputDescripton {3}\n", InputDimension, OutputDimension, LastTrainedOn, InputMapCacheDescription) ;  
 }  
 }  
}

## .\NeaLibrary\Data\Other/SettingsItem.cs

﻿using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace NeaLibrary.Data.Other  
{  
 public class SettingsItem  
 {  
 public string Value { get; set; }  
 public string TypeName { get; set; }  
 public bool Categorical { get; set; }  
 public object[] Categories { get; set; }  
 public bool PathSelector { get; set; }  
 public string FileExtension { get; set; }  
 public bool IsNumeric { get; set; }  
 public double[] BoundsAndStep { get; set; }  
 public bool IsRegex { get; set; }  
 public string Regex { get; set; }  
  
 public SettingsItem(string value, string type, bool categorical, object[] categories, bool pathSelector, string fileExtension, bool isNumeric, double[] boundsAndStep, bool isRegex, string regex)  
 {  
 Value = value;  
 TypeName = type;//typeof(testc.GetType()).FullName!;  
 Categorical = categorical;  
 Categories = categories;  
 PathSelector = pathSelector;  
 FileExtension = fileExtension;  
 PathSelector=pathSelector;  
 FileExtension=fileExtension;  
 IsNumeric = isNumeric;  
 BoundsAndStep = boundsAndStep;  
 IsRegex = isRegex;  
 Regex = regex;  
 }  
 }  
}

## .\NeaLibrary\Data\Other/SourceParameterInfo.cs

﻿using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace NeaLibrary.Data.Other  
{  
 public class SourceParameterInfo  
 {  
 public string Name { get; set; }  
 public bool IsRegex { get; set; }  
 public string Regex { get; set; }  
 public bool IsCategorical { get; set; }  
 public string[] Categories { get; set; }  
 public bool IsNumeric { get; set; }  
 public double[] BoundsAndStep { get; set; }  
 public SourceParameterInfo() {  
 Name = "";  
 IsRegex = false;  
 Regex = "";  
 IsCategorical = false;  
 Categories = new string[] { };  
 IsNumeric = false;  
 BoundsAndStep = new double[] { };  
 }  
 }  
}

## .\NeaLibrary\DataStructures/BinarySearch.cs

using System;  
  
namespace NeaLibrary.DataStructures  
{  
 public class BinarySearch  
 {  
 static public int binarySearch<T>(T[] arr, T target) where T : IComparable  
 {  
 int l = arr.Length;  
  
 int i = l / 2;  
 try  
 {  
 if (target.CompareTo(arr[i]) == 0) { return i; }  
 else if (target.CompareTo(arr[i]) < 0)  
 {  
 //go left  
 Console.WriteLine($"{target} < {arr[i]} going left");  
 T[] newArr = new T[i];  
 for (int n = 0; n < i; n++)  
 {  
 newArr[n] = arr[n];  
 Console.Write($"{newArr[n]} ");  
 }  
 Console.WriteLine();  
 return binarySearch(newArr, target);  
 }  
 else  
 {  
 //go right  
 Console.WriteLine($"{target} > {arr[i]} going right");  
 T[] newArr = new T[l - i - 1];  
 for (int n = l - 1; n > i; n--)  
 {  
 newArr[n - i - 1] = arr[n];  
 Console.Write($"{newArr[n - i - 1]} ");  
 }  
 Console.WriteLine();  
 int r = binarySearch(newArr, target);  
 return r >= 0 ? i + r + 1 : -1;  
 }  
  
  
 }  
 catch (IndexOutOfRangeException)  
 {  
 return -1;  
 }  
 }  
  
 public static int binarySearchProcedural<T>(T[] arr, T target) where T : IComparable  
 {  
 int left = 0;  
 int right = arr.Length - 1;  
 int i;  
 bool found = false;  
 do  
 {  
 i = (left + right) / 2; //integer division so middle left if odd  
 if (target.CompareTo(arr[i]) == 0) { found = true; return i; }  
 else if (left == right)  
 {  
 return -1;  
 }  
 else if (target.CompareTo(arr[i]) < 0)  
 {  
 //target<arr[i]  
 right = i - 1;  
 }  
 else  
 {  
 //target>arr[i]  
 left = i + 1;  
 }  
  
 } while (!found);  
 return i;  
 }  
 }  
}

## .\NeaLibrary\DataStructures/DataSet.cs

﻿  
  
using NeaLibrary.Data.Other;  
  
using System.Runtime.Serialization.Formatters.Binary;  
using static System.Runtime.InteropServices.JavaScript.JSType;  
  
namespace NeaLibrary.DataStructures  
{  
 [Serializable]  
 public class DataSet : IDataSet  
 {  
 List<Vector> \_data;  
 List<Vector> \_expected;  
 int batch\_counter = 0;  
  
 public InputMapCache InputMapCache;  
 public void SetInputMapCache(InputMapCache inputMapCache)  
 {  
 this.InputMapCache = inputMapCache;  
 }  
 public InputMapCache GetInputMapCache()  
 {  
 return this.InputMapCache;  
 }  
 public DataSet()  
 {  
 \_data = new List<Vector>();  
 \_expected = new List<Vector>();  
 }  
  
 (Vector, Vector) IDataSet.this[int n]  
 {  
 get { return (\_data[n], \_expected[n]); }  
 set { \_data.Add(value.Item1); \_expected.Add(value.Item2); }  
 }  
  
 public void Add(Vector data, Vector expected)  
 {  
 \_data.Add(data);  
 \_expected.Add(expected);  
 }  
  
  
 /// <summary>   
 ///returns iterator of pairs of vectors, where each vector is a sample from the data and the expected value, respectively.  
 /// </summary>  
 /// <param name="size"></param>  
 /// The number of elements in batch.  
 /// <param name="start"></param>  
 /// the index of the first element to include in the batch (default is 0).  
 /// <returns></returns>  
 public IEnumerable<(Vector, Vector)> Batch(int size, int start = 0)  
 {  
 for (int s = start; s < start + size; s++)  
 {  
 yield return (\_data[s], \_expected[s]);  
 }  
 }  
  
 public int Count()  
 {  
 return \_data.Count();  
 }  
  
 public IEnumerator<(Vector, Vector)> GetEnumerator()  
 {  
 for (int s = 0; s < Count(); s++) yield return (\_data[s], \_expected[s]);  
 }  
  
 public IEnumerable<(Vector, Vector)> NextBatch(int size, int start = -1)  
 {  
 if (start != -1)  
 {  
 batch\_counter = start;  
 }  
 return Batch(size, batch\_counter);  
 }  
  
 public void Save(string path)  
 {  
 Tools.Tools.Serialize(path, this);  
 }  
 public static IDataSet Load(string path)  
 {  
 //TODO null handling  
 DataSet? n = null;  
 FileStream fs = new FileStream(path, FileMode.Open);  
 BinaryFormatter formatter = new BinaryFormatter();  
 #pragma warning disable  
 n = (DataSet)formatter.Deserialize(fs);  
 fs.Close();  
 return n;  
 }  
  
 }  
  
 [Serializable]  
 public class ArrayDataSet : IDataSet  
 {  
 public InputMapCache InputMapCache;  
 public void SetInputMapCache(InputMapCache inputMapCache)  
 {  
 this.InputMapCache = inputMapCache;  
 }  
 public InputMapCache GetInputMapCache()  
 {  
 return this.InputMapCache;  
 }  
 Vector[] \_data;  
 Vector[] \_expected;  
 public ArrayDataSet(int size)  
 {  
 \_data = new Vector[size];  
 \_expected = new Vector[size];  
 }  
 int batch\_counter = 0;  
 int count = 0;  
 public void Add((Vector, Vector) val)  
 {  
 if (count < \_data.Length)  
 {  
 \_data[count] = val.Item1;  
 \_expected[count] = val.Item2;  
 count++;  
 }  
 }  
  
 (Vector, Vector) IDataSet.this[int n]  
 {  
 get { return (\_data[n], \_expected[n]); }  
 set { Add(value); }  
 }  
  
 public IEnumerable<(Vector, Vector)> Batch(int size, int start = 0)  
 {  
 for (int s = start; s < start + size; s++)  
 {  
 yield return (\_data[s], \_expected[s]);  
 }  
 }  
  
 public int Count()  
 {  
 return \_data.Count();  
 }  
  
 public IEnumerator<(Vector, Vector)> GetEnumerator()  
 {  
 for (int s = 0; s < Count(); s++) yield return (\_data[s], \_expected[s]);  
 }  
  
 public IEnumerable<(Vector, Vector)> NextBatch(int size, int start = -1)  
 {  
 if (start != -1)  
 {  
 batch\_counter = start;  
 }  
 return Batch(size, batch\_counter);  
 }  
 public void Save(string path)  
 {  
 Tools.Tools.Serialize(path, this);  
 }  
 public static IDataSet Load(string path)  
 {  
 //TODO null handling  
 ArrayDataSet? n = null;  
 FileStream fs = new FileStream(path, FileMode.Open);  
 BinaryFormatter formatter = new BinaryFormatter();  
 #pragma warning disable  
 n = (ArrayDataSet)formatter.Deserialize(fs);  
 fs.Close();  
 return n;  
 }  
 }  
}

## .\NeaLibrary\DataStructures/Graph.cs

using System;  
using NeaLibrary.Tools;  
namespace NeaLibrary.DataStructures  
{  
 [Serializable]  
 public class Graph{  
 public Matrix adjacencyMatrix; //REFER TO MY BOOK  
 public Matrix incidenceMatrix; //NOT IMPLEMETED  
 public Matrix Laplacian; //NOT IMPLEMENTED  
 public Vector nodeStates; // REFER TO MY BOOK  
 public int EdgeCount;  
 public int NodeCount;  
  
 public Dictionary<int, (double,double)> NodesCoordinates = new Dictionary<int, (double, double)>();  
  
 public Vector total\_activation\_accumulation;  
 public Vector total\_sum\_accumulation;  
  
 //public int maxSize;  
 public Vector deltas\_accumulation;  
 public Graph(int size,int maxEdge=-1){  
 if(maxEdge==-1) maxEdge=size\*(size-1); //Simple(not rly as 2 edges per pair) and bidirected there and back so no div by 2  
 //wont actually need this many for NEAT  
  
 //CREATES AN EMPTY GRAPH WITH size NODES  
 //NO CONNECTIONS YET  
 EdgeCount = 0;  
 NodeCount = size;  
 //maxSize = size;  
 adjacencyMatrix = new Matrix(size,size);  
 incidenceMatrix = new Matrix(size,maxEdge);  
 Laplacian = new Matrix(size,size);  
 nodeStates = new Vector(size);  
 total\_activation\_accumulation = new Vector(size);  
  
 }  
 public void AddEdge(int to, int from, double weight, bool directed=true){  
 if((to>NodeCount) | (from>NodeCount)) throw new Exception();  
 //refer too book for next part  
 if(directed){ adjacencyMatrix[to,from] = weight;}  
 else{ adjacencyMatrix[to,from] = weight;adjacencyMatrix[from,to] = weight;}   
 EdgeCount++;  
 }  
 public void SetEdge(int to, int from, double weight, bool directed = true)  
 {  
 if ((to > NodeCount) | (from > NodeCount)) throw new Exception();  
 //refer too book for next part  
 if (directed) { adjacencyMatrix[to, from] = weight; }  
 else { adjacencyMatrix[to, from] = weight; adjacencyMatrix[from, to] = weight; }  
 }  
 public void AddNode(double value=0){  
 //if() throw new Exception();  
  
 adjacencyMatrix = adjacencyMatrix.Upscale(1,1);  
 nodeStates = nodeStates.Upscale(1);  
  
 nodeStates[nodeStates.dimension-1] = value;  
 //maxSize++;  
 NodeCount++;  
 }  
 public void SetNode(int i, double val){  
 if(i>NodeCount) throw new Exception();  
 nodeStates[i] = val;  
 }  
 public void Update(){  
 nodeStates = adjacencyMatrix \* nodeStates;  
 total\_sum\_accumulation += nodeStates;  
 //HERE  
 nodeStates = nodeStates.PutThrough(Tools.Tools.S\_ReLu);  
 // NODE STATES HAVE BEEN ADDINFG  
 }  
 public void resetaccumulations() {  
  
 total\_activation\_accumulation = new Vector(nodeStates.dimension);  
 deltas\_accumulation = new Vector(nodeStates.dimension);  
 total\_sum\_accumulation = new Vector(nodeStates.dimension);  
 }  
  
 public void Forward(Vector input, int cycles)  
 {  
 resetaccumulations();//is fine here too, fine here too since 1 forward then 1 back  
 nodeStates = 0 \* nodeStates; //zero everything  
 for (int i=0;i<input.dimension;i++)  
 {  
 nodeStates[i] = input[i];  
 total\_activation\_accumulation[i] = input[i];  
 }  
 for (int n=0;n<cycles;n++){  
 Update();  
 total\_activation\_accumulation += nodeStates;  
 }  
 //all the accumulations for 1 forward  
 }  
 public void Backward(int node, double error)  
 {  
 Console.WriteLine(node);  
 //works only for acyclic  
  
 //we got the delta cost / delta a \_node  
  
 //see all the ones leading in  
 for (int in\_node=0; in\_node<NodeCount;in\_node++)  
 {  
 if (in\_node==node) { continue; } //self loop for "remembering" shouldnt count  
 if (adjacencyMatrix[node, in\_node]!=0)  
 {  
   
 double weight = adjacencyMatrix[node, in\_node]; // perhaps consider to chnaging to leaky relu?  
 deltas\_accumulation[in\_node] += error \* weight \* Tools.Tools.S\_ReLu\_Derivative(weight \* total\_activation\_accumulation[in\_node]);  
 Backward(in\_node, deltas\_accumulation[in\_node]);  
 }  
 }  
 //all the accumulations for 1 backward  
 }  
  
  
 public void Print(){  
 Console.WriteLine($"Printing graph {this.GetHashCode()}\nNodes: {NodeCount}\nEdges: {EdgeCount}");  
 Console.WriteLine("Adjacency matrix:");  
 adjacencyMatrix.Print();  
 Console.WriteLine("\nNode States:");  
 nodeStates.Print();  
 Console.WriteLine("\n");  
  
 }  
 public void Backprop()  
 {  
  
 }  
  
 public double this[int to,int from]{  
 get{  
 return adjacencyMatrix[to,from];  
 }set{  
 adjacencyMatrix[to,from]=value;  
 }  
 }  
 public double this[int Node]{  
 get{  
 return nodeStates[Node];  
 }set{  
 nodeStates[Node]=value;  
 }  
 }  
  
 }  
}

## .\NeaLibrary\DataStructures/HashMap.cs

using System.Collections.Generic;  
using System.Collections;  
using System.Linq;  
namespace NeaLibrary.DataStructures  
{  
 public class HashMap<TKey, TVal>  
 {  
 /\*  
 C# is confusing with Hashset, Dictionary, Hashtable. So i made my own  
 implementaion. It takes in a TKey and maps it to TVal  
  
 TKey gets hashed and gives position in the array where TVal is  
 if not, then linear probing is used.   
 \*/  
 private int size;  
 public int Size { get { return size; } }  
 public int Count { get { return count; } }  
 private int count=0;  
  
 TVal?[] hashtable;  
 public HashMap(int Size){  
 hashtable = new TVal?[Size];  
 size=Size;  
 count=0;  
 }  
 public bool Add(TKey k, TVal v){  
 if (count<size){  
 int hash=k.GetHashCode();  
 hash = hash%size;  
 hash += size;  
 hash = hash % size;  
 if(!Occupied(hash)){  
 hashtable[hash] = v;  
 }else{  
 //linear probing for a free space  
 int probed = 0;  
 do{  
 hash++;  
 hash = hash%size;  
 if( !Occupied(hash) ){  
 hashtable[hash] = v;  
 }  
 probed++;  
 } while (Occupied(hash) && probed<size);  
 }  
  
 count++;  
 return true;  
  
 }else{  
 //full  
 return false;  
 }  
 }  
 public bool Contains(TKey k, TVal v)  
 {  
 int hash=k.GetHashCode() % size;  
   
 if (hashtable[hash]!=null && hashtable[hash].Equals(v)) //still required as collision's item may have been removed  
 {  
 return true;  
 }  
 else  
 {  
 //linear probing for a free space  
 int probed = 0;  
 do  
 {  
 hash++;  
 hash = hash % size;  
 if (!Occupied(hash))  
 {  
 hashtable[hash] = v;  
 }  
 probed++;  
 } while (Occupied(hash) && probed < size);  
 return false;  
 }  
 }  
 public bool Occupied(TKey key)  
 {  
 int hash = key.GetHashCode();  
 hash = hash % size;  
 return !( hashtable[hash].Equals(default(TVal))) ;  
 }  
 public bool Occupied(int n)  
 {  
 if (hashtable[n]==null) return false;  
 return !(hashtable[n].Equals(default(TVal)));  
 }  
 public void Remove(TKey k, TVal v){  
 int hash=k.GetHashCode();  
 hash = hash%size;  
 if(hashtable[hash].Equals(v)){  
 hashtable[hash] = default;  
 }else{  
 //linear probing for correct entry  
 int probed = 0;  
 do{  
 hash++;  
 hash = hash%size;  
 if(!Occupied(hash)){  
 hashtable[hash] = v;  
 }  
 probed++;  
 }while(Occupied(hash) && probed<size);  
 }  
 }  
  
}  
}

## .\NeaLibrary\DataStructures/IDataSet.cs

﻿using NeaLibrary.Data.Other;  
using System;  
using System.Collections;  
using System.Collections.Generic;  
  
using System.Drawing;  
using System.Linq;  
using System.Reflection.Metadata.Ecma335;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace NeaLibrary.DataStructures  
{  
 public interface IDataSet : IEnumerable<(Vector, Vector)>  
 {  
 ///<summary>just an interface so that we can train on live data or old</summary>  
 public abstract IEnumerable<(Vector, Vector)> Batch(int size, int start = 0);  
 public abstract IEnumerable<(Vector, Vector)> NextBatch(int size, int start = 0);  
 //public abstract IEnumerable<(Vector,Vector)> RandomBatch(int size);  
 public virtual IEnumerable<(Vector, Vector)> RandomBatch(int size)  
 {  
 for (int i = 0; i < size; i++)  
 {  
 int r = Tools.Tools.RandomInt(0, Count() - 1);//since inclusive -1??? CHECK  
 yield return this[r];  
 }  
 }  
 public abstract (Vector,Vector) this[int n]{  
 get;  
 set;  
 }  
  
 public abstract void Save(string location);  
 public static IDataSet Load(string location)  
 {  
 throw new NotImplementedException();  
 }  
  
 IEnumerator IEnumerable.GetEnumerator() { return GetEnumerator(); }  
  
 ///<summary>number of elements, -1 for uknown</summary>  
 public abstract int Count();  
  
 public abstract InputMapCache GetInputMapCache();  
 }  
  
}

## .\NeaLibrary\DataStructures/Matrix.cs

using System;  
using NeaLibrary.Tools;  
namespace NeaLibrary.DataStructures  
{  
 [Serializable]  
 public class Matrix{  
 //private Random rand = new Random();  
 private int \_n;  
 private int \_m;  
  
 public int GetRows() => \_n;  
 public int GetColumns() => \_m;  
  
 private dynamic matrix;  
  
 public Matrix(int n, int m)  
 {//N rows my M columns  
 \_n=n;  
 \_m=m;  
 matrix = new double[n,m];  
 }  
 public double this[int x, int y]{  
 get{  
 return this.matrix[x,y];  
 }  
 set{  
 this.matrix[x,y] = value;  
 }  
 }  
  
  
 public void Print(){  
 for (int row=0; row<this.\_n;row++){  
 for (int col=0; col<this.\_m;col++){  
 string s=String.Format("{0,10:0.000}" , this.matrix[row,col]);  
 if(this.matrix[row,col]==0) {  
 Console.ForegroundColor = ConsoleColor.Gray;   
 }  
 else if(this.matrix[row,col]<0){  
 Console.ForegroundColor = ConsoleColor.Cyan;  
 }else{  
 Console.ForegroundColor = ConsoleColor.Green;  
 }  
 Console.Write($"{s} ");  
 }Console.WriteLine();  
 }Console.ResetColor();Console.WriteLine("\n\n");  
 }  
  
 public Matrix Transpose(){  
 Matrix r = new Matrix(this.\_m,this.\_n); //note n m order switch, for non square matrix  
 for (int row=0; row<this.\_n;row++){  
 for (int col=0; col<this.\_m;col++){  
 r.matrix[col,row]=this.matrix[row,col];//note col row for new, row col for old  
 }  
 }  
 return r;  
 }  
  
 public Matrix Randomise(double maximum, double minimum){  
 Matrix r = new Matrix(this.\_n,this.\_m);  
 for(int n =0; n<this.\_n;n++){  
 for(int m=0; m<this.\_m;m++){  
 this.matrix[n, m] = Tools.Tools.RandomDouble(minimum, maximum);//rand.NextDouble() \* (maximum - minimum) + minimum;//this will just place random number in the range we want  
 }  
 }  
 return(r);  
 }  
 public Matrix Copy\_To(){  
 Matrix r =new Matrix(this.\_n,this.\_m);  
 for (int x = 0; x < this.\_m; x++)  
 {  
 for (int y = 0; y < this.\_n; y++)  
 {  
 r.matrix[y,x]=this.matrix[y,x];  
 }  
 }  
 return r;  
 }  
 public void Copy\_To(Matrix r){  
 if((r.\_n<this.\_n)|(r.\_m<this.\_m)) throw new Exception("Too small");  
 for (int x = 0; x < this.\_m; x++)  
 {  
 for (int y = 0; y < this.\_n; y++)  
 {  
 r.matrix[y,x]=this.matrix[y,x];  
 }  
 }  
 }  
 public Matrix Upscale(int rows, int columns){  
 Matrix temp = new Matrix(this.\_n+rows,this.\_m+columns);  
 this.Copy\_To(temp);  
 return temp;  
 }  
//add an upscale this so no need for new matrix?   
  
 public Matrix Minor(int x,int y){  
 if((this.\_n!=1)&(this.\_m!=1)){  
 Matrix r=new Matrix(this.\_n-1,this.\_m-1);  
 //pull up and/or pull left idea  
 for (int row=0;row<this.\_n;row++){  
 for(int col=0;col<this.\_m;col++){  
 switch ((row>y),(col>x)){  
 case (true,true):  
 //pull up and left  
 r.matrix[row-1,col-1]=this.matrix[row,col];  
 break;  
 case (true,false):  
 //pull up  
 r.matrix[row-1,col]=this.matrix[row,col];  
 break;  
 case (false,true):  
 //pull left  
 r.matrix[row,col-1]=this.matrix[row,col];  
 break;  
 case (false,false):  
 //nothing or vanish  
 if((row!=y)&(col!=x)){  
 r.matrix[row,col]=this.matrix[row,col];}  
 //if they are equal nothing happens  
 break;}  
  
 }  
 }  
 return r;  
 }return this.matrix[0,0];  
 }  
   
 public static Matrix operator \*(Matrix a, Matrix b){  
 if (a.\_m==b.\_n){//dimension check n m n m mid 2 should be same remember Further Math  
 Matrix r = new Matrix(a.\_n,b.\_m);//dimensions of new matrix  
 //loop through each cell and set r's element to sum of elements of a b  
 for (int rrow=0;rrow<r.\_n;rrow++){  
 for(int rcol=0;rcol<r.\_m;rcol++){  
 //remember my working out  
 //rcol is b's col, rrow is a's row  
  
 //go through the a's row (acol++) and pair it with b elems  
 for (int acol=0;acol<a.\_m;acol++){  
 //note, since dimension check at start acol mathces for a and b.  
 r.matrix[rrow,rcol] += a.matrix[rrow, acol]\*b.matrix[acol,rcol];  
 //look at diagram. += adds on stuffs  
  
 }//i think thats it  
  
 }  
 }  
 return r;  
 }else{  
 throw new Exception("Dimension error");  
 }  
 }  
  
 public static Vector operator \* (Matrix a, Vector b)  
 {  
 if(a.\_m!=b.dimension) throw new Exception();  
 Vector r = new Vector(a.\_n);  
 for(int i =0;i<a.\_n;i++){  
 for(int x=0;x<a.\_m;x++){  
 r[i]+=a.matrix[i,x] \* b[x];//BUG WAS HERE. NOW FIXED, FORGOT TO MULTIPLY  
 }  
 }  
 return r;  
   
 }  
  
  
 public static Matrix operator +(Matrix a, Matrix b){  
 if ((a.\_n==b.\_n)&(a.\_m==b.\_m)){  
 Matrix r = new Matrix(a.\_n,a.\_m);  
 //loop through each cell and set r's element to sum of elements of a b  
 for (int row=0; row<a.\_n;row++){  
 for (int col=0; col<a.\_m;col++){  
 r.matrix[row,col]=a.matrix[row,col]+b.matrix[row,col];  
 }   
 }  
 return r;  
 }else{  
 throw new Exception("Dimension error");  
 }  
 }  
  
 public static Matrix operator \*(double a, Matrix b){  
 Matrix r = new Matrix(b.\_n,b.\_m);  
 for(int x=0; x<b.\_n; x++){  
 for (int y = 0; y < b.\_m; y++)  
 {  
 r.matrix[x,y] = b.matrix[x,y]\*a;  
 }  
 }  
 return r;  
 }  
 public static Matrix operator /( Matrix b,double a){  
 Matrix r = new Matrix(b.\_n,b.\_m);  
 for(int x=0; x<b.\_n; x++){  
 for (int y = 0; y < b.\_m; y++)  
 {  
 r.matrix[x,y] = b.matrix[x,y]/a;  
 }  
 }  
 return r;  
 }  
  
  
  
 }  
}

## .\NeaLibrary\DataStructures/Multimap.cs

using System.Collections.Generic;  
using System.Diagnostics.CodeAnalysis;  
  
  
//yes i know there are nullability problems  
public class MultiMap<TKey,TVal>  
{  
  
 Dictionary<TKey, List<TVal>> \_dictionary =  
 new Dictionary<TKey, List<TVal>>();  
   
 public void Add(TKey key, TVal value)  
 {  
 // Add a key.  
 List<TVal> list;  
 if (this.\_dictionary.TryGetValue(key, out list))  
 {  
 list.Add(value);  
 }  
 else  
 {  
 list = new List<TVal>();  
 list.Add(value);  
 this.\_dictionary[key] = list;  
 }  
 }  
 public void Add(TKey key)  
 {  
 // Add a key.  
 List<TVal> list;  
 if (this.\_dictionary.TryGetValue(key, out list))  
 {  
 Console.WriteLine("Already exists key");  
 }  
 else  
 {  
 list = new List<TVal>();  
 this.\_dictionary[key] = list;  
 }  
 }  
  
 public void Clear()  
 {  
 \_dictionary.Clear();  
 }  
   
 public IEnumerable<TKey> Keys  
 {  
 get  
 {  
 return this.\_dictionary.Keys;  
 }  
 }  
   
 public List<TVal> this[TKey key]  
 {  
 get  
 {  
 List<TVal> list;  
 if (!this.\_dictionary.TryGetValue(key, out list))  
 {  
 list = new List<TVal>();  
 this.\_dictionary[key] = list;  
 }  
 return list;  
 }  
 }  
}

## .\NeaLibrary\DataStructures/NonmatrixGraph.cs

namespace NeaLibrary.DataStructures  
{  
  
 public class Node  
 {  
  
 }  
}

## .\NeaLibrary\DataStructures/RandomHashedSet.cs

using System;  
using System.Collections;  
using System.Collections.Generic;  
using System.Linq;  
using NeaLibrary.Tools;  
namespace NeaLibrary.DataStructures  
{  
 [Serializable]  
 public class RandomHashedSet<T>:IEnumerable<T> where T:IComparable<T>{  
  
 Dictionary<T,T> hashset;  
 List<T> values;  
 public int Count;  
  
 public RandomHashedSet(){  
 hashset = new Dictionary<T, T>();  
 values = new List<T>();  
 Count=0;  
 }  
 public bool Contains(T key){  
 return hashset.Keys.Contains(key);  
 }  
  
 public void Add(T key,T val){  
 if(!this.Contains(val)){  
 hashset.Add(key,val);  
 values.Add(val);  
 Count++;  
 }  
 }  
 public void Add(T val){  
 this.Add(val,val);  
 }  
 public void Remove(T val){  
 if(this.Contains(val)){  
 hashset.Remove(val);  
 values.Remove(val);  
 Count--;  
 }  
 }  
 public void Remove(int i){  
 if(i<0) return;  
 if(i>=Count) return;  
 T val = values[i];  
 Remove(val);  
 }  
  
 public T GetValue(T key){  
 return hashset[key];  
 }  
 public T GetValue(int index){  
 return values[index];  
 }  
 public void AddSorted(T val){   
 if(this.Count==0){ Add(val,val);}  
 else{  
 for(int i = 0; i<this.Count; i++){  
 if((val.CompareTo(values[i])<=0)|(i==Count-1)){ // <0  
 values.Insert(i, val);  
 hashset.Add(val,val);  
 Count++;  
 return;  
 }  
 }  
 }  
 }  
  
 public T GetRandom(){  
 if(Count==0) throw new Exception();  
 return values[Tools.Tools.RandomInt(0, values.Count-1)];  
 }  
 public T GetRandomBiased(){  
 if(Count==0) throw new Exception();  
 values.Sort();  
 return values[Tools.Tools.BiasedToStartInt(0, Count)];  
 }  
  
 public int IndexOf(T val){  
 return values.IndexOf(val);  
 }  
 // public T Max(){  
 // //should REALLY (uhhh?) be last  
 // return values.Last<T>();  
 // //POSSIBLE BUG CHECK  
 // }  
 public T this[int i]{  
 get{  
 return values[i];  
 }  
 }  
  
   
 public IEnumerator<T> GetEnumerator(){  
 return values.GetEnumerator();  
 }  
 IEnumerator IEnumerable.GetEnumerator(){  
 return values.GetEnumerator();  
 }  
  
 internal void Clear()  
 {  
 values.Clear();  
 hashset.Clear();  
 Count=0;  
 }  
 }   
}

## .\NeaLibrary\DataStructures/Vector.cs

using System.Collections.Generic;  
using System;  
using System.Linq;  
  
namespace NeaLibrary.DataStructures  
{  
 [Serializable]  
 public class Vector : IEnumerable<double>  
 {  
 public double[] vector {get;set;}  
 public int dimension { get; }  
  
  
 System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()  
 {  
 // call the generic version of the method  
 return this.GetEnumerator();  
 }  
  
 public IEnumerator<double> GetEnumerator()  
 {  
 foreach (double d in vector)  
 yield return d;  
 }  
  
 public Vector(int Dimension){  
 vector = new double[Dimension];  
 dimension = Dimension;  
 }  
 public void Print(){  
 Console.Write("[");  
 //foreach(double d in this.vector){  
 // Console.Write($"{d} ");  
 Console.Write(string.Join(", ", this.AsEnumerable().Select<double,string>(x=>x.ToString()) ));  
 Console.WriteLine("]");  
 }  
  
 public static Vector fromArray(double[] a){  
 Vector r= new Vector(a.Length);  
 for(int i=0;i<a.Length;i++){  
 r.vector[i]=a[i];  
 }return r;  
 }  
 public Vector Upscale(int dimensions){  
 if(dimensions<0) throw new Exception();  
 Vector r = new Vector(this.dimension+dimensions);  
 int c=0;  
 foreach(double d in this){  
 r[c] = this[c];  
 c++; // haha C++  
 }return r;  
 }  
  
 public Vector PutThrough(Func<double,double> a){  
 Vector r = new Vector(this.dimension);  
 for (int i=0;i<this.dimension;i++){  
 r.vector[i] = a(this.vector[i]);  
 }  
 return r;  
 }  
  
 public void Insert\_at\_start\_Same\_Length(double val)  
 {  
 double[] new\_vals = new double[this.dimension];  
 vector.Take<double>(this.dimension -1).ToArray().CopyTo(new\_vals, 1);  
 new\_vals[0] = val;  
 this.vector = new\_vals;  
 } public void Insert\_at\_end\_Same\_Length(double val)  
 {  
 for (int i=0;i<this.dimension-1;i++)  
 {  
 this[i] = this[i + 1];  
 }  
 this[this.dimension - 1] = val;  
 }  
  
 public static double operator \* (Vector a, Vector b){  
 if(a.dimension!=b.dimension) throw new Exception("Wrong dimensions");  
 double r=0;  
 for(int i =0;i<a.dimension;i++){  
 r+=a.vector[i]\*b.vector[i];  
 }  
 return r;  
 }// le dot product  
  
 public static Vector operator ^ (Vector a, Vector b){ //hadamard product  
 if(a.dimension!=b.dimension) throw new Exception("Wrong dimensions");  
 Vector r= new Vector(a.dimension);  
 for(int i =0;i<a.dimension;i++){  
 r.vector[i]+=a.vector[i]\*b.vector[i];  
 }  
 return r;  
 }  
  
 public static Vector operator + (Vector a, Vector b){  
 if(a.dimension!=b.dimension) throw new Exception("Wrong dimensions");  
 Vector r= new Vector(a.dimension);  
 for (int i = 0; i < a.dimension; i++)  
 {  
 r.vector[i] = a.vector[i]+b.vector[i];  
 }  
 return r;  
 }  
 public static Vector operator - (Vector a, Vector b){  
 if(a.dimension!=b.dimension) throw new Exception("Wrong dimensions");  
 Vector r= new Vector(a.dimension);  
 for (int i = 0; i < a.dimension; i++)  
 {  
 r.vector[i] = a.vector[i]-b.vector[i];  
 }  
 return r;  
 }  
 public static Vector operator \*(double a, Vector b){  
 Vector r = new Vector(b.dimension);   
 for(int i=0; i<b.dimension;i++){  
 r.vector[i]=b.vector[i]\*a;  
 }  
 return r;  
 }  
 public static Vector operator /(Vector b, double a){  
 Vector r = new Vector(b.dimension);   
 for(int i=0; i<b.dimension;i++){  
 r.vector[i]=b.vector[i]/a;  
 }  
 return r;  
 }  
  
 public static explicit operator int[](Vector v){  
 //convert vector to it array  
 int[] r = new int[v.dimension];  
 for(int i=0;i<v.dimension;i++){  
 r[i] = (int)v[i];  
 }return r;  
 }  
  
 public double this[int i]{  
 get{  
 return this.vector[i];  
 }set{  
 this.vector[i] = value;  
 }  
 }  
  
 }  
}

## .\NeaLibrary\NeuralNetwork/INeuralNetwork.cs

﻿using NeaLibrary.Data.Other;  
using NeaLibrary.DataStructures;  
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Security.Policy;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace NeaLibrary.NeuralNetwork  
{  
 public interface INeuralNetwork  
 {  
 public abstract NNSpecification GetNNSpecification();  
 public abstract double Train(IDataSet dataset, int iterations);  
 public abstract double Train(IDataSet dataset);  
 public abstract Vector BestPrediction(Vector input);  
  
 public abstract Vector AveragePrediction(Vector input);  
  
 public abstract void Pause();  
 public abstract void Unpause();  
 public abstract void TogglePause();  
 public abstract void Terminate();  
  
 public abstract void Save(string path);  
 public abstract static INeuralNetwork Load(string path);  
  
 //public abstract Graph ToGraph();  
  
 public abstract event EventHandler<(int, double)> NextGeneration;  
  
 public bool IsCategorical { get; set; }  
 public Func<Vector, Vector> CategoryTreshholdFunction { get; set; }  
  
  
 }  
}

## .\NeaLibrary\NeuralNetwork\FFNN/FFNN\_Client.cs

using System;  
using System.Collections.Concurrent;  
using System.Data;  
using System.IO;  
using System.Linq;  
using System.Runtime.Serialization;  
using System.Runtime.Serialization.Formatters.Binary;  
using System.Security.Cryptography;  
using System.Security.Policy;  
using System.Threading;  
using System.Threading.Tasks;  
using NeaLibrary.Data.Other;  
using NeaLibrary.DataStructures;  
  
  
namespace NeaLibrary.NeuralNetwork.FFNN  
{  
 [Serializable]  
 public class FFNN\_Client : INeuralNetwork  
 {  
 public NNSpecification Specification;  
  
 private double learning\_rate;  
 public double GetLearningRate() => learning\_rate;  
 //public double SetLearningRate(double val) { learning\_rate = val; }  
 private Vector topology;  
 public Vector GetTopology() => topology;  
 //public Vector SetTopology(Vector topology) { this.topology = topology; }  
 private object clientsLock = new object();  
 private List<Network> clients;  
 public List<Network> GetClients() => clients;  
 //public List<Network> SetLearningRate(double val) { learning\_rate = val; }  
  
 public bool Dynamic = true;  
  
 private int generations = 1;  
 private double lowest\_error = 1;  
  
 private object PauseLock = new object();  
 private bool Paused = false;  
  
  
 //private object LearningRateLock = new object();  
 object workingLock = new object();  
  
  
 [NonSerialized]  
 bool \_Working = false;  
   
 public bool Working { get => \_Working; set => \_Working = value; }  
  
 //[NonSerialized]  
 private object lowestErrLock = new object();  
  
 private Network? bestFFNN;  
  
 [field: NonSerialized]  
 public event EventHandler<(int, double)> NextGeneration;  
  
  
  
 //ParallelOptions \_ParallelSpecifier = new ParallelOptions();  
 //CancellationTokenSource cts = new CancellationTokenSource();  
  
  
 protected void OnNextGeneration((int, double) e)  
 {  
 EventHandler<(int, double)> handler = NextGeneration;  
 if (handler != null)  
 {  
 handler(this, e);  
 }  
 }  
  
 public void SetLearningRate(double rate)  
 {  
 learning\_rate = rate;  
 lock (clientsLock)  
 {  
 foreach (Network n in clients)  
 {  
 n.SetLearningRate(rate);  
 }  
 }  
 }  
  
 public void Pause()  
 {  
 lock (PauseLock)  
 {  
 Paused = true;  
 }  
 }  
 public void Unpause()  
 {  
 lock (PauseLock)  
 {  
 Paused = false;  
 }  
 }  
 public void TogglePause()  
 {  
 lock (PauseLock)  
 {  
 Paused = !Paused;  
 }  
 }  
 public void Terminate()  
 {  
 Paused = true;  
 //cts.Cancel();  
 }  
 public bool GetPause()  
 {  
 //if (Paused == null) Paused = false ;  
 return Paused;  
 }  
  
 public Graph ToGraph()  
 {  
 int n = (int)topology.Aggregate((sum, v) => { sum += v; return sum; });  
 Graph g = new Graph(n);  
 double BORDER = 0.05;  
 double SPAN = 1 - 2 \* BORDER;  
 double xstep = SPAN / topology.dimension;  
 int node = 0;  
 Dictionary<(int, int), int> nodeLayerMap = new Dictionary<(int, int), int>();  
 for (int layer = 0; layer < topology.dimension; layer++)  
 {  
 double ystep = SPAN / topology[layer];  
 for (int i = 0; i < topology[layer]; i++)  
 {  
 g.NodesCoordinates.Add(node, (xstep \* layer + BORDER, ystep \* i + BORDER));  
 nodeLayerMap.Add((layer, i), node);  
 node++;  
 }  
 }  
 for (int layer = 0; layer < topology.dimension - 1; layer++)  
 {  
 for (int from = 0; from < topology[layer]; from++)  
 {  
 for (int to = 0; to < topology[layer + 1]; to++)  
 {  
 try  
 {  
 g[nodeLayerMap[(layer, from)], nodeLayerMap[(layer + 1, to)]] = clients.First().GetLayers()[layer][to, from]; //layers[0] matches 0 to 1 n m m p => n p cols match first layer, rows match second layer  
 }  
 catch (Exception e)  
 {  
 Console.WriteLine(e);  
 }  
 }  
 }  
 }  
 return g;  
 }  
 [field: NonSerialized]  
 IDataSet? ds;  
  
 //public bool IsCategorical { get => throw new NotImplementedException(); set => throw new NotImplementedException(); }  
 [field: NonSerialized]  
 private Func<Vector, Vector> \_CategoryTreshholdFunction = (Vector x) => x;  
 //[field: NonSerialized]  
 private bool \_IsCategorical = false;  
 public Func<Vector, Vector> CategoryTreshholdFunction { get => \_CategoryTreshholdFunction; set => \_CategoryTreshholdFunction=value; }  
 public bool IsCategorical { get => \_IsCategorical; set => \_IsCategorical=value; }  
 //public Func<Vector, Vector> CategoryTreshholdFunction { get => (x)=>x; set => CategoryTreshholdFunction=value; }  
  
  
 public FFNN\_Client(Vector Topology, double LearningRate, int Clients, IDataSet? ds = null)  
 {  
 learning\_rate = LearningRate;  
 topology = Topology;  
 clients = new List<Network>();  
 for (int i = 0; i < Clients; i++)  
 {  
 clients.Add(new Network(topology));  
 }  
 //this.ds = ds;  
  
 Specification = new NNSpecification((int)topology.First(), (int)topology.Last(), "", ds == null ? null : ds.GetInputMapCache());  
 //\_ParallelSpecifier.CancellationToken = cts.Token;  
 //\_ParallelSpecifier.MaxDegreeOfParallelism = System.Environment.ProcessorCount; //TODO test if this makes things faster  
  
 }  
  
 void UpdateNNSpecification(IDataSet ds)  
 {  
 if (ds == null) return;  
 this.ds = ds;  
 Specification.LastTrainedOn = ds.ToString();  
 Specification.InputMapCacheDescription = ds.GetInputMapCache();  
 }  
  
 //private static void Train\_Worker(Network[] networks, Vector[] ins, Vector[] outs, int repeats)  
 //{  
 // for (int r = 0; r < repeats; r++)  
 // {  
 // for (int i = 0; i < networks.Length; i++)  
 // {  
 // networks[i].Train(ins, outs);  
 // }  
 // }  
 //}  
 //public void TrainAll(double target\_acc = 0.9, int iterations = 100, IDataSet? dataset = null)  
 //{  
 // if (dataset == null&&ds==null) throw new Exception("Provide a dataset");  
 // UpdateNNSpecification(dataset);  
 // generations++;  
 // Parallel.ForEach(clients, x =>  
 // {  
  
 // double er = x.Train(dataset, target\_acc, iterations);  
 // if (er < lowest\_error)  
 // {  
 // lock (lowestErrLock)  
 // {  
 // lowest\_error = er;  
 // bestFFNN = x;  
 // }  
 // }  
 // });  
 // OnNextGeneration((generations, lowest\_error));  
 //}  
  
 /// <summary>  
 /// Uses Parallel.ForEach to train each network  
 /// </summary>  
 /// <param name="backprops\_per\_iteration"></param>  
 /// <param name="dataset"></param>  
 /// <exception cref="Exception"></exception>  
 private void TrainAll\_ParallelForEach(IDataSet dataset, int backprops\_per\_iteration = 1)  
 {  
   
 if (Working) return;  
 lock (workingLock)  
 {  
 Working = true;  
 }  
  
  
 double er;  
  
 if (dataset == null && ds == null) throw new Exception("Provide a dataset");  
 UpdateNNSpecification(dataset);  
 ds = dataset; //comes after the updateNNspecification bc no point updating spec to the same one of same dataset  
 generations++;  
 lowest\_error = 1000000; //no need to lock, dont want strange behaviour so pick big number but not double.MaxValue  
 lock (clientsLock)  
 {  
 Parallel.ForEach(clients, //\_ParallelSpecifier ,  
 x =>  
 {  
 er = x.Train(ds);  
 if (er < lowest\_error)  
 {  
 lock (lowestErrLock)  
 {  
 lowest\_error = er;  
 bestFFNN = x;  
 }  
 }  
 }  
 );  
 }  
 OnNextGeneration((generations, lowest\_error));  
 lock (workingLock)  
 {  
 Working = false;  
 }  
  
 }  
 //public void LinearTrainAll(double target\_acc = 0.9, int iterations = 10, IDataSet? dataset = null)  
 //{  
 // UpdateNNSpecification(dataset);  
 // generations++;  
 // foreach (Network client in clients)  
 // {  
 // double er = client.Train(ds, target\_acc, 1, iterations);  
 // if (er < lowest\_error)  
 // {  
 // lock (lowestErrLock)  
 // {  
 // lowest\_error = er;  
 // bestFFNN = client;  
 // }  
 // }  
 // }  
 // OnNextGeneration((generations, lowest\_error));  
  
 //}  
 //public void LinearTrainAll(IDataSet ds, double target\_acc = 0.9, int iterations = 10, IDataSet? dataset = null)  
  
 //{  
 // UpdateNNSpecification(dataset);  
 // generations++;  
 // foreach (Network client in clients)  
 // {  
 // double er = client.Train(ds, target\_acc, 1, iterations);  
 // if (er < lowest\_error)  
 // {  
 // lock (lowestErrLock)  
 // {  
 // lowest\_error = er;  
 // bestFFNN = client;  
 // }  
 // }  
 // }  
 // OnNextGeneration((generations, lowest\_error));  
 //}  
  
 public Network GetBest()  
 {  
 return clients.First(x => x.GetFitness() == clients.Max(x => x.GetFitness()));  
 }  
  
 //public void Train(IDataSet ds, double target)  
 //{  
  
 //}  
  
 public void Save(string path)  
 {  
 Tools.Tools.Serialize(path, this);  
 }  
 public static FFNN\_Client Load(string path)  
 {  
 FFNN\_Client? n = null;  
 FileStream fs = new FileStream(path, FileMode.Open);  
 BinaryFormatter formatter = new BinaryFormatter();  
#pragma warning disable  
 n = (FFNN\_Client)formatter.Deserialize(fs);  
 fs.Close();  
 return n;  
 }  
 public void SetDataset(IDataSet ds)  
 {  
 this.ds = ds;  
 }  
  
  
 public double Train(IDataSet dataset)  
 {  
 TrainAll\_ParallelForEach(dataset: dataset);  
 return lowest\_error;  
 }  
 public double Train(IDataSet dataset, int iterations)  
 {  
 TrainAll\_ParallelForEach(dataset: dataset, backprops\_per\_iteration:iterations);  
 return lowest\_error;  
 }  
  
 public Vector BestPrediction(Vector input)  
 {  
 if(IsCategorical)return CategoryTreshholdFunction( GetBest().Forward(input));  
 return GetBest().Forward(input);  
 }  
  
 public Vector AveragePrediction(Vector input)  
 {  
 Vector result = new Vector(Specification.OutputDimension);  
 foreach(Network n in clients)  
 {  
 result+=n.Forward(input);  
 }result = (1/clients.Count)\*result;  
 if (IsCategorical) return CategoryTreshholdFunction(result);  
 return result;  
 }  
  
 public NNSpecification GetNNSpecification()  
 {  
 return Specification;  
 }  
  
 static INeuralNetwork INeuralNetwork.Load(string path)  
 {  
 return Load(path);  
 }  
  
 [OnDeserialized]  
 public void onDeserialised(StreamingContext cx)  
 {  
 Working = false;  
 }  
 }  
}

## .\NeaLibrary\NeuralNetwork\FFNN/NeuralNetwork.cs

using NeaLibrary.DataStructures;  
using System.Runtime.Serialization.Formatters.Binary;  
  
namespace NeaLibrary.NeuralNetwork.FFNN  
{  
 [Serializable]  
 public class Network  
 {  
  
  
 private int n\_layers { get; set; } //not count input   
 // weights , bias  
 //List<Matrix> layers; //array is faster than list  
 private double learning\_rate = 0.1;  
 public double GetLearningRate() => learning\_rate;  
 public void SetLearningRate(double v) { learning\_rate = v; }  
 private Matrix[] layers { get; set; }  
 public Matrix[] GetLayers() => layers;  
 private Vector[] biases { get; set; }  
  
 private Vector topology;  
 public Vector GetTopology() => topology;  
 public void SetTopology(Vector topology) { this.topology = topology; }  
 private Vector[] neuron\_values { get; set; }  
 private Vector[] activations { get; set; }  
 private Vector[] deltas { get; set; }  
  
 private Vector[] training\_deltas { get; set; }  
 private Matrix[] training\_weights { get; set; }  
 private Vector[] training\_bias { get; set; }  
  
 private double fitness { get; set; }  
 public double GetFitness() => fitness;  
  
 public Network(Vector topography)  
 { // n of input nodes n of hidden layers ..... n of output  
 if (topography.dimension < 2)  
 {  
 //wrong  
 Console.WriteLine("Wrong use");  
 throw new Exception();  
 }  
  
 topology = topography;  
  
 n\_layers = topography.dimension - 1;  
 layers = new Matrix[n\_layers];  
 biases = new Vector[n\_layers];//les biases  
  
 activations = new Vector[n\_layers];  
 neuron\_values = new Vector[n\_layers];  
 deltas = new Vector[n\_layers];  
  
 training\_deltas = new Vector[n\_layers];  
 training\_weights = new Matrix[n\_layers];  
 training\_bias = new Vector[n\_layers];  
  
 //create the hidden layers and (<=) the out layer  
 for (int n = 1; n <= n\_layers; n++)  
 {  
 //the previous layers n of nodes  
 int prev = (int)topography[n - 1];  
 //Console.Write($"{prev} => ");  
 //this layers n of nodes  
 int thsnodes = (int)topography[n];  
 //Console.WriteLine($"{thsnodes}");  
 //matrix to represent the weights  
 Matrix weights = new Matrix(thsnodes, prev);  
 training\_weights[n - 1] = new Matrix(thsnodes, prev);  
 //randomise weights  
 weights.Randomise(-1, 1);// idk what range to pick  
 //add to the layers List  
 layers[n - 1] = weights; //array starts from 0 innih  
 }  
  
 //vector[] bias initialisation  
 double max = 0.1;  
 double min = -0.1;  
 for (int n = 0; n < n\_layers; n++)  
 {  
 biases[n] = new Vector((int)topography[n + 1]);  
 training\_bias[n] = new Vector((int)topography[n + 1]);  
 training\_deltas[n] = new Vector((int)topography[n + 1]);  
 // layer of biases, now loop through each   
  
 for (int b = 0; b < biases[n].dimension; b++)  
 {  
 biases[n][b] = Tools.Tools.RandomDouble(min, max);//rand.NextDouble() \* (max - min) + min;  
 }  
 }  
 }  
  
 //oof just finished initialising  
  
 /\*public double[] Forward(double[] input){  
 //can use iteration but eh  
  
 //convert input to a matrix  
 Matrix activation = new Matrix(1,input.Length);  
 for (int i=0;i<input.Length;i++){  
 activation.matrix[0,i]=input[i]; //just put all the inpu to activation intermediate  
 }  
  
 //go through the layers  
 for (int l=0;l<this.n\_layers;l++){  
 //Console.WriteLine($"LAYER {l}");  
 activation = activation\*this.layers[l];//just the sums of ac\*weight, no bias or activation func  
 //Console.WriteLine("Before Tanh");  
 //activation.Print();  
 //feed into activation function tanh  
 for(int i=0; i<activation.\_m;i++){  
 activation.matrix[0,i]=Math.Tanh(activation.matrix[0,1] +biases[l].vector[i]);//add bias to this layer then tanh  
 }  
 //Console.WriteLine("After Tanh");  
 //activation.Print();  
 }  
  
 //we gone through layers, convert to double[]  
 double[] r=new double[activation.\_m];  
 for(int i=0;i<activation.\_m;i++){  
 r[i] = activation.matrix[0,i];  
 }return r;//return output  
 }\*/  
  
 /// <summary>  
 /// Pass forward pass from input to an output  
 /// </summary>  
 /// <param name="input"></param>  
 /// <returns>Vector representing the output</returns>  
 public Vector Forward(Vector input)  
 {  
 Vector temp = input; //shallow or no?  
 for (int l = 0; l < n\_layers; l++)  
 {  
 temp = layers[l] \* temp; // left multiply by matrix columns match the current vector rows match next vector  
 // m by n \* n by p  
 temp += biases[l];  
 neuron\_values[l] = temp;  
 temp = temp.PutThrough(Tools.Tools.Tanh);  
 activations[l] = temp;  
 }  
 return temp;  
 }  
 //public Vector ForwardSingleStep(Vector input)  
 //{  
 // Vector temp = input; //shallow or no?  
 // for (int l = 0; l < n\_layers; l++)  
 // {  
 // temp = this.layers[l] \* temp; // left multiply by matrix columns match the current vector rows match next vector  
 // // m by n \* n by p  
 // temp += this.biases[l];  
 // this.neuron\_values[l] = temp;  
 // temp = temp.PutThrough(Tools.Tools.Tanh);  
 // this.activations[l] = temp;  
 // }  
 // return temp;  
 //}  
  
 /// <summary>  
 /// Backpropagation, single pass.  
 /// To be used after a Network.Forward()  
 /// </summary>  
 /// <param name="target">Takes in the label</param>  
 public void Backward(Vector target)  
 {  
  
 //last layer aka output  
 int l = n\_layers - 1;  
  
 Vector deltaL = new Vector((int)topology[l + 1]);  
 Vector nabla\_Cost = new Vector((int)topology[l + 1]);//starts from input, so +1, but 0 index  
 Vector σ\_gradient = new Vector((int)topology[l + 1]);  
 for (int i = 0; i < topology[l + 1]; i++)  
 {  
  
 nabla\_Cost[i] = Tools.Tools.derivative\_MSE(activations[l][i], target[i]); //derivative of cost f for 1 example  
 // ∇ Cₐ = [ partial derivatives ]  
 σ\_gradient[i] = Tools.Tools.Derivative\_Tanh(neuron\_values[l][i]);  
 // σ'(zᴸ)  
 }  
 deltaL = nabla\_Cost ^ σ\_gradient; //hadamard product, more loops than necessary but just go with it  
 // δᴸ = ∇C \* σ'(zᴸ)  
 deltas[l] = deltaL;//want last one. Correct up to this point  
 void recursion(int l, Vector delta)  
 {// l is current, working back to l-1  
 if (l >= 0)  
 {  
 Vector new\_delta = new Vector((int)topology[l + 1]);// l+1 but then -1 for last layer  
 Vector σ\_gradient = new Vector((int)topology[l + 1]);  
 σ\_gradient = neuron\_values[l - 1 + 1].PutThrough(Tools.Tools.Derivative\_Tanh);  
 //for(int i=0;i<topology[l];i++){//new grad vector of activation. of the size l-1 layer  
 //σ\_gradient.vector[i]= tools.Derivative\_Tanh(neuron\_values[l-1].vector[i]);  
 //σ'(zˡ)  
 // }  
 new\_delta = layers[l + 1].Transpose() \* delta ^ σ\_gradient;  
 deltas[l] = new\_delta;  
  
 recursion(l - 1, new\_delta);  
  
 }  
 //done recursion  
 return;  
 }  
  
 recursion(l - 1, deltaL);  
  
  
 }  
  
 private double Rate(Vector input, Vector target)  
 {  
 //RATES ERROR  
 return Tools.Tools.MSE(Forward(input), target);  
 }  
 //public double Rate(Vector[] input, Vector[] target)  
 //{  
 // if (input.Length != target.Length) throw new Exception();  
 // double total = 0;  
 // for (int i = 0; i < input.Length; i++)  
 // {  
 // total += Rate(input[i], target[i]);  
 // }  
 // fitness = 1 - total / input.Length;  
 // return total / input.Length;  
 //}  
  
 public void Improve(Vector input, Vector target)  
 {  
 Forward(input);  
 Backward(target);  
  
  
 void reccur(int l)  
 {  
 if (l >= 0)  
 {  
 //biases dC/dbˡⱼ = δˡⱼ  
  
 biases[l] += -1 \* learning\_rate \* deltas[l];  
 //vectors line up  
  
 if (l != 0)  
 {  
 //use last activations  
 for (int k = 0; k < layers[l].GetRows(); k++)  
 {  
 for (int j = 0; j < layers[l].GetColumns(); j++)  
 {  
 layers[l][k, j] += -1 \* learning\_rate \* deltas[l][k] \* activations[l - 1][j];  
 //dC / dwˡₖⱼ = δˡₖ \* aˡ⁻¹ⱼ   
 }  
 }  
 }  
 else  
 {  
 //use inputs  
 for (int k = 0; k < layers[l].GetRows(); k++)  
 {  
 for (int j = 0; j < layers[l].GetColumns(); j++)  
 {  
 layers[l][k, j] += -1 \* learning\_rate \* deltas[l][k] \* input[j];  
 //dC / dwˡₖⱼ = δˡₖ \* aˡ⁻¹ⱼ   
 }  
 }  
 }  
  
 }  
 return;  
 }  
 reccur(n\_layers - 1);  
 }  
  
 /// <summary>  
 /// Training method, both Forward and Backward on each training cycle  
 ///   
 /// </summary>  
 /// <param name="dataset">IDataSet instance</param>  
 /// <param name="bacth\_size">Average of how many examples to be used to work out error</param>  
 /// <returns></returns>  
 public double Train(IDataSet dataset/\*, int bacth\_size = -1\*/)  
 {  
 /\*if (bacth\_size == -1) \*/int bacth\_size = Int32.Parse(Tools.Tools.GetGlobalVar("Random\_Batch\_Items\_Count"));  
 foreach ((Vector, Vector) tup in dataset.RandomBatch(bacth\_size))  
 {  
 Forward(tup.Item1);  
 Backward(tup.Item2);  
  
 for (int x = 0; x < n\_layers; x++)  
 {  
  
  
 training\_deltas[x] += deltas[x];  
  
 training\_bias[x] += -1 \* learning\_rate \* deltas[x];  
  
  
 //work out required weight changes  
 if (x != 0)  
 {  
 //use last activations  
  
 for (int k = 0; k < layers[x].GetRows(); k++)  
 {  
 for (int j = 0; j < layers[x].GetColumns(); j++)  
 {  
 training\_weights[x][k, j] += -1 \* learning\_rate \* deltas[x][k] \* activations[x - 1][j];  
 //dC / dwˡₖⱼ = δˡₖ \* aˡ⁻¹ⱼ   
 }  
 }  
 }  
 else  
 {  
 //use inputs  
 for (int k = 0; k < layers[x].GetRows(); k++)  
 {  
 for (int j = 0; j < layers[x].GetColumns(); j++)  
 {  
 training\_weights[x][k, j] += -1 \* learning\_rate \* deltas[x][k] \* tup.Item1[j];  
 //dC / dwˡₖⱼ = δˡₖ \* aˡ⁻¹ⱼ not this one, its the inpputs as activation one  
 }  
 }  
 }  
 }// done matrix  
 }  
 for (int i = 0; i < n\_layers; i++)  
 {  
 training\_bias[i] /= dataset.Count();  
 training\_deltas[i] /= dataset.Count();  
 training\_weights[i] /= dataset.Count();  
  
 biases[i] += training\_bias[i];  
 layers[i] += training\_weights[i];  
 }  
 //rate accuracy  
 double error = 0;  
 int n = 1;  
  
 foreach ((Vector, Vector) v in dataset.RandomBatch(bacth\_size)) { error = (error \* (n - 1) + Rate(v.Item1, v.Item2)) / n; n++; }//hmm? Rate vEctor[] func? TODO  
 return error;  
 }  
 /// <summary>  
 /// Training method, both Forward and Backward on each training cycle  
 /// Specify accuracy to reach or maximum number of training iterations  
 /// </summary>  
 /// <param name="ds"></param>  
 /// <param name="target\_acc"></param>  
 /// <param name="iterations\_per\_loop"></param>  
 /// <param name="loops"></param>  
 /// <returns></returns>  
 public double TrainLoop(IDataSet ds, double target\_acc, int iterations\_per\_loop = 1, int loops = 1000)  
 {  
  
 double er = 0;  
 int iters = 0;  
 do  
 {  
 for (int n = 0; n < iterations\_per\_loop; n++)  
 {  
 er = Train(ds, 100); //DS and BatchSize  
  
 }  
 iters++;  
 } while (1 - er < target\_acc & iters < loops);  
 return er;  
 }  
 /// <summary>  
 /// Training method, both Forward and Backward on each training cycle  
 /// </summary>  
 /// <param name="ds">IDataSet instance</param>  
 /// <param name="iterationsperloop"> iterations per training cycle</param>  
 /// <param name="loops">training cycles</param>  
 /// <returns></returns>  
 public double Train(IDataSet ds, int iterationsperloop = 100, int loops = 1000)  
 {  
  
 double er = 0;  
 int iters = 0;  
 do  
 {  
 for (int n = 0; n < iterationsperloop; n++)  
 {  
 er = Train(ds, Int32.Parse(Tools.Tools.GetGlobalVar("Random\_Batch\_Items\_Count")) ); //DS and BatchSize  
  
 }  
 iters++;  
 } while (iters < loops);  
 return er;  
 }  
  
 //public double Train(Vector[] inputs, Vector[] expected)  
 //{  
 // if (inputs.Length != expected.Length) throw new Exception();  
 // int num\_examples = inputs.Length;  
 // for (int i = 0; i < inputs.Length; i++)  
 // {  
  
 // }  
 // // go thru layers updating the arrays, then average arrays  
  
 // Vector[] average = new Vector[n\_layers];  
 // for (int i = 0; i < num\_examples; i++)  
 // {  
 // Forward(inputs[i]);  
 // Backward(expected[i]);  
  
 // for (int x = 0; x < n\_layers; x++)  
 // {  
  
  
 // training\_deltas[x] += deltas[x];  
  
 // training\_bias[x] += -1 \* learning\_rate \* deltas[x];  
  
  
 // //work out required weight changes  
 // if (x != 0)  
 // {  
 // //use last activations  
  
 // for (int k = 0; k < layers[x].\_n; k++)  
 // {  
 // for (int j = 0; j < layers[x].\_m; j++)  
 // {  
 // training\_weights[x].matrix[k, j] += -1 \* learning\_rate \* deltas[x].vector[k] \* activations[x - 1].vector[j];  
 // //dC / dwˡₖⱼ = δˡₖ \* aˡ⁻¹ⱼ   
 // }  
 // }  
 // }  
 // else  
 // {  
 // //use inputs  
 // for (int k = 0; k < layers[x].\_n; k++)  
 // {  
 // for (int j = 0; j < layers[x].\_m; j++)  
 // {  
 // training\_weights[x].matrix[k, j] += -1 \* learning\_rate \* deltas[x].vector[k] \* inputs[x].vector[j]; //inputs[j] was before  
 // //dC / dwˡₖⱼ = δˡₖ \* aˡ⁻¹ⱼ   
 // }  
 // }  
 // }  
 // }// done matrix  
 // }  
  
 // //average all results and change parameters  
 // for (int i = 0; i < n\_layers; i++)  
 // {  
 // training\_bias[i] /= num\_examples;  
 // training\_deltas[i] /= num\_examples;  
 // training\_weights[i] /= num\_examples;  
  
 // biases[i] += training\_bias[i];  
 // layers[i] += training\_weights[i];  
 // }  
 // //rate accuracy  
 // double error = 0;  
 // for (int i = 0; i < num\_examples; i++) error += Rate(inputs[i], expected[i]);//hmm? Rate vEctor[] func?  
 // return error / num\_examples;  
 //}  
 // public async Task<double> AsyncTrain(Vector[] bigInputs, Vector[] bigOutputs,int batchsize){  
 // if(bigInputs.Length!=bigOutputs.Length) throw new Exception();  
 // if(bigInputs.Length<1000) {  
 // //small enough  
 // return Train(bigInputs,bigOutputs);  
 // }else{  
 // // double Worker(Vector[] ins, Vector[] outs){  
 // // return Train(ins,outs);  
 // // }  
 // Vector[][] ins = tools.SplitArrayEvenly(bigInputs,batchsize);  
 // Vector[][] puts = tools.SplitArrayEvenly(bigOutputs,batchsize);  
 // double temp = (bigInputs.Length/batchsize);  
 // temp=Math.Ceiling(temp);  
 // // Thread[] workers = new Thread[(int)temp];  
 // // foreach(Thread worker in workers){  
 // // worker=new Thread(){  
  
 // // }  
 // // }  
 // //TODODO  
  
 // }  
  
 // }  
  
 //Mutate  
 public void Mutate(float chance, float weightval, float biasval)  
 {//chance and max value to mutate by  
 foreach (Matrix layer in layers)  
 {  
 //foreach layer  
 for (int n = 0; n < layer.GetRows(); n++)  
 {  
 for (int m = 0; m < layer.GetColumns(); m++)  
 {  
 //mutate a little  
 if (Tools.Tools.RandomDouble(0, 1) < chance) layer[n, m] += Tools.Tools.RandomDouble(-weightval, weightval);//rand.NextDouble() \* weightval\*2-weightval;  
 }  
 }  
 }  
 //muate bias  
 for (int i = 0; i < biases.Length; i++)  
 {  
 for (int n = 0; n < biases[i].dimension; n++)  
 {  
 if (Tools.Tools.RandomDouble(0, 1) < chance) biases[i][n] += Tools.Tools.RandomDouble(-biasval, biasval);//rand.NextDouble() \* biasval\*2-biasval;  
 }  
 }  
 }  
  
 public Network Crossover(Network other)  
 {  
 //very simple, probably will make things worse  
 Network r = new Network(topology);  
 for (int w = 0; w < layers.Length; w++)  
 {  
 r.layers[w] = (layers[w] + other.layers[w]) / 2;  
 }  
 for (int b = 0; b < biases.Length; b++)  
 {  
 r.biases[b] = (biases[b] + other.biases[b]) / 2;  
 }  
 //simple average of both  
 return r;  
 }  
  
 /\*public async Task ASYNC\_JSON\_Serialise(string path){  
 using FileStream stream = File.Create(path);  
 await JsonSerializer.SerializeAsync(stream, this);  
 await stream.DisposeAsync();  
 }\*/  
 public void Save(string path)  
 {  
 Tools.Tools.Serialize(path, this);  
  
 //File.WriteAllText(path,JsonSerializer.Serialize<Network>(this));  
 }  
#nullable enable  
 public static Network? Load(string path)  
 {  
 Network? n = null;  
 FileStream fs = new FileStream(path, FileMode.Open);  
 BinaryFormatter formatter = new BinaryFormatter();  
  
 // Deserialize the hashtable from the file and  
 // assign the reference to the local variable.  
#pragma warning disable  
 n = (Network)formatter.Deserialize(fs);  
 //string text = System.IO.File.ReadAllText(path);  
 //return JsonSerializer.Deserialize<Network>(text);  
 fs.Close();  
 return n;  
 }  
#nullable disable  
 public void DeepCopy(Network target)  
 {  
 /\*this.layers.CopyTo(target.layers,0);  
 //should copy our layer matrix array  
 //to the target's  
 //is size of array an issue?  
 this.biases.CopyTo\*/  
 //assuming same sizes and topography  
 for (int x = 0; x < layers.Length; x++)  
 {  
 target.layers[x] = layers[x].Copy\_To();  
 }  
 for (int x = 0; x < biases.Length; x++)  
 {  
 for (int y = 0; y < biases[x].dimension; y++)  
 {  
 target.biases[x][y] = biases[x][y];  
 }  
 }  
 }  
 }  
  
}

## .\NeaLibrary\NeuralNetwork\NEAT/calculator.cs

namespace NeaLibrary.NeuralNetwork.NEAT  
{  
  
}

## .\NeaLibrary\NeuralNetwork\NEAT/Gene.cs

using System;  
  
namespace NeaLibrary.NeuralNetwork.NEAT  
{  
 [Serializable]  
 public class Gene  
 {  
 //this isnt to be used for anything  
 //just a container for innovation number  
 public int innovation\_number { get; set; }  
  
 public Gene(int innovation)  
 {  
 innovation\_number = innovation;  
 }  
 public override int GetHashCode()  
 {  
 return innovation\_number;  
 }  
  
 }  
 [Serializable]  
 public class NodeGene : IComparable<NodeGene>  
 {  
 public int innovation\_number { get; set; }  
 public double x { get; set; }  
 public double y { get; set; }  
 Gene gene;  
 public NodeGene(int innovation\_numberr)  
 {  
 gene = new Gene(innovation\_numberr); //do i need a gene class. consider both struct?  
 innovation\_number = innovation\_numberr;  
 }  
  
 public override bool Equals(object? obj)  
 {  
  
 if (obj == null || GetType() != obj.GetType())  
 {  
 return false;  
 }  
  
 NodeGene c = (NodeGene)obj;  
 return c.gene == gene;  
 }  
  
 // override object.GetHashCode  
 public override int GetHashCode()  
 {  
  
 return gene.GetHashCode();  
 }  
 public int CompareTo(NodeGene? b)  
 {  
 if (b == null) return 0; //is this ok  
 return innovation\_number - b.innovation\_number;  
 }  
  
 }  
 [Serializable]  
 public class ConnectionGene : IComparable<ConnectionGene>  
 {  
 public int innovation\_number { get; set; }  
 public NodeGene from { get; }  
 public NodeGene to { get; }  
 public double weight { get; set; }  
 public bool enabled = true;  
  
 public int SplitIndex { get; set; }  
  
 public ConnectionGene(NodeGene From, NodeGene To)  
 {  
 to = To;  
 from = From;  
 SplitIndex = -1;  
 }  
 public ConnectionGene Clone()  
 {  
 ConnectionGene cg = new ConnectionGene(from, to); //BUG WAS HERE, wrong way from and to  
 cg.weight = weight;  
 cg.enabled = enabled;  
 cg.innovation\_number = innovation\_number;  
 return cg;  
 }  
  
 public override bool Equals(object? obj)  
 {  
  
 if (obj == null || GetType() != obj.GetType())  
 {  
 return false;  
 }  
 ConnectionGene c = (ConnectionGene)obj;  
 if (c.to == to && c.from == from) return true;  
 return false;  
 }  
 public override int GetHashCode()  
 {  
 return to.GetHashCode() << 20 + from.GetHashCode();  
 // so like (to node innov n) 00000000 (from node innov n)  
 }  
 public int CompareTo(ConnectionGene? b)  
 {  
 // if less than zero this before other  
  
 if (b == null) return 0; //is this ok  
 return innovation\_number - b.innovation\_number;  
 }  
 }  
}

## .\NeaLibrary\NeuralNetwork\NEAT/Genotype.cs

﻿using System;  
using System.Collections.Generic;  
using System.Linq;  
using NeaLibrary.DataStructures;  
using NeaLibrary.Tools;  
  
  
namespace NeaLibrary.NeuralNetwork.NEAT  
{  
 [Serializable]  
 public class Genotype : IComparable<Genotype>  
 {  
 public Neat master { get; set; }  
 public RandomHashedSet<NodeGene> Nodes { get; set; }  
 public RandomHashedSet<ConnectionGene> Connections { get; set; }  
 //this is a container for all our genes  
 public Graph? Phenotype;  
 public Dictionary<NodeGene, int> NodePhenotypeMap;  
 public double fitness { get; set; }  
 public Species? specie { get; set; }  
 Vector back\_prop\_delta\_avg;  
 Vector activation\_in\_avg;  
  
 public Genotype(Neat m)  
 {//new empty genotype  
 Nodes = new RandomHashedSet<NodeGene>();  
 Connections = new RandomHashedSet<ConnectionGene>();  
 NodePhenotypeMap = new Dictionary<NodeGene, int>();  
 master = m;  
 //Fetch in and out nodes  
 InitialiseInputOutputNodes();  
 //Need to make a phenotype  
 //Phenotype = new Graph(0); //just empty?  
 GeneratePhenotype();  
 }  
 public void AddNode(NodeGene g)  
 {  
 UpdatePhenotype\_AddNode(g);  
 if (Nodes.Contains(g)) return;  
 Nodes.Add(g);  
  
 //update Phenotype  
  
  
  
 //basic sol  
 //Used to update map here but should be handled in UpdatePhentope  
 }  
 public void AddConn(ConnectionGene g)  
 {  
 if (Connections.Contains(g)) return;  
 Connections.Add(g);  
  
 //update Phenotype  
 UpdatePhenotype\_AddEdge(g.from, g.to, g.weight);  
 }  
 public ConnectionGene CloneConn(int index)  
 {  
 return Connections[index].Clone();  
 }  
 //public NodeGene GetNode(int index){  
 // return Nodes[index].Clone();  
 //}  
 public int Highest\_Innov()  
 {  
 if (Nodes.Count == 0) return -1;  
 if (Connections.Count == 0) return Nodes.Max()!.innovation\_number;  
 int c = Connections.Max()!.innovation\_number;  
 //int n = Nodes.MaxBy(x=>x.innovation\_number).innovation\_number;  
 //return ((n>c)? n:c);  
 return c;  
 //Connection has highest innov number because,  
 //when you add a node, you split a connection  
 //and therefore add another connection with a higher innov n  
 }  
 public bool Fitter(Genotype b)  
 {  
  
 return fitness > b.fitness ? true : false;  
  
 }  
  
 public static double Highest\_Fitness(Genotype a, Genotype b)  
 {  
  
 return a.fitness > b.fitness ? a.fitness : b.fitness;  
  
 }  
 public double Distance(Genotype other)  
 {  
 //distance between this andother  
 //we need a to have highest innov  
 Genotype a, b;  
 if (Highest\_Innov() > other.Highest\_Innov())  
 {  
 a = this;  
 b = other;  
 }  
 else  
 {  
 a = other;  
 b = this;  
 }  
 int A = 0;  
 int B = 0;  
 int sim = 0;  
 int dis = 0;  
 int exc = 0;  
 double weight\_dif = 0;  
 while (A < a.Connections.Count && B < b.Connections.Count)  
 {  
 ConnectionGene g1 = a.Connections[A];  
 ConnectionGene g2 = b.Connections[B];  
 //every time we add a new NODE  
 //we make new CONNECTIONS  
 //so by only looking at CONNECTION innov  
 //we know it is the highest  
  
 int in1 = g1.innovation\_number;  
 int in2 = g2.innovation\_number;  
  
 if (in1 == in2)  
 {  
 //matching  
 A++;  
 B++;  
 sim++;  
 weight\_dif += Math.Abs(g1.weight - g2.weight);  
 }  
 else if (in1 > in2)  
 {  
 //disjoint gene of b. b has this extra gene  
 B++;  
 dis++;  
 }  
 else  
 {  
 //disjoint gene of a. a has extra  
 A++;  
 dis++;  
 }  
 }  
 exc = a.Connections.Count - A; //excess as a has bigger innov  
 weight\_dif /= (sim!=0)? sim:1;  
 int n = Math.Max(a.Connections.Count, b.Connections.Count);  
 if (n < 20) n = 1;  
 return master.c1 \* exc / n + master.c2 \* dis / n + master.c3 \* weight\_dif;  
 }  
  
  
 public static Genotype Crossover(Genotype first, Genotype second)  
 {  
 //Crrossover between this andother  
 //we need a to have highest innov  
 Genotype a, b, c;  
  
 if (first.Highest\_Innov() > second.Highest\_Innov())  
 {  
 a = first;  
 b = second;  
 }  
 else  
 {  
 a = second;  
 b = first;  
 }  
 c = a.master.EmptyGenotype();  
 //child  
  
 Neat master = a.master;  
  
 int A = 0;  
 int B = 0;  
  
 void addConnsNodes(ConnectionGene cg)  
 {  
 c.AddNode(cg.to);  
 c.AddNode(cg.from);  
 }//helper function  
 //Changed my mind, will iterate over node genes?  
 //nvm causes key not in dictionary error  
  
 while (A < a.Connections.Count && B < b.Connections.Count)  
 {  
 ConnectionGene g1 = a.Connections[A];  
 ConnectionGene g2 = b.Connections[B];  
 //every time we add a new NODE  
 //we make new CONNECTIONS  
 //so by only looking at CONNECTION innov  
 //we know it is the highest  
  
 //also since we are only looking at connections,  
 //I will need to add the Nodes myself (\*)  
  
 int in1 = g1.innovation\_number;  
 int in2 = g2.innovation\_number;  
  
 if (in1 == in2)  
 {  
 //matching  
  
  
 //we clone the connections genes for the CHILD  
 //as we do nt want it to be the EXACT SAME  
 //reference as parent  
 //WE DO NOT want them to share EXACT same gene instance, but a copy  
 if (Tools.Tools.RandomDouble(0, 1) > 0.5)  
 {  
 ConnectionGene cg = master.GetConnection(g1);  
 addConnsNodes(cg); //this has to be before next line  
 c.AddConn(cg.Clone());  
  
 }  
 else  
 {  
 ConnectionGene cg = master.GetConnection(g2);  
 addConnsNodes(cg);  
 c.AddConn(cg.Clone());  
  
 }  
 A++;  
 B++;  
  
 }  
 else if (in1 > in2)  
 {  
 //disjoint gene of b. b has this extra gene  
  
 if (b.Fitter(a))  
 {  
 ConnectionGene cg = b.CloneConn(B);  
 addConnsNodes(cg);  
 c.AddConn(cg);  
  
 }  
 B++;  
 ////GetConnection(g2).Clone()  
  
 }  
 else  
 {  
 //disjoint gene of a. a has extra  
  
 if (a.Fitter(b))  
 {  
 ConnectionGene cg = a.CloneConn(A);  
 addConnsNodes(cg);  
 c.AddConn(cg);  
  
 }  
 //if(a.Fitter(b)) c.AddConn(master.GetConnection(g1).Clone());  
 A++;  
 }  
 }  
  
 if (a.Fitter(b))  
 {  
 //the excess of a if a is fitter  
 for (int i = A; i < a.Connections.Count; i++)  
 {  
 ConnectionGene cg = a.CloneConn(i);  
 addConnsNodes(cg);  
 c.AddConn(cg);  
 }  
 }  
 // foreach(ConnectionGene cg in c.Connections){  
 // //make sure c has the right node genes  
 // //c.Nodes.Add(cg.to); // SHOULD USE this.AddNode for maintabilityt  
 // //c.Nodes.Add(cg.from);  
 // c.AddNode(cg.to);  
 // c.AddNode(cg.from);  
 // } shpuldnt be necessary anymore  
  
 return c;  
  
  
 }  
  
 public void Mutate()  
 {  
 if (Tools.Tools.RandomDouble() < master.probability\_mutate\_link) MutateLink();  
 if (Tools.Tools.RandomDouble() < master.probability\_mutate\_node) MutateNode();  
 if (Tools.Tools.RandomDouble() < master.probability\_mutate\_weight\_random) WeightRandom();  
 if (Tools.Tools.RandomDouble() < master.probability\_mutate\_weight\_shift) WeightShift();  
 if (Tools.Tools.RandomDouble() < master.probability\_mutate\_link\_toggle) ToggleLink();  
 }  
  
 public void MutateLink()  
 {  
 if (Nodes.Count == 0) { Console.WriteLine("dont be dum"); return; }  
 for (int i = 0; i < 100; i++)  
 {  
 NodeGene a = Nodes.GetRandom();  
 NodeGene b = Nodes.GetRandom();  
 //Console.WriteLine($"Atempt link in {this.GetHashCode()} between {a.innovation\_number} and {b.innovation\_number}");  
  
 if (a.x == b.x)  
 {  
 //Console.WriteLine("Same nodes, illegal, continuing");   
 continue;  
 }  
  
 ConnectionGene c;  
 if (a.x > /\*TODO, which is it > or <\*/ b.x)  
 {//avoid recursion or going backwards  
 c = new ConnectionGene(b, a);  
 //Console.WriteLine($"Made dummy link from {b.innovation\_number} to {a.innovation\_number}");  
 //from b to a  
 }  
 else  
 {  
 c = new ConnectionGene(a, b);  
 //Console.WriteLine($"Made dummy link from {a.innovation\_number} to {b.innovation\_number}");  
 //from a to b  
 }  
 if (Connections.Contains(c))  
 {  
 //if such one exists  
 //Console.WriteLine("Already exists in Genotype, illegal, continue");  
 continue;  
 }  
  
  
 c = master.MakeConnection(c.from, c.to); //we replaced it with one from global neat pool  
 // here /\ /\ /\ Make rather than Get, but make creates a new one anf does all checking  
 // || || ||  
 //Console.WriteLine($"Successfully fetched link from global with number {c.innovation\_number}");  
 c.weight = Tools.Tools.RandomDouble(-master.weight\_random\_strength, master.weight\_random\_strength);  
 //random weight  
 Connections.AddSorted(c);  
  
 //Since we did not call this.AddConn()  
 UpdatePhenotype\_AddEdge(c.from, c.to, c.weight);  
  
 return;  
 }  
  
 }  
 public void MutateNode()  
 {  
 if (Connections.Count == 0)  
 {  
 //Console.WriteLine($"No con to split in {this}");  
 return;  
 }  
 ConnectionGene c = Connections.GetRandom();  
 if (c == null) return;  
 //Console.WriteLine($"Mutating node in {this.GetHashCode()} in connection {c.innovation\_number}");  
  
 NodeGene from = c.from;  
 NodeGene to = c.to;  
  
 NodeGene mid;  
 int replaced = master.GetSplitIndex(from, to);  
 if (replaced == -1)  
 {  
 mid = master.MakeNode();  
 master.SetSplitIndex(from, to, mid.innovation\_number);  
 }  
 else  
 {  
 mid = master.GetNode(replaced);  
 }  
  
 mid.x = (to.x + from.x) / 2;  
 mid.y = (to.y + from.y) / 2;  
  
 ConnectionGene con1 = master.MakeConnection(from, mid);  
 ConnectionGene con2 = master.MakeConnection(mid, to);  
  
 con1.weight = 1;  
 con2.weight = c.weight;  
 con2.enabled = c.enabled;  
  
 c.enabled = false;  
  
 //Console.WriteLine($"Got node, new cons {con1.innovation\_number}, {con2.innovation\_number}\n with number {mid.innovation\_number} having split con {c.innovation\_number}");  
  
  
 Connections.Remove(c);  
 //TODO what should i do about removing connections from phenotype?  
 //set it to 0?  
 UpdatePhenotype\_RemoveEdge(c);  
  
  
 AddNode(mid); //mid is new so safe  
  
 AddConn(con1);  
 AddConn(con2);  
 //used to be Connections.Add  
 //shheeessh  
 }  
 public void WeightShift()  
 {  
  
 if (Connections.Count > 0)  
 {  
 ConnectionGene con = Connections[Tools.Tools.RandomInt(0, Connections.Count - 1)];  
 if (con != null)  
 {  
 con.weight += Tools.Tools.RandomDouble(-master.weight\_shift\_strength, master.weight\_shift\_strength);  
 UpdatePhenotype\_UpdateEdge(con, con.weight);  
 }  
 }  
 }  
 public void WeightRandom()  
 {  
 if (Connections.Count > 0)  
 {  
 ConnectionGene con = Connections[Tools.Tools.RandomInt(0, Connections.Count - 1)];  
 if (con != null)  
 {  
 con.weight = Tools.Tools.RandomDouble(-master.weight\_random\_strength, master.weight\_random\_strength);  
 UpdatePhenotype\_UpdateEdge(con, con.weight);  
 }  
 }  
 }  
 public void ToggleLink()  
 {  
 if (Connections.Count == 0)  
 {  
 //Console.WriteLine("No links to toggle");  
 return;  
 }  
 ConnectionGene con = Connections[Tools.Tools.RandomInt(0, Connections.Count - 1)];  
 if (con != null)  
 {  
 con.enabled = !con.enabled;  
 }  
 }  
  
 public Graph ToGraph()  
 {  
 Graph g = new Graph(Nodes.Count);  
 //ALREADY CREATES NODES  
 //foreach(NodeGene ng in Nodes){  
 // g.AddNode()  
 //}  
 int i = 0;  
  
 if (Nodes.Count != 0)  
 {  
 foreach (NodeGene ng in Nodes)  
 {  
 NodePhenotypeMap[ng] = i;  
 g.NodesCoordinates.Add(i, (ng.x, ng.y));  
 i++;  
 }  
 }  
 if (Connections.Count != 0)  
 {  
 foreach (ConnectionGene cg in Connections)  
 {  
 g.AddEdge(NodePhenotypeMap[cg.to], NodePhenotypeMap[cg.from], cg.enabled ? cg.weight : 0);  
 //will the order of list be important?  
 //set disabled to 0  
 //using nodephenotypemap was before Node.IndexOf(cg.to/from)  
 }  
 }  
  
 return g;  
 }  
  
 public void InitialiseInputOutputNodes()  
 {  
 for (int i = 0; i < master.Input + master.Output; i++)  
 {  
 NodeGene ng = master.GetNode(i);  
 Nodes.Add(ng);  
 NodePhenotypeMap.Add(ng, i);  
 //GeneratePhenotype();  
 //Phenotype.Nodes.Add(i, (ng.x,ng.y));  
 //The crude method rather than AddNode cause phenotype not yet initiated  
 //the index i should be fine as we are looping from start  
 //and filling empty genome with initial genes  
 //input at start and output stacked on, everything else after  
  
  
 //it is the FIRST node. but stored in the adj matrix in index i  
 //so to compensate in the update edge i will make sure to -1  
 //IGNORE /\  
 }  
 }  
  
  
 //whats faster? making a new matrix or upscaling existing one?  
 //I think upscale one would be more efficient. as in to graph method we search for correct node in Nodes  
 public void GeneratePhenotype()  
 {  
 Phenotype = ToGraph();  
 for (int i = 0; i < master.Output; i++)  
 {  
 //Phenotype[master.Input+i,master.Input+i] = 1;  
 //so that it remembers results  
 //set the self loop of output nodes to 1  
  
 //gonna use total sum instead  
 }  
 }  
 public void UpdatePhenotype\_AddNode(NodeGene ng)  
 {  
 if (NodePhenotypeMap.ContainsKey(ng)) return;  
 Phenotype!.AddNode(); //will make new node of index +1  
 NodePhenotypeMap.Add(ng, Phenotype.NodeCount - 1); //so no need +1 here  
 Phenotype.NodesCoordinates.Add(Phenotype.NodeCount - 1, (ng.x, ng.y));  
 // so for eg 5th node. but in array it is in place 4  
 //thus -1  
 //all the meddle with Connections should be handled in Mutate func  
 }  
 public void UpdatePhenotype\_AddEdge(NodeGene From, NodeGene To, double weight)  
 {  
 Phenotype!.AddEdge(NodePhenotypeMap[To], NodePhenotypeMap[From], weight);  
 }  
 public void UpdatePhenotype\_UpdateEdge(ConnectionGene cg, double weight)  
 {  
 Phenotype!.adjacencyMatrix[NodePhenotypeMap[cg.to], NodePhenotypeMap[cg.from]] = weight;  
 //refer to book   
 //TODO the -1?  
 }  
 public void UpdatePhenotype\_RemoveEdge(ConnectionGene cg)  
 {  
 UpdatePhenotype\_UpdateEdge(cg, 0);  
 //since there arent multiple of same edge, we just change the edge in the matrix  
 //nodes arent removed innit?  
 }  
  
 public void Print()  
 {  
 string specieText = specie == null ? "null" : specie.GetHashCode().ToString();  
 Console.WriteLine($"Creature {GetHashCode()} in NEAT {master.GetHashCode()}, specie {specieText}");  
 Console.WriteLine($"Fitness: {fitness}\nComplexity: {Nodes.Count} Nodes, {Connections.Count} Connections");  
 if (Phenotype != null)  
 {  
 Console.WriteLine("==Phenotype==");  
 Phenotype.Print();  
 }  
 }  
  
  
 public Vector Forwards(Vector input)  
 {  
 if (Phenotype == null) GeneratePhenotype();  
 if (input.dimension != master.Input) throw new Exception("Mismatching input dimensions");  
 //for(int i =0;i<input.dimension;i++ ){  
 // Phenotype![i] = input[i]; //maps into first few nodes, should be input nodes  
 //}  
 //for(int i =input.dimension;i<Phenotype.NodeCount;i++ ){  
 // Phenotype![i] = 0; //reset all other nodes  
 //}  
 Phenotype.Forward(input, Nodes.Count - master.Input - master.Output + 1);  
  
 //bool complete=false;  
 //int n\_passes=0;  
 //do{  
 // Phenotype!.Update();  
 // n\_passes++;  
 // continue;  
 //}while((!complete)&(n\_passes<Nodes.Count -master.Input-master.Output+1));  
 Vector r = new Vector(master.Output);  
 for (int i = 0; i < master.Output; i++)  
 {  
 r[i] = Tools.Tools.S\_ReLu(Phenotype.total\_activation\_accumulation[master.Input + i]); //todo  
 //read off the output nodes  
 }  
 return r;  
 }  
 public double RateFitness(Vector input, Vector output)  
 {  
 fitness = 1 - Tools.Tools.MSE(Forwards(input), output);  
 return fitness;  
 }  
  
 public double RateFitness(Vector[] inputs, Vector[] outputs)  
 {  
 if (inputs.Length != outputs.Length) Console.WriteLine("Mismatching training set");  
 double error = 0;  
 for (int i = 0; i < inputs.Length; i++)  
 {  
 error += Tools.Tools.MSE(Forwards(inputs[i]), outputs[i]);  
 }  
 if (inputs.Length == 0) throw new Exception("Give a set with values");  
 fitness = 1 - error / inputs.Length;  
  
 return fitness;  
 }  
 public double RateFitness(IDataSet ds)  
 {  
 double error = 0;  
 foreach ((Vector, Vector) data in ds)  
 {  
 error += Tools.Tools.MSE(Forwards(data.Item1), data.Item2);  
 }  
 if (ds.Count() == -1)  
 {  
 throw new Exception("Uknown length data set");  
 }  
 fitness = 1 - error / ds.Count();  
 return fitness;  
 }  
 public double RateFitness(IEnumerable<(Vector, Vector)> Data)  
 {  
 double error = 0;  
 foreach ((Vector, Vector) data in Data)  
 {  
 error += Tools.Tools.MSE(Forwards(data.Item1), data.Item2);  
 }  
 if (Data.Count() == -1)  
 {  
 throw new Exception("Uknown length data set");  
 }  
 fitness = 1 - error / Data.Count();  
 return fitness;  
 }  
 public (Vector, Vector) Backwards(Vector input, Vector target)  
 {  
 //Phenotype.resetaccumulations();// for this 1 example, this is why there exists the other vectors in this class rather it is done in the forwards method  
 Vector r = Forwards(input);  
  
 Vector nabla\_Cost = new Vector(master.Output);  
 Vector σ\_gradient = new Vector(master.Output);  
 for (int i = 0; i < master.Output; i++)  
 {  
  
 nabla\_Cost[i] = Tools.Tools.derivative\_MSE(r[i], target[i]); //derivative of cost f for 1 example  
 // ∇ Cₐ = [ partial derivatives ]  
 σ\_gradient[i] = Tools.Tools.S\_ReLu\_Derivative(r[i]);  
 // σ'(zᴸ)  
 }  
 Vector deltaL = nabla\_Cost ^ σ\_gradient; //hadamard product, more loops than necessary but just go with it  
  
  
 for (int n = 0; n < master.Output; n++)  
 {  
 Phenotype.Backward(master.Input + n, deltaL[n]); //no need to divide here, since effects from multiple output neurons still increasse a nodes error value  
  
 }  
 Vector delta = Phenotype.deltas\_accumulation;  
 for (int i = 0; i < master.Output; i++)  
 {  
 delta[master.Input + i] = deltaL[i];  
 }  
 return (delta, Phenotype.total\_activation\_accumulation);  
 }  
  
 public void improve(Vector[] ins, Vector[] outs)  
 {  
 back\_prop\_delta\_avg = new Vector(Phenotype.NodeCount);  
 activation\_in\_avg = new Vector(Phenotype.NodeCount);  
  
 for (int n = 0; n < outs.Length; n++)  
 {  
 Vector temp1;  
 Vector temp2;  
 (temp1, temp2) = Backwards(ins[n], outs[n]);//for 1 example  
 back\_prop\_delta\_avg += temp1;//accumulations of examples  
 activation\_in\_avg += temp2;  
 }  
 back\_prop\_delta\_avg = back\_prop\_delta\_avg / outs.Length;  
 activation\_in\_avg = activation\_in\_avg / outs.Length;  
  
 foreach (ConnectionGene cg in Connections)  
 {  
 cg.weight += -1 \* back\_prop\_delta\_avg[NodePhenotypeMap[cg.to]] \* activation\_in\_avg[NodePhenotypeMap[cg.from]];  
 Phenotype.SetEdge(NodePhenotypeMap[cg.to], NodePhenotypeMap[cg.from], cg.weight);  
 }  
  
 }  
 public void improve(IEnumerable<(Vector, Vector)> data)  
 {  
 back\_prop\_delta\_avg = new Vector(Phenotype.NodeCount);  
 activation\_in\_avg = new Vector(Phenotype.NodeCount);  
  
 for (int n = 0; n < data.Count(); n++)  
 {  
 Vector temp1;  
 Vector temp2;  
 (temp1, temp2) = Backwards(data.ElementAt(n).Item1, data.ElementAt(n).Item2);//for 1 example  
 back\_prop\_delta\_avg += temp1;//accumulations of examples  
 activation\_in\_avg += temp2;  
 }  
 back\_prop\_delta\_avg = back\_prop\_delta\_avg / data.Count();  
 activation\_in\_avg = activation\_in\_avg / data.Count();  
  
 foreach (ConnectionGene cg in Connections)  
 {  
 cg.weight += -1 \* back\_prop\_delta\_avg[NodePhenotypeMap[cg.to]] \* activation\_in\_avg[NodePhenotypeMap[cg.from]];  
 Phenotype.SetEdge(NodePhenotypeMap[cg.to], NodePhenotypeMap[cg.from], cg.weight);  
 }  
  
 }  
#nullable enable  
 public int CompareTo(Genotype? obj)  
 {  
 return fitness.CompareTo(obj.fitness);  
 }  
#nullable disable  
 }  
}

## .\NeaLibrary\NeuralNetwork\NEAT/NEAT.cs

using System;  
using NeaLibrary.Tools;  
using NeaLibrary.DataStructures;  
using System.Collections.Concurrent;  
using System.Collections.Generic;  
  
using System.Runtime.Serialization.Formatters.Binary;  
using System.Threading.Tasks;  
using System.IO;  
using System.Linq;  
using System.Runtime.Serialization;  
  
namespace NeaLibrary.NeuralNetwork.NEAT  
{  
  
 //testing saves 234  
 [Serializable]  
 public class Neat  
 {  
  
 public bool PAUSED = false;  
  
 //public int InputDimension, OutputDimension;  
  
 public int Input;  
 public int Output;  
 public int max\_clients;  
 public double best\_acc;  
 int highest\_innovation\_number;  
  
 public int generations = 0;  
  
 //parameters  
 public double weight\_shift\_strength = 0.1;  
 public double weight\_random\_strength = 2;  
 public double probability\_mutate\_link = 0.3;  
 public double probability\_mutate\_node = 0.05;  
 public double probability\_mutate\_weight\_shift = 0.1;  
 public double probability\_mutate\_weight\_random = 0.01;  
 public double probability\_mutate\_link\_toggle = 0.01;  
  
 public double specie\_distance = 5;  
 public double kill\_rate = 0;  
  
 public int revivals = 0;  
  
 public RandomHashedSet<NodeGene> Global\_Nodes;  
 public RandomHashedSet<ConnectionGene> Global\_Connectons;  
 public RandomHashedSet<Genotype> Creatures;  
 public RandomHashedSet<Species> Species;  
  
 //bool punish\_complexity = false;  
  
 public double c1 = 0.4;  
 public double c2 = 0.3;  
 public double c3 = 0.35;  
  
  
 [field:NonSerialized]  
 public SortedSet<Genotype> Best\_Dead = new SortedSet<Genotype>();//in case we need to revive them  
  
  
  
 public Vector motionless\_check = new Vector(5);  
 public Vector accel\_check = new Vector(10);  
 public double motion = 0;  
  
 public object ClientLock = new object(); //lock Creatures, Species and other collections during LifeCycle and operations like GetBest().  
 //Gave an error when calling GetBest and LifeCycle happening at same time  
  
 //public bool pause=false;  
 public Neat(int input, int output, int n\_creatures)  
 {  
 Reset(input, output, n\_creatures);//this is not null, compiler is just wrong  
 }  
  
 public void Reset(int input, int output, int n\_creatures)  
 {  
 Input = input;  
 Output = output;  
 highest\_innovation\_number = -1;  
 //so that first one starts at 0  
 Global\_Connectons = new RandomHashedSet<ConnectionGene>();  
 Global\_Nodes = new RandomHashedSet<NodeGene>();  
 Species = new RandomHashedSet<Species>();  
  
 double BORDER = 0.05;  
 double SPACE = 1 - 2 \* BORDER;  
  
 for (int i = 0; i < input; i++)  
 {  
 NodeGene inNode = MakeNode();  
 inNode.x = BORDER;  
 inNode.y = BORDER + SPACE \* i / input;  
 }  
 for (int o = 0; o < output; o++)  
 {  
 NodeGene outNode = MakeNode();  
 outNode.x = 1 - BORDER;  
 outNode.y = BORDER + SPACE \* o / output;  
 }  
 Creatures = new RandomHashedSet<Genotype>();  
 for (int i = 0; i < n\_creatures; i++)  
 {  
 //Console.WriteLine($"{i}th creature");  
 //Creatures[i] = EmptyGenotype();  
 Genotype instance = EmptyGenotype();  
 Creatures.Add(instance, instance);  
 }  
 max\_clients = n\_creatures;  
 Console.WriteLine($"Reset NEAT instance {GetHashCode()}\n{max\_clients} clients\n{input} => {output}");  
 }  
 public NodeGene MakeNode()  
 {  
 //new node gene  
 NodeGene ng = new NodeGene(highest\_innovation\_number + 1);  
 highest\_innovation\_number++;  
 Global\_Nodes.Add(ng, ng);  
 return ng;  
 }  
 public NodeGene GetNode(int innovation\_number)  
 {  
 //if(innovation\_number>Global\_Nodes.Count) return Global\_Nodes[Global\_Nodes.Count-1];  
 //return Global\_Nodes[innovation\_number]; //TODO is it innovation number her?  
 return Global\_Nodes.First(x => x.innovation\_number == innovation\_number);  
 } //TODO, this is not so efficient  
  
 // public NodeGene MakeNodeFromSplit(ConnectionGene cg){  
 // if(cg.SplitIndex==-1) return MakeNode();  
 // NodeGene ng = new NodeGene(cg.SplitIndex);  
 // return ng;  
 // }  
 public int GetSplitIndex(NodeGene from, NodeGene to)  
 {  
 ConnectionGene cg = new ConnectionGene(from, to);  
 if (Global\_Connectons.Contains(cg))  
 {  
 cg = GetConnection(cg);  
 return cg.SplitIndex;  
 }  
 else  
 {  
 return -1;  
 }  
 }  
 public void SetSplitIndex(NodeGene from, NodeGene to, int val)  
 {  
 ConnectionGene cg = new ConnectionGene(from, to);  
 GetConnection(cg).SplitIndex = val;  
 }  
 public ConnectionGene MakeConnection(NodeGene node1, NodeGene node2)  
 {  
 ///<summary>  
 /// n1 -> n2  
 ///Despite the name Make,  
 ///this method returns the link if suxh exists,  
 ///or makes a new one and stores it in Global\_Connections if not  
 ///</summary>  
 ConnectionGene cg = new ConnectionGene(node1, node2);  
 if (Global\_Connectons.Contains(cg))  
 {  
 ConnectionGene existing = Global\_Connectons.GetValue(cg);  
 cg.innovation\_number = existing.innovation\_number;  
 cg.SplitIndex = existing.SplitIndex;  
 //already acts as clone  
  
 }  
 else  
 {  
 //new conn and add to global  
  
 cg.innovation\_number = highest\_innovation\_number + 1;  
 highest\_innovation\_number++;  
 Global\_Connectons.Add(cg, cg);  
 //Console.WriteLine($"New link globally of number {cg.innovation\_number}");  
 }  
 return cg;  
 }  
 public ConnectionGene GetConnection(ConnectionGene g)  
 {  
 return Global\_Connectons.GetValue(g);  
 }  
  
  
 public Genotype EmptyGenotype()  
 {  
 Genotype g = new Genotype(this);  
 //moved adding inout and output nodes to Genotype  
 return g;  
 }  
 public void Evolve()  
 {  
 foreach (Genotype creature in Creatures)  
 {  
  
 creature.Mutate();  
  
 }  
 //debug  
  
 }  
  
 public void Respeciate()  
 {  
 genspecies();  
 }  
 private void genspecies()  
 {  
 foreach (Species s in Species)  
 {  
 s.reset();  
 }  
 }  
 private void putcreaturesinspecies()  
 {  
 //ConcurrentBag<Genotype> specieless = new ConcurrentBag<Genotype>();  
 //Parallel.ForEach(Creatures, c =>  
 //{  
 // if (c.specie != null) return;  
 // bool found = false;  
 // foreach (Species s in Species)  
 // {  
 // if (s.put(c))  
 // {  
 // found = true;  
 // break;  
 // }  
 // }  
 // if (!found)  
 // {  
 // //Species.Add(new Species(c));  
 // specieless.Add(c); //so its safe with threading  
 // }  
 //}  
 //);  
 foreach (Genotype c in Creatures)  
 {  
 if (c.specie != null) continue;  
 bool found = false;  
 foreach (Species s in Species)  
 {  
 if (s.put(c))  
 {  
 found = true;  
 break;  
 }  
 }  
 if (!found)  
 {  
 Species.Add(new Species(c));  
 //specieless.Add(c); //so its safe with threading  
 }  
 }  
 //if (Species.Count==0) {  
 // Species.Add(new Species(Creatures[0]));  
 //}  
 //There exists a problem, if many genotypes need to go to same species  
  
 //foreach (Genotype c in specieless)  
 //{ //i dont think parallel would suit, writing to dictionary only 1 writer anyway  
 // Species.Add(new Species(c));  
 //}  
 foreach (Species s in Species)  
 {  
 s.take\_average();  
 }  
 }  
 private void remove\_extinct\_species()  
 {  
 for (int i = Species.Count - 1; i >= 0; i--)  
 {  
 if (Species[i].Members.Count <= 1)  
 {  
 Species[i].go\_extinct();  
 Species.Remove(i);  
 }  
 }  
 }  
 private void breed()  
 {  
 int spaces = 0;  
 for (int c = 0; c < Creatures.Count; c++)  
 {  
 if (Creatures[c].specie == null)  
 {  
 spaces++;  
 Creatures.Remove(c);  
 }  
 }  
  
 if (Species.Count == 0)  
 {  
 Species.Add(new Species(Creatures.GetRandom()));  
 return;  
 }  
  
 for (int i = 0; i < spaces; i++)  
 {  
 Creatures.Add(Species.GetRandomBiased().breed());  
 }  
 }  
 public void Kill(double kill = -1)  
 {  
 if (kill == -1) kill = kill\_rate;  
 foreach (Species s in Species)  
 {  
 s.kill(kill);  
 }  
 }  
 public void LifeCycle()  
 {  
 lock (ClientLock)  
 {  
 genspecies();  
 putcreaturesinspecies();  
 Kill();  
 remove\_extinct\_species();  
 putcreaturesinspecies();  
 breed();  
 Evolve();  
 generations++;  
 }  
  
 }  
  
 public void Train(Vector[] ins, Vector[] outs)  
 {  
  
 foreach (Genotype creature in Creatures)  
 {  
 creature.RateFitness(ins, outs);  
 }  
  
 LifeCycle();  
 }  
 public void Train(IDataSet ds)  
 {  
  
 foreach (Genotype creature in Creatures)  
 {  
 creature.RateFitness(ds.RandomBatch(Int32.Parse(Tools.Tools.GetGlobalVar("Random\_Batch\_Items\_Count"))));  
 }  
  
 LifeCycle();  
 }  
 public void Train(IDataSet ds, double target\_acc, int MaxIter = 9999)  
 {  
 int i = 0;  
 double acc = 0;  
 bool c1 = false;  
 bool c2 = false;  
 do  
 {  
 Train(ds);  
 Genotype best\_guy = GetBest();  
 acc = best\_guy.RateFitness(ds.RandomBatch( Int32.Parse(Tools.Tools.GetGlobalVar("Random\_Batch\_Items\_Count")) ));  
 lock (ClientLock)  
 {  
 Best\_Dead.Add(best\_guy);  
 if (Best\_Dead.Count > 10) { Best\_Dead.Remove(Best\_Dead.Min()); }  
 }  
 i++;  
 if (!c1 && acc > target\_acc \* 0.9)  
 {  
 Console.WriteLine("Change 1");  
 c1 = true;  
 probability\_mutate\_link \*= 0.5;  
 probability\_mutate\_node \*= 0.5;  
 probability\_mutate\_weight\_shift \*= 3;  
 specie\_distance \*= 0.8;  
 }  
 if (!c2 && acc > target\_acc \* 0.95)  
 {  
 Console.WriteLine("Change 2");  
 c2 = true;  
 probability\_mutate\_link \*= 0.2;  
 probability\_mutate\_node \*= 0.2;  
 probability\_mutate\_weight\_shift \*= 2;  
 probability\_mutate\_weight\_shift \*= 4;  
 weight\_shift\_strength \*= 0.5;  
 }  
 if (generations > 50)  
 {  
 for (int xx = 0; xx < 10; xx++)  
 {  
 best\_guy.improve(ds.RandomBatch(Int32.Parse(Tools.Tools.GetGlobalVar("Random\_Batch\_Items\_Count"))));  
 double t = best\_guy.RateFitness(ds.RandomBatch(Int32.Parse(Tools.Tools.GetGlobalVar("Random\_Batch\_Items\_Count"))));  
 //Tools.Tools.write\_to\_file(t.ToString(),"log.txt");  
 }  
 }  
 Console.WriteLine($"Best acc {acc}, iteration {i}");  
 best\_acc = acc;  
 motionless\_check.Insert\_at\_start\_Same\_Length(best\_acc);  
 double first\_der = Tools.Tools.first\_deriv\_from\_values(motionless\_check);  
 accel\_check.Insert\_at\_start\_Same\_Length(first\_der);  
 double accel = Tools.Tools.first\_deriv\_from\_values(accel\_check);  
 //if ((motion=first\_der)<0)  
 //{  
 // //revive  
 // foreach(Genotype g in Best\_Dead)  
 // {  
 // Creatures.Add(g);  
 // revivals++;  
 // }//then respeciation happens, these genomes should be at the top  
 //}  
 //if (accel==0)  
 //{  
 // probability\_mutate\_weight\_shift \*= 1.01;  
 // probability\_mutate\_weight\_random \*= 1.005;  
 //}  
  
 } while (i < MaxIter && acc < target\_acc);  
 }  
 public Vector BestPrediction(Vector inp)  
 {  
 lock (ClientLock)  
 {  
 return Creatures.Max().Forwards(inp);  
 }  
 }  
 public Graph GetCreatureAsGraph(int i)  
 {  
 return Creatures[i].ToGraph();  
 }  
 public Genotype GetBest()  
 {  
 lock (ClientLock)  
 {  
 return Creatures.Max();  
 }  
 }  
 public void PrintBest()  
 {  
 Creatures.Max().Print();  
 }  
  
 public void Save(string path)  
 {  
 Tools.Tools.Serialize(path, this);  
 }  
 public static Neat? Load(string path)  
 {  
 Neat? n = null;  
 FileStream fs = new FileStream(path, FileMode.Open);  
 BinaryFormatter formatter = new BinaryFormatter();  
#pragma warning disable  
 n = (Neat)formatter.Deserialize(fs);  
 fs.Close();  
 return n;  
 }  
  
 [OnDeserializing]  
 public void Deserializing(StreamingContext context)  
 {  
 //Console.WriteLine(context);  
 ////debug code  
 //Console.WriteLine(max\_clients);  
 //Console.WriteLine(Global\_Nodes);  
 //Console.WriteLine(Global\_Connectons);  
 //Console.WriteLine(Creatures);  
  
 }  
 [OnDeserialized] public void OnDeserialized(StreamingContext context)  
 {  
 Best\_Dead = new SortedSet<Genotype>();  
 }  
 [OnSerializing] public void OnSerializing(StreamingContext context)  
 {  
  
 }  
  
 }  
}

## .\NeaLibrary\NeuralNetwork\NEAT/NeatHandler.cs

using NeaLibrary.Data;  
using NeaLibrary.Data.Other;  
using NeaLibrary.DataStructures;  
using System;  
using System.CodeDom;  
using System.Reflection.Metadata.Ecma335;  
using System.Runtime.Serialization.Formatters.Binary;  
using System.Threading.Tasks;  
  
namespace NeaLibrary.NeuralNetwork.NEAT  
{  
 [Serializable]  
 public class NEAT\_Handler : INeuralNetwork  
 {  
 public Neat? client;  
   
 public NNSpecification Specification;  
  
 //in case weird GUI glitch and user does change multiple/same value too fast and cause corruption  
 object ParameterLock = new object();  
   
  
 private bool working = false;  
  
 public bool GetWorkingStatus() => working;  
 public void SetWorkingStatus(bool s) => working = s;  
 bool isCategorical;  
  
 bool INeuralNetwork.IsCategorical { get => isCategorical; set => isCategorical = value; }  
  
 [field: NonSerialized]  
 Func<Vector, Vector> categoryThreshholdFunc = (x)=>x.PutThrough(Tools.Tools.SimpleTreshholdFunction);  
  
  
  
 // reference to https://stackoverflow.com/questions/7693391/nonserialized-on-property  
  
 //[: NonSerialized]  
 Func<Vector, Vector> INeuralNetwork.CategoryTreshholdFunction { get => categoryThreshholdFunc; set => categoryThreshholdFunc = value; }  
  
 [field: NonSerialized]  
 public event EventHandler<(int, double)>? NextGeneration;  
  
 [field: NonSerialized]  
 public IDataSet? dataset;  
  
 //public NEAT\_Handler(Neat? Client)  
 //{  
 // client = Client;  
 // Specification=new NNSpecification(Client.Input,Client.Output,null,null);  
 //}  
  
 public NEAT\_Handler(int input, int output, int n\_creatures)  
 {  
 client = new Neat(input,output,n\_creatures);  
 Specification = new NNSpecification(client.Input, client.Output, null, null);  
 }  
  
 /// <summary>  
 /// Actions valid for change parameter method  
 /// </summary>  
 public enum Action  
 {  
 //weight\_shift\_strength,  
 //weight\_random\_strength,  
 probability\_mutate\_link,  
 probability\_mutate\_node,  
 probability\_mutate\_weight\_shift,  
 probability\_mutate\_weight\_random,  
 probability\_mutate\_link\_toggle,  
 specie\_distance,  
 kill\_rate,  
 c1,  
 c2,  
 c3,  
  
 weight\_shift\_strength,  
  
 creature\_count  
  
 //weight\_shift\_strength\_decrease,  
 //weight\_random\_strength\_decrease,  
 //probability\_mutate\_link\_decrease,  
 //probability\_mutate\_node\_decrease,  
 //probability\_mutate\_weight\_shift\_decrease,  
 //probability\_mutate\_weight\_random\_decrease,  
 //probability\_mutate\_link\_toggle\_decrease,  
 //specie\_distance\_decrease,  
 //kill\_rate\_decrease,  
  
 //pause,  
 //start  
 }  
  
  
 /// <summary>  
 /// method to change the parameters using the Action enum  
 /// </summary>  
 /// <param name="action"></param>  
 /// <param name="val"></param>  
 public void ChangeParameter(Action action, double val)  
 {  
 if (client == null) return;  
 lock (ParameterLock) {  
 switch (action)  
 {  
 case Action.probability\_mutate\_weight\_shift:  
 client.weight\_shift\_strength = val;  
 break;  
 case Action.probability\_mutate\_weight\_random:  
 client.probability\_mutate\_weight\_random = val;  
 break;  
 case Action.probability\_mutate\_link:  
 client.probability\_mutate\_link = val;  
 break;  
 case Action.probability\_mutate\_node:  
 client.probability\_mutate\_node = val;  
 break;  
  
 case Action.specie\_distance:  
 client.specie\_distance = val;  
 break;  
 case Action.kill\_rate:  
 client.kill\_rate = val;  
 break;  
  
 case Action.c1:  
 client.c1 = val;  
 break;  
 case Action.c2:  
 client.c2 = val;  
 break;  
 case Action.c3:  
 client.c3 = val;  
 break;  
 case Action.weight\_shift\_strength:  
 client.weight\_shift\_strength = val;  
 break;  
  
 case Action.creature\_count:  
 client.max\_clients = (int)val;  
 break;  
 }  
 }  
  
  
 }  
  
  
  
 /// <summary>  
 /// the private implemetation for the exposed interface method  
 /// </summary>  
 /// <param name="dataset"></param>  
 /// <param name="iters"></param>  
 /// <returns></returns>  
 /// <exception cref="ArgumentException"></exception>  
 /// <exception cref="ArgumentNullException"></exception>  
 /\*async Task<double>\*/private double \_Train(IDataSet dataset/\*double target\_acc,\*/, int iters=int.MaxValue )  
 {  
 if (client == null) throw new Exception("No neural network");  
 working = true;  
 int i = 0;  
 double acc = 0;  
 //bool c1 = false; redundant. experimenting with momentum  
 //bool c2 = false;  
  
 if (dataset != null)  
 {  
 if (dataset.First().Item1.dimension != client.Input || dataset.First().Item2.dimension != client.Output)  
 {  
 throw new ArgumentException("Different dimensionality input or output");  
 }  
 }  
 else  
 {  
 throw new ArgumentNullException("Dataset not given");  
 }  
  
 Specification.InputMapCacheDescription = dataset.GetInputMapCache();  
 Specification.LastTrainedOn = dataset.GetHashCode().ToString();  
  
 do  
 {  
 client.Train(dataset);  
 Genotype best\_guy = client.GetBest();  
 acc = best\_guy.RateFitness(dataset.RandomBatch(100));  
 client.Best\_Dead.Add(best\_guy);  
 if (client.Best\_Dead.Count > 10) { client.Best\_Dead.Remove(client.Best\_Dead.Min()); }  
 i++;  
 //if (!c1 && acc > target\_acc \* 0.9)  
 //{  
 // Console.WriteLine("Change 1");  
 // c1 = true;  
 // client.probability\_mutate\_link \*= 0.5;  
 // client.probability\_mutate\_node \*= 0.5;  
 // client.probability\_mutate\_weight\_shift \*= 3;  
 // client.specie\_distance \*= 0.8;  
 //}  
 //if (!c2 && acc > target\_acc \* 0.95)  
 //{  
 // Console.WriteLine("Change 2");  
 // c2 = true;  
 // client.probability\_mutate\_link \*= 0.2;  
 // client.probability\_mutate\_node \*= 0.2;  
 // client.probability\_mutate\_weight\_shift \*= 2;  
 // client.probability\_mutate\_weight\_shift \*= 4;  
 // client.weight\_shift\_strength \*= 0.5;  
 //}  
 //if (client.generations > 50)  
 //{  
 // for (int xx = 0; xx < 10; xx++)  
 // {  
 // best\_guy.improve(dataset.RandomBatch(100));  
 // double t = best\_guy.RateFitness(dataset.RandomBatch(200));  
 // //Tools.Tools.write\_to\_file(t.ToString(),"log.txt");  
 // }  
 //}  
 Console.WriteLine($"Best acc {acc}, iteration {i}");  
 client.best\_acc = acc;  
 client.motionless\_check.Insert\_at\_start\_Same\_Length(client.best\_acc);  
 double first\_der = Tools.Tools.first\_deriv\_from\_values(client.motionless\_check);  
 client.accel\_check.Insert\_at\_start\_Same\_Length(first\_der);  
 double accel = Tools.Tools.first\_deriv\_from\_values(client.accel\_check);  
  
 OnNextGeneration((client.generations, client.best\_acc));  
  
 //if ((motion=first\_der)<0)  
 //{  
 // //revive  
 // foreach(Genotype g in Best\_Dead)  
 // {  
 // Creatures.Add(g);  
 // revivals++;  
 // }//then respeciation happens, these genomes should be at the top  
 //}  
 //if (accel==0)  
 //{  
 // probability\_mutate\_weight\_shift \*= 1.01;  
 // probability\_mutate\_weight\_random \*= 1.005;  
 //}  
  
 } while (i < iters /\* && acc < target\_acc\*/ && !client.PAUSED);  
 working = false;  
 return acc;  
 }  
  
  
 //public async Task<double> TrainAsync(double target\_acc, int MaxIter = 1000)  
 //{  
  
 //}  
 protected void OnNextGeneration((int, double) e)  
 {  
 EventHandler<(int, double)>? handler = NextGeneration;  
 if (handler != null)  
 {  
 handler(this, e);  
 }  
 }  
  
  
 public int NumberOfCreatures()  
 {  
 if (client == null) throw new Exception("No neural network");  
 return client.Creatures.Count();  
 }  
  
 /// <summary>  
 /// Pauses the training, which escapes the train method and all train methods will terminate. The only way to terminate training method without max iterations specified  
 /// </summary>  
 public void Pause()  
 {  
 if (client == null) throw new Exception("No neural network");  
 client.PAUSED = true;  
   
 }  
 /// <summary>  
 /// Unpause the model, if model wont train when calling Train method, consider calling this first  
 /// </summary>  
 public void Unpause()  
 {  
 if (client == null) throw new Exception("No neural network");  
 client.PAUSED = false;  
 }  
  
 public void TogglePause()  
 {  
 if (client == null) throw new Exception("No neural network");  
 if (client.PAUSED) Unpause();  
 else Pause();  
  
 }  
  
 /// <summary>  
 /// just a wrapper on pause  
 /// </summary>  
 public void Terminate()  
 {  
 Pause();  
 }  
  
 /// <summary>  
 ///   
 /// </summary>  
 /// <returns>Pause status</returns>  
 public bool GetPaused()  
 {  
 if (client == null) throw new Exception("No neural network");  
 return client.PAUSED;  
 }  
  
 public Vector BestPrediction(Vector input)  
 {  
 if (client == null) throw new Exception("No neural network");  
 if(isCategorical) return categoryThreshholdFunc( client.GetBest().Forwards(input));  
 return client.GetBest().Forwards(input);  
 }  
  
 /// <summary>  
 /// take the mean of all the genotypes and return this as output  
 /// </summary>  
 /// <param name="input"></param>  
 /// <returns></returns>  
 public Vector AveragePrediction(Vector input)  
 {  
 if (client == null) throw new Exception("No neural network");  
 Vector sum = new Vector( client.Output);  
 foreach (Genotype g in client.Creatures)  
 {  
 sum += g.Forwards(input);  
 }  
 sum = (1 / client.Creatures.Count())\*sum;  
 return sum;  
 }  
  
 /// <summary>  
 /// Get specification about input/output sizes and more  
 /// </summary>  
 /// <returns>NNSpecification</returns>  
 public NNSpecification GetNNSpecification()  
 {  
 return Specification;  
 }  
  
 /// <summary>  
 /// train for a specified number of generations  
 /// </summary>  
 /// <param name="dataset"></param>  
 /// <param name="iterations"></param>  
 /// <returns></returns>  
 public double Train(IDataSet dataset, int iterations)  
 {  
 return \_Train(dataset, iters:iterations);  
 }  
 /// <summary>  
 /// train without stopping (until paused) or Int32.MaxValue (practical infinity)  
 /// </summary>  
 /// <param name="dataset"></param>  
 /// <returns></returns>  
 public double Train(NeaLibrary.DataStructures.IDataSet dataset)  
 {  
 return \_Train(dataset);  
 }  
  
 public void Save(string path)  
 {  
 if (client == null) throw new ArgumentNullException("No associated NEAT instance");  
 lock (client.ClientLock)  
 {  
 Tools.Tools.Serialize(path, this);  
 }  
 }  
  
 public static INeuralNetwork Load(string path)  
 {  
 NEAT\_Handler? n = null;  
 FileStream fs = new FileStream(path, FileMode.Open);  
 BinaryFormatter formatter = new BinaryFormatter();  
  
 #pragma warning disable  
 n = (NEAT\_Handler)formatter.Deserialize(fs);  
 fs.Close();  
 return n;  
 }  
  
 //public async Task MonitorKeyPresses()  
 // {  
  
 // if (Console.KeyAvailable)  
 // {  
 // var key = Console.ReadKey(intercept: true).Key;  
 // OnKeyPress(key);  
 // }  
  
 // await Task.Delay(10);  
  
 // }  
 // public void OnKeyPress(ConsoleKey key)  
 // {  
 // Target newDirection;  
  
 // switch (key)  
 // {  
 // case ConsoleKey.Q:  
 // newDirection = Target.weight\_shift\_strength\_increase;  
 // client.weight\_shift\_strength\*=1.05;  
 // break;  
  
 // case ConsoleKey.W:  
 // newDirection = Target.weight\_shift\_strength\_decrease;  
 // client.weight\_shift\_strength/=1.05;  
 // break;  
  
 // case ConsoleKey.E:  
 // newDirection = Target.probability\_mutate\_link\_increase;  
 // client.probability\_mutate\_link\*=1.05;  
 // break;  
  
 // case ConsoleKey.R:  
 // newDirection = Target.probability\_mutate\_link\_decrease;  
 // client.probability\_mutate\_link/=1.05;  
 // break;  
  
 // case ConsoleKey.T:  
 // newDirection = Target.probability\_mutate\_node\_increase;  
 // client.probability\_mutate\_node\*=1.05;  
 // break;  
  
 // case ConsoleKey.Y:  
 // newDirection = Target.probability\_mutate\_node\_decrease;  
 // client.probability\_mutate\_node/=1.05;  
 // break;  
  
 // case ConsoleKey.U:  
 // newDirection = Target.specie\_distance\_decrease;  
 // client.specie\_distance/=1.05;  
 // break;  
  
 // case ConsoleKey.I:  
 // newDirection = Target.kill\_rate\_increase;  
 // client.kill\_rate\*=1.05;  
 // break;  
  
 // case ConsoleKey.O:  
 // newDirection = Target.kill\_rate\_decrease;  
 // client.kill\_rate/=1.05;  
 // break;  
  
 // default:  
 // return;  
 // }  
 //}  
 }  
}

## .\NeaLibrary\NeuralNetwork\NEAT/Species.cs

using System;  
using System.Linq;  
using NeaLibrary.DataStructures;  
  
namespace NeaLibrary.NeuralNetwork.NEAT  
{  
 [Serializable]  
 public class Species : IComparable<Species>  
 {  
 ///<summary>This is a class to represent the species</summary>  
 public RandomHashedSet<Genotype> Members;  
 Genotype Representitive;  
 Neat master;  
 double average\_fitness;  
 public Species(Genotype representative)  
 {  
 Members = new RandomHashedSet<Genotype>();  
 Representitive = representative;  
 representative.specie = this;  
 Members.Add(Representitive);  
 master = representative.master;  
 }  
 public bool put(Genotype creature)  
 {  
 if (creature.Distance(Representitive) < master.specie\_distance)  
 {  
 Members.Add(creature);  
 return true;  
 }  
 return false;  
 }  
 public void force\_put(Genotype creature)  
 {  
 Members.Add(creature);  
 }  
 public void go\_extinct()  
 {  
 foreach (Genotype g in Members)  
 {  
 g.specie = null;  
 }  
 }  
 public double take\_average()  
 {  
 double sum = 0;  
 if (Members.Count == 0) return 0;  
 sum = Members.Sum(x => x.fitness);  
 return sum / Members.Count;  
 }  
 public void reset()  
 {  
 Genotype newrep = Members.GetRandom();  
 go\_extinct();  
 Members.Clear();  
 newrep.specie = this;  
 average\_fitness = 0;  
 //this.Representative  
 Representitive = newrep;  
 force\_put(newrep);  
 }  
 public void kill(double rate)  
 {  
 double victims = rate \* Members.Count;  
 for (int i = 0; i < victims; i++)  
 {  
 Genotype victm = Members.Min();  
 victm.specie = null;  
 Members.Remove(victm);  
 }  
 }  
 public Genotype breed()  
 {  
 Genotype a = Members.GetRandom();  
 Genotype b = Members.GetRandom();  
 Genotype c = a.fitness > b.fitness ? Genotype.Crossover(a, b) : Genotype.Crossover(b, a);  
 force\_put(c);  
 return c;  
 }  
  
 public int CompareTo(Species? other)  
 {  
 if (other == null) throw new Exception();  
 return average\_fitness.CompareTo(other.average\_fitness);  
 }  
 }  
}

## .\NeaLibrary\Tools/Misc.cs

﻿using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace NeaLibrary.Tools  
{  
 public class RandomHolder  
 {  
 private static Random \_instance;  
  
 public static Random Instance  
 {  
 get { return \_instance ?? (\_instance = new Random()); }  
 }  
 }  
 public class Globals  
 {  
   
 }  
}

## .\NeaLibrary\Tools/tools.cs

using System;  
using System.Text.Json;  
using Newtonsoft.Json;  
using System.Runtime.Serialization.Formatters.Binary;  
using System.IO;  
  
using NeaLibrary.DataStructures;  
using System.Collections.Generic;  
using System.Linq;  
using System.Threading;  
using System.Security.Cryptography;  
using NeaLibrary.Data.Other;  
using System.Runtime.CompilerServices;  
using Newtonsoft.Json.Linq;  
  
namespace NeaLibrary.Tools  
{  
 public static partial class Tools{  
 private static readonly Random rand;//=new Random();  
 //private static readonly RandomNumberGenerator secure\_rand;  
 #pragma warning disable  
 private static RNGCryptoServiceProvider secure\_rand = new RNGCryptoServiceProvider();  
 private static RandomHolder RandomHolder = new RandomHolder();  
  
 private static object settinglock = new object();  
  
 private static int seed = Environment.TickCount;  
 public static event EventHandler SettingsChanged;  
 private static string json\_settings = File.ReadAllText("settings.json"); //should be included in the bin  
 static Tools()  
 {  
 rand=new Random();  
 }  
  
 private static JsonSerializerOptions JSON\_options = new JsonSerializerOptions{  
 WriteIndented=true  
 };  
  
 /// <summary>  
 /// Gets solution files directory, just usefyul for testing and coding  
 /// </summary>  
 /// <returns></returns>  
 public static string get\_root\_folder\_directory()  
 {  
 string path=System.AppContext.BaseDirectory;  
  
 path = Directory.GetParent(path).ToString();  
 path = Directory.GetParent(path).ToString();  
  
 path = Directory.GetParent(path).ToString();  
 path = Directory.GetParent(path).ToString();  
  
 return path;  
 }  
  
 public static string GetGlobalVar(string s)  
 {  
 Newtonsoft.Json.Linq.JToken t = Newtonsoft.Json.Linq.JToken.Parse(json\_settings);  
 return t[s]["Value"].ToString().Trim();  
 }  
 public static string GetAllSettings()  
 {  
 return json\_settings;  
 }  
 public static void SaveGlobalVar(string s, SettingsItem val)  
 {  
 lock (settinglock)  
 {  
 Newtonsoft.Json.Linq.JToken t = Newtonsoft.Json.Linq.JToken.Parse(json\_settings);  
 //problem before was that we took a class serialised, and replaced it with a string. so when it tries to deserialise it it fails  
 //string serialisedSettingItem = JsonConvert.SerializeObject(val);  
 JObject serialisedSettingItemJObject = JObject.FromObject(val);  
 t[s] = serialisedSettingItemJObject;  
 json\_settings = t.ToString();  
 File.WriteAllText($"{System.AppContext.BaseDirectory}/settings.json", json\_settings);  
 SettingsChanged(new Tuple<string, SettingsItem>(s, val), new EventArgs());  
 }  
 }  
 public static void SetGlobalVar(string s, string val)  
 {  
 Newtonsoft.Json.Linq.JToken t = Newtonsoft.Json.Linq.JToken.Parse(json\_settings);  
 SettingsItem sett = t[s].ToObject<SettingsItem>();  
 sett.Value = val;  
 SaveGlobalVar(s, sett);  
 }  
  
 public static double RSI\_Calc(double prev, double newaverage, double pricechange ,int n=14){  
 double nposaverage;  
 double nnegaverage;  
 if(pricechange>0){  
 nposaverage = (prev \*(n-1)+pricechange)/n;  
 nnegaverage = (prev \*(n-1)+0)/n;  
 }else{  
 nposaverage = (prev \*(n-1)+0)/n;  
 nnegaverage = (prev \*(n-1)+pricechange)/n;  
 }  
 double RS = nposaverage/nnegaverage;  
 return 1/(1+RS);  
 }  
  
 public static double RandomDouble(double min, double max){  
 return rand.NextDouble() \* (max - min) + min;  
 }  
 public static double RandomDouble(){  
 return rand.NextDouble();  
 }  
 public static int RandomInt(int min, int max){  
 ///<summary>Inclusive random int</summary>  
 return rand.Next(min,max+1);//+1 to make it inclusive  
 }  
 public static int SecureRandomInt(int min, int max)  
 {  
 ///<summary>Inclusive random int</summary>  
 byte[] bytes = new byte[4];  
 secure\_rand.GetBytes(bytes);//+1 to make it inclusive  
 return BitConverter.ToInt32(bytes, 0);  
 }  
 public static int RandomInt\_ThreadSafe(int min, int max){  
 int n;  
 lock (rand)  
 {  
 n = rand.Next(min, max + 1);  
 }  
 return n;  
 }  
  
  
 public static double BiasedToStart(double min, double max){  
 return Math.Floor(Math.Abs(RandomDouble() - RandomDouble()) \* (1 + max - min) + min);  
 }//https://gamedev.stackexchange.com/questions/116832/random-number-in-a-range-biased-toward-the-low-end-of-the-range  
 //https://stackoverflow.com/questions/61056693/random-number-within-a-range-biased-towards-the-minimum-value-of-that-range  
 //random biased functions credited to those sources  
 public static double BiasedToStartSqrt(double min, double max){  
 return (1 - Math.Sqrt(1 - RandomDouble()))\*(max-min)+min;  
 }  
 public static int BiasedToStartInt(int min,int max){  
 return (int)Math.Floor(BiasedToStartSqrt(min,max));  
 }  
  
 public static T[][] SplitArrayEvenly<T>(T[] v, int chunks){  
 int chunksize = v.Length/chunks;  
 T[][] r = new T[chunks][];  
 for(int i =0; i<chunks-1; i++){  
 r[i] = new T[chunksize];  
 }  
 r[r.Length-1] = new T[v.Length%chunks];  
 for(int i =0; i<chunks\*chunksize; i++){  
 int chunk=i/chunksize;  
 r[chunk][i%chunksize]=v[i];  
 }  
 for(int i=chunks\*chunksize;i<v.Length;i++){  
 r[r.Length-1][i] = v[i];  
 }  
  
 return r;  
 }  
 public static int IndexOf<T>(IEnumerable<T> collection, T val)  
 {  
 int p = 0;  
 foreach(T t in collection)  
 {  
 if (t.Equals(val)) return p;  
 p++;  
 }return -1;  
 }  
 public static void Serialize(string path,object obj){  
   
 //File.WriteAllText(path,JsonSerializer.Serialize(obj,JSON\_options));  
 BinaryFormatter bf = new BinaryFormatter();  
 try  
 {  
 using (FileStream fs = new FileStream(path, FileMode.OpenOrCreate))  
 {  
 #pragma warning disable  
 bf.Serialize(fs, obj);  
 }  
 }  
 catch (UnauthorizedAccessException e)  
 {  
 throw e;  
 }  
 }  
  
 public static string CleanString(string s){  
 string r="";  
 foreach(char c in s){  
 if(Char.IsLetter(c)){  
 r+=c;  
 }else{  
 r+='\_';  
 }  
 }return r;  
 }  
 public static string Join(IEnumerable<string> strings){  
 return String.Join(",",strings);  
 }  
  
 public static void write\_to\_file(string message, string file)  
 {  
 using(StreamWriter sw = new StreamWriter(get\_root\_folder\_directory()+"/"+file, true))  
 {  
 sw.WriteLine(message);  
 }  
 }  
 }  
}

## .\NeaLibrary\Tools/toolsFinancial.cs

using NeaLibrary.DataStructures;  
namespace NeaLibrary.Tools  
{  
 public partial class Tools  
 {  
 /// <summary>  
 /// Stochastic oscillator over Closing prices, of length closing prices. Between 0 and 1  
 /// </summary>  
 /// <param name="ClosingPrices"></param>  
 /// <returns>between 0 and 1</returns>  
 public static double StochasticOscillator(Vector ClosingPrices)  
 {  
 double max = ClosingPrices.Max();  
 double min = ClosingPrices.Min();  
 return (ClosingPrices[0]-min)/(max-min);  
 }  
  
 public static double AroonOscillator(Vector ClosingPrices)  
 {  
 int n = ClosingPrices.dimension;  
 int aroon\_up = (n - IndexOf(ClosingPrices,ClosingPrices.Max())) / n;  
 int aroon\_down = (n - IndexOf(ClosingPrices,ClosingPrices.Min())) / n;  
  
 return aroon\_up-aroon\_down;  
 }  
 public static (double,double) RS\_Step1(Vector closes, int span=14)///<summary>Vector of close prices, most recent last</summary>  
 {  
 int n = closes.dimension;  
 if (n < span) throw new Exception("Need more data values");  
 double gains=0;  
 double losses=0;  
 double temp = closes[0];  
 for (int i = 1; i < span; i++)  
 {  
 if (closes[i]-temp>0)  
 {  
 gains += closes[i]-temp;  
 }  
 else  
 {  
 losses += temp - closes[i];  
 }  
 temp = closes[i];  
 }  
 gains = gains/n;  
 losses = losses/n;  
 return (gains, losses);  
 }  
 public static (double,double) RS\_Repetitive(double prev\_av\_gain, double prev\_av\_loss, double gain, double loss, int span=14)  
 {  
 prev\_av\_gain = ((span - 1) \* prev\_av\_gain + gain)/span;  
 prev\_av\_loss = ((span - 1) \* prev\_av\_loss + loss)/span;  
 return (prev\_av\_gain,prev\_av\_loss);  
 }  
 public static double RSI(double prev\_av\_gain, double prev\_av\_loss) => 1-(1 / (1+ (prev\_av\_gain/prev\_av\_loss) ) );  
 public static IEnumerable<double> RSI(IEnumerable<double> closing\_prices, int span=14)  
 {   
 int n = 0;  
 double temp = 0;  
 double gain=0, loss=0;  
 Vector stage1 = new Vector(span);  
 foreach (double val in closing\_prices)  
 {  
 if (n<span)  
 {//first stage  
 stage1.Insert\_at\_end\_Same\_Length(val);  
 yield return 0.5; //sensible default value  
 }  
 else if(n==span)  
 {  
 (gain, loss) = RS\_Step1(stage1,span);  
 yield return RSI(gain,loss);  
 }  
 else   
 {  
 double cur\_g = (val - temp > 0) ? val-temp : 0;  
 double cur\_l = (val - temp < 0) ? temp-val : 0;  
 (gain,loss)=RS\_Repetitive(gain,loss, cur\_g, cur\_l);  
 yield return RSI(gain,loss);  
 }  
 temp = val;  
 n++;  
 }  
 }  
 }  
}

## .\NeaLibrary\Tools/toolsMathematical.cs

using System;  
using System.Linq;  
using NeaLibrary.DataStructures;  
namespace NeaLibrary.Tools  
{  
 public partial class Tools{  
 //functions  
 public static IDataSet Vector\_XOR\_Example(int len, bool noise=true, double strength=0.005){  
 Vector[] ins = new Vector[len];  
 Vector[] outs = new Vector[len];  
 DataSet ds = new DataSet();  
 for (int i = 0; i < len; i++)  
 {   
 ins[i] = new Vector(2);  
 ins[i][0]= Tools.RandomInt(0,1);  
 ins[i][1]= Tools.RandomInt(0,1);  
 outs[i] = new Vector(1);  
 outs[i][0] = (ins[i][0]==ins[i][1])? 0 : 1;  
  
 ins[i] = VectorRangeNoisify(ins[i], strength);  
 outs[i] = VectorRangeNoisify(outs[i], strength);  
  
 ds.Add(ins[i], outs[i]);  
 }  
 return ds;  
 }  
 public static IDataSet Vector\_XOR\_Example2(int len, bool noise=true, double strength=0.005){  
 Vector[] ins = new Vector[len];  
 Vector[] outs = new Vector[len];  
 DataSet ds = new DataSet();  
 for (int i = 0; i < len; i++)  
 {   
 ins[i] = new Vector(2);  
 int[] choice= new int[]{-1,1};  
 ins[i][0]= choice[Tools.RandomInt(0,1)];  
 ins[i][1]= choice[Tools.RandomInt(0,1)];  
 outs[i] = new Vector(1);  
 outs[i][0] = (ins[i][0]==ins[i][1])? -1 : 1;  
  
 ins[i] = VectorRangeNoisify(ins[i], strength);  
 outs[i] = VectorRangeNoisify(outs[i], strength);  
 ds.Add(ins[i], outs[i]);  
 }  
 return ds;  
 }  
 //Vectorial  
 public static Vector RandomVector(int dimension, double min, double max){  
 Vector r=new Vector(dimension);  
 for(int i=0;i<dimension;i++){  
 r[i] = Tools.RandomDouble(min,max);  
 }  
 return r;  
 }  
 public static Vector[] SplitVectorEvenly(Vector v, int chunks){  
 int chunksize = v.dimension/chunks;  
 Vector[] r = new Vector[chunks];  
 for(int i =0; i<chunks-1; i++){  
 r[i] = new Vector(chunksize);  
 }  
 r[r.Length-1] = new Vector(v.dimension%chunks);  
 for(int i =0; i<chunks\*chunksize; i++){  
 int chunk=i/chunksize;  
 r[chunk][i%chunksize]=v[i];  
 }  
 for(int i=chunks\*chunksize;i<v.dimension;i++){  
 r[r.Length-1][i] = v[i];  
 }  
  
 return r;  
 }  
  
  
  
 public static double first\_deriv\_from\_values(Vector v)  
 {  
 if (v.dimension<=1) { return 0; }  
 Vector difs = new Vector(v.dimension-1);  
 for (int i=0;i<v.dimension-1;i++)  
 {  
 difs[i] = v[i+1]-v[i];  
 }  
 return difs.Sum() / difs.dimension;  
 }  
  
  
 public static Vector VectorNoisify(Vector v, double strength=0.01){  
 return strength\*v;  
 }  
 public static Vector VectorRangeNoisify(Vector v, double range=0.01){  
 Vector r= new Vector(v.dimension);  
 for(int i=0;i<v.dimension;i++){  
 r[i] = v[i] + RandomDouble(-range,range);  
 }  
 return r;  
 }  
  
 public static bool isPeak(double a, double b, double c)  
 {  
 if (a>=b) return false;  
 if (c>=b) return false;  
 return true;  
 }  
 public static bool isThrough(double a, double b, double c)  
 {  
 if (a <= b) return false;  
 if (c <= b) return false;  
 return true;  
 }  
  
 /// <summary>  
 /// 1 - max  
 /// 0 - neither  
 /// -1 - min  
 /// credit: this code was inspired by a stack overflow answer. see link in comment below  
 /// </summary>  
 /// <returns></returns>  
 public static IEnumerable<(int ,double,int)> ExtractMinimaMaxima(List<double> values)  
 {  
 if (values.Count < 3) yield break;  
 for (int i=1;i<values.Count-1;i++)  
 {  
 if (isPeak(values[i - 1], values[i], values[i + 1])) yield return (1, values[i], i);  
 else if (isThrough(values[i - 1], values[i], values[i + 1])) yield return (-1, values[i], i);  
 else yield return (0, values[i], i);  
 }  
 }  
  
 //public static IEnumerable<Tuple<int, double>> LocalMaxima(IEnumerable<double> source, int windowSize) //https://stackoverflow.com/questions/5269000/finding-local-maxima-over-a-dynamic-range  
 //{  
 // // Round up to nearest odd value  
 // windowSize = windowSize - windowSize % 2 + 1;  
 // int halfWindow = windowSize / 2;  
  
 // int index = 0;  
 // var before = new Queue<double>(Enumerable.Repeat(double.NegativeInfinity, halfWindow));  
 // var after = new Queue<double>(source.Take(halfWindow + 1));  
  
 // foreach (double d in source.Skip(halfWindow + 1).Concat(Enumerable.Repeat(double.NegativeInfinity, halfWindow + 1)))  
 // {  
 // double curVal = after.Dequeue();  
 // if (before.All(x => curVal > x) && after.All(x => curVal >= x))  
 // {  
 // yield return Tuple.Create(index, curVal);  
 // }  
 // before.Enqueue(curVal);  
 // before.Dequeue();  
 // after.Enqueue(d);  
 // index++;  
 // }  
 //}  
 }  
}

## .\NeaLibrary\Tools/toolsNeuralNetwork.cs

using System;  
using System.Text.Json;  
using System.Runtime.Serialization.Formatters.Binary;  
using System.IO;  
  
using NeaLibrary.DataStructures;  
  
namespace NeaLibrary.Tools{  
 public static partial class Tools{  
 public static Func<double,double> Tanh = x => Math.Tanh(x);  
 public static Func<double,double> Derivative\_Tanh = x=> 1-Math.Pow(Tanh(x),2);  
 public static Func<double,double> Sigmoid = x => 1/(1+Math.Pow(Math.E,-x));  
 public static Func<double,double> Sigmoid\_Derivative = x => Sigmoid(x)\*(1-Sigmoid(x));  
 public static Func<double,double> Leaky\_ReLu = x => (x>0) ? x : 0.01\*x;  
 public static Func<double,double> Leaky\_ReLu\_Derivative = x => (x>0) ? 1 : 0.01;  
 public static Func<double,double> ReLu = x => (x>0)? x:0 ;  
 public static Func<double,double> ReLu\_Derivative = x => (x>0)? 1:0.01 ;  
 public static Func<double, double> ReLu2 = x => (x > -1) ? ((x>1)? 1 : x ) : -1;  
 public static Func<double, double> ReLu2\_Derivative = x => (x > -1) ? ((x > 1) ? 0 : 1) : 0;  
 public static Func<double,double> S\_ReLu = x => (x<1)? ReLu(x): 1;  
 public static Func<double,double> S\_ReLu\_Derivative = x => (x<1)? ReLu\_Derivative(x): 0.01;  
  
 public static Func<double, double> SimpleTreshholdFunction = x =>  
 {  
 double v=0;  
 if (x < -0.5) v = -1; else if (x > 0.7) v = 1; else if (-0.5 < x & x < 0.7) v = 0;  
 return v;  
 };  
  
   
 public static double MSE(Vector actual, Vector target){  
 if (actual.dimension != target.dimension) throw new Exception();  
 double r= 0;  
 for(int i =0; i<actual.dimension;i++){  
 r+=0.5\*Math.Pow(target[i]-actual[i],2);  
 }  
 return r;  
 }  
 public static double derivative\_MSE(double aL\_j, double y){  
 return aL\_j - y;  
 }  
  
 public static double Avg\_Cost(Vector[] a, Vector[] b ){  
 if (a.Length != b.Length) throw new Exception();  
 double s=0;  
 for (int i=0;i<a.Length;i++){  
 s+=MSE(a[i],b[i]);  
 }return s/a.Length;  
 }  
  
 public static Vector Softmax(Vector rawinput)  
 {  
 //https://learn.microsoft.com/en-us/dotnet/api/system.linq.enumerable.aggregate?view=net-8.0  
 double expsum = rawinput.Aggregate(0.0, (total, v) => total+=Math.Exp(v));  
 Vector result = new Vector(rawinput.dimension);  
 for (int i = 0; i < rawinput.dimension; i++)  
 {  
 result[i] = Math.Exp(rawinput[i])/expsum;  
 }  
 return result;  
 }  
 }  
  
  
}

## .\NeaLibrary\WebRequest/WebRequest.cs

using System;  
//using System.Text.Json;  
using System.Net;  
using System.IO;  
using System.Net.Http;  
//using System.Net.Http.Headers;  
//using System.Net.Http.Json;  
namespace NeaLibrary.WebRequests  
{  
 public static class WebRequestsHandler  
 {  
 //private const string URL = @"http://www.alphavantage.co/query?function={}&symbol={}&outputsize=full&apikey={}";  
 //private const string DATA = @"{""object"":{""name"":""Name""}}";  
  
 public static string Req(string URL)  
 {  
#pragma warning disable  
 WebRequest request = WebRequest.Create(URL);  
  
 WebResponse response = request.GetResponse();  
  
 using (Stream dataStream = response.GetResponseStream())  
 {  
 StreamReader reader = new StreamReader(dataStream);  
 string responseFromServer = reader.ReadToEnd();  
  
 response.Close();  
 return responseFromServer;  
 }  
  
 }  
 }  
}

## .\test/Program.cs

﻿using NeaLibrary.Data;  
using NeaLibrary.Tools;  
using NeaLibrary.DataStructures;  
  
using System.Runtime.InteropServices;  
using NeaLibrary.NeuralNetwork;  
using NeaLibrary.Data.Other;  
using Newtonsoft.Json;  
using NeaLibrary.NeuralNetwork.FFNN;  
using Skender.Stock.Indicators;  
using static System.Net.Mime.MediaTypeNames;  
using NeaLibrary.NeuralNetwork.NEAT;  
  
namespace Program  
{  
 class Program  
 {  
 public static void Main(string[] args) {  
 //Neat n = new Neat(2,1,1000);  
 //Vector[] a, b;  
 //(a, b) = Tools.Vector\_XOR\_Example2(100);  
 //for (int i = 0; i < 100; i++)  
 //{  
 // n.Train(a, b,1,10);  
 // Console.WriteLine(n.best\_acc);  
 //}  
  
 string path = System.AppContext.BaseDirectory;  
  
  
 string osKind = "Unknown";  
 if (RuntimeInformation.IsOSPlatform(OSPlatform.Linux)) { osKind = "Linux"; }  
 if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows)) { osKind = "Windows"; }  
 Console.WriteLine($"You run on {osKind} : {RuntimeInformation.OSDescription}");  
  
  
  
#nullable enable  
 path = Directory.GetParent(path).ToString();  
 path = Directory.GetParent(path).ToString();  
  
 path = Directory.GetParent(path).ToString();  
 path = Directory.GetParent(path).ToString();  
  
 //path = Directory.GetParent(path).ToString();  
 path = Directory.GetParent(path).ToString();  
#nullable disable  
 //path += @"\NeaLibrary\Data\database";  
  
  
  
 //Console.WriteLine(Tools.GetGlobalVar("db\_dir"));  
 //path += $"/{db}";  
 //SQL\_Driver db = new SQL\_Driver(@"C:\Users\s220411\source\repos\SaeedMahar2006\NeaRemaster\Models\database.sqlite3");  
  
 ////foreach (double d in SQL\_Driver.ReadColumn<double>(db.conn, "AAPL", "EMA"))  
 ////{  
 //// Console.WriteLine(d);  
 ////}  
  
 //SQL\_Driver.CreateTable(db.conn,"demonstration","(testColumn INTEGER NOT NULL UNIQUE, PRIMARY KEY (testColumn))");  
 //var r= SQL\_Driver.Query(db.conn,"SELECT \* FROM sqlite\_schema WHERE tbl\_name='demonstration'");  
 //r.Read();  
 //Console.WriteLine(r.GetString(4));  
  
 //Data\_Handler dh = new Data\_Handler(Tools.GetGlobalVar("db\_dir"));  
  
 path = System.AppContext.BaseDirectory;  
#nullable enable  
 path = Directory.GetParent(path).ToString();  
 path = Directory.GetParent(path).ToString();  
  
 path = Directory.GetParent(path).ToString();  
 path = Directory.GetParent(path).ToString();  
  
 //path = Directory.GetParent(path).ToString();  
 path = Directory.GetParent(path).ToString();  
#nullable disable  
  
  
 //NEAT\_Handler neat = new NEAT\_Handler(2, 1, 1000);  
 //IDataSet ds = Tools.Vector\_XOR\_Example(1000, noise:true, strength:0.2);  
 //neat.Train(ds);  
  
  
  
  
 //while (true)  
 //{  
 // Console.WriteLine("enter name");  
 // string na = Console.ReadLine();  
  
 // dh.Fetch(2, new string[] { na, "14", "ID0BMKXS5HQ4MIST" }, false);  
 // dh.Fetch(1, new string[] { na, "full", "ID0BMKXS5HQ4MIST" }, false);  
 //}  
  
 //using (IEnumerator<string> date = SQL\_Driver.ReadColumn<string>(db.conn, "IBM", "Date", "1=1 ORDER BY Date DESC").GetEnumerator())  
 //{  
 // using (IEnumerator<double> rsi = SQL\_Driver.ReadColumn<double>(db.conn, "IBM", "RSI", "1=1 ORDER BY Date DESC").GetEnumerator())  
 // {  
 // while (date.MoveNext() && rsi.MoveNext())  
 // {  
 // Console.Write(date.Current);  
 // if (rsi.Current > 0.7)  
 // {  
 // Console.ForegroundColor= ConsoleColor.Red;  
 // }  
 // else if(rsi.Current<0.3)  
 // {  
 // Console.ForegroundColor = ConsoleColor.Green;  
 // }  
 // else  
 // {  
 // Console.ForegroundColor = ConsoleColor.Blue;  
 // }  
 // Console.WriteLine($" {rsi.Current}");  
 // Console.ForegroundColor = ConsoleColor.Gray;  
 // }  
 // }  
 //}  
  
 //foreach (Vector v in SQL\_Driver.ReadRow\_AsVector(db.conn,"IBM",new string[] {"open","close","RSI"}))  
 //{  
 // v.Print();  
 //}  
 //Data\_Handler dh=new Data\_Handler(path);  
  
  
  
 //while (true)  
 //{  
 // Console.WriteLine("aaaa");  
 // string t = Console.ReadLine();  
 // dh.Fetch(1, new string[] { t, "full", "ID0BMKXS5HQ4MIST" });  
 // dh.Fetch(2, new string[] { t, "14", "ID0BMKXS5HQ4MIST" }, false);  
 // Calculator.CalculateRSI(t);  
 // Console.WriteLine("bb");  
 // Calculator.CalculateStochastic(t);  
 //}  
  
  
 //SourceParameterInfo s = new SourceParameterInfo();  
 //s.Name = "a";  
 //s.IsRegex=true;  
 //s.Regex="a";  
 //s.IsCategorical=true;  
 //s.Categories=new string[] { "a","b","c"};  
  
 //Console.WriteLine(JsonConvert.SerializeObject(s));  
 //Console.WriteLine();  
  
 //var watch = System.Diagnostics.Stopwatch.StartNew();  
 //Matrix[] ar = new Matrix[10001];  
 //for (int i=0;i<10001;i++)  
 //{  
 // Matrix m = new Matrix(12, 12).Randomise(-1, 1);  
 // ar[i] = m;  
 //}  
 //var elapsedMs = watch.ElapsedMilliseconds;  
 //Console.WriteLine(elapsedMs);  
  
  
 //var watch2 = System.Diagnostics.Stopwatch.StartNew();  
 //for (int i = 0; i < 10000; i++)  
 //{  
 // Matrix r = ar[i] \* ar[i + 1];  
 //}  
 //watch.Stop();  
 //var elapsedMs2 = watch2.ElapsedMilliseconds;  
 //Console.WriteLine(elapsedMs2);  
  
  
 //DataSet XOR\_example = new DataSet();  
 //Vector[] inputs, labels;  
 //(inputs, labels) = Tools.Vector\_XOR\_Example(100000, noise: true);  
 //for (int i = 0; i < inputs.Length; i++)  
 //{  
 // XOR\_example.Add(inputs[i], labels[i]);  
 //}  
 //XOR\_example.Save("C:/Users/s\_mah/source/repos/Nea/Models/XOR\_Test.ds");  
  
 //IDataSet iDataSet = DataSet.Load("C:/Users/s\_mah/source/repos/Nea/Models/XOR\_Test.ds");  
 //DataSet dataSet = (DataSet)DataSet.Load("C:/Users/s\_mah/source/repos/Nea/Models/XOR\_Test.ds");  
  
 //dataSet.ElementAt(1234).Item1.Print(); dataSet.ElementAt(1234).Item2.Print();  
 //iDataSet.ElementAt(1234).Item1.Print(); iDataSet.ElementAt(1234).Item2.Print();  
  
 //client.Save(@"C:\Users\s220411\source\repos\SaeedMahar2006\NeaRemaster\test\test.ffnn");  
  
  
 IDataSet ds = Tools.Vector\_XOR\_Example(10000);  
 ds.Save(path+"/xor.ds");  
  
  
  
 //Console.WriteLine("a");  
 ////ThreadPool.QueueUserWorkItem((state) => { Calculator.CalculateMacd("MSFT"); Console.WriteLine("macd"); });  
 ////ThreadPool.QueueUserWorkItem((state) => { Calculator.CalculateAdx("MSFT"); Console.WriteLine("adx"); });  
 ////ThreadPool.QueueUserWorkItem((state) =>{ Calculator.CalculateAroon("MSFT"); Console.WriteLine("aroon"); });  
 ////var v = Task.Run(() => { Calculator.CalculateMacd("MSFT"); Console.WriteLine("macd"); });  
 ////var v2 = Task.Run(() => { Calculator.CalculateAdx("MSFT"); Console.WriteLine("adx"); });  
 ////var v3 = Task.Run(() => { Calculator.CalculateAroon("MSFT"); Console.WriteLine("Aroon"); });  
 //var v3 = Task.Run(() => { Calculator.CalculateSmi("MSFT"); Console.WriteLine("Smi"); });  
 //var v2 = Task.Run(() => { Calculator.CalculateSma("MSFT"); Console.WriteLine("Sma"); });  
 //var v1 = Task.Run(() => { Calculator.CalculateCci("MSFT"); Console.WriteLine("Cci"); });  
 //v1.Wait();  
 //// Calculator.CalculateMfi("MSFT");  
 //Console.WriteLine("b");  
  
 //DB\_DataSet ds = (DB\_DataSet)DB\_DataSet.Load(@"C:\Users\s\_mah\source\repos\Nea\Mega.dbds");  
 //foreach ((DateTime, Vector, Vector, Vector) v in ds.Debug(1000, 42, "IBM"))  
 //{  
 // Console.WriteLine(v.Item1);  
 // v.Item2.Print();  
 // v.Item3.Print();  
 // v.Item4.Print();  
 // Console.WriteLine("\n\n");  
 //}  
  
 //DB\_DataSet Ds = new DB\_DataSet(new List<string> { "IBM" },  
 // new List<string> { "close", "volume", "EMA", "RSI" },  
 // new List<string> { },  
 // new List<string> { "close", "EMA", "volume" },  
 // new List<string> {}, 2  
 // );  
 //DataSet ds = Ds.ToDataSet();  
 //foreach ((Vector, Vector) d in ds)  
 //{  
 // d.Item1.Print();  
 // d.Item2.Print();  
 // Console.WriteLine();  
 //}  
  
 //Ds.Cache();  
  
 //Ds.Save(path+"/data.dbds");  
 //ds.Save(path + "/data2.ds");  
  
 //DataSet ds2= (DataSet)DataSet.Load(path + "/data2.ds");  
 //Console.ReadLine();  
 //foreach ((Vector, Vector) d in ds)  
 //{  
 // d.Item1.Print();  
 // d.Item2.Print();  
 // Console.WriteLine();  
 //}  
  
 //Console.WriteLine(ds2.GetType());  
 //Vector v = new Vector(4);  
 //v[0] = 4;  
 //v[1] = 8;  
 //v[2] = 5;  
 //v[3] = 3;  
 //FFNN\_Client f =new FFNN\_Client(v, 0.1, 1000, ds);  
 //f.TrainAll();  
  
 //Neat n = new Neat(4,3,10000);  
 //n.Train(ds,0.9);  
  
 //foreach((Vector,Vector) data in ds)  
 //{  
 // data.Item1.Print();data.Item2.Print();Console.WriteLine();  
 //}  
 //Console.ReadLine();  
 //Neat n = new Neat(4, 3, 10);  
 //n.Train(ds2, 0.5);  
 //n.Save(path+"/Neat.neat");  
 //ID0BMKXS5HQ4MIST  
 //while (true)  
 //{  
 // Console.WriteLine("Enter:");  
 // string choice = Console.ReadLine();  
 // string[] ar = new string[] { choice, "full", "ID0BMKXS5HQ4MIST", };  
 // dh.Fetch(1, ar);  
 // SQL\_Driver sql = new SQL\_Driver();  
 // //foreach (double d in SQL\_Driver.ReadColumn<double>(db.conn, "IBM", "close"))  
 // //{  
 // // Console.WriteLine(d);  
 // //}  
 // Console.WriteLine();  
 // //using (IEnumerator<string> date = SQL\_Driver.ReadColumn<string>(db.conn, choice, "Date", "1=1 ORDER BY Date ASC").GetEnumerator())  
 // //{  
 // // using (IEnumerator<double> rsi = Tools.RSI(SQL\_Driver.ReadColumn<double>(db.conn, choice, "close", "1=1 ORDER BY Date ASC")).GetEnumerator())  
 // // {  
 // // while (date.MoveNext() && rsi.MoveNext())  
 // // {  
 // // Console.SetCursorPosition(0, Console.CursorTop - 1);  
  
 // // Console.WriteLine($"{date.Current} {rsi.Current}");  
 // // Dictionary<string, string> dic = new Dictionary<string, string>();  
 // // dic.Add("RSI", rsi.Current.ToString());  
 // // SQL\_Driver.Update(db.conn, choice, dic, $"Date = '{date.Current}'");  
 // // }  
 // // }  
 // //}  
 // foreach (Vector v in )  
 // {  
  
 // }  
 //}  
  
 }  
 }  
}

## .\WinFormsApp2/Form1.cs

using NeaLibrary.Data;  
using NeaLibrary.DataStructures;  
using System.Diagnostics;  
using System.Drawing;  
using System.Runtime.InteropServices;  
using System.Security.Policy;  
using System.Windows.Forms;  
using WindowsFormsApp1;  
using WindowsFormsApp1.NEATControls;  
using WinFormsApp2.DataControls;  
using WinFormsApp2.FFNNControls;  
using WinFormsApp2.MainControls;  
using static System.Windows.Forms.VisualStyles.VisualStyleElement;  
  
namespace WinFormsApp2  
{  
 public partial class Form1 : Form  
 {  
 public Dictionary<Control, TabPage> ControlPageMap = new Dictionary<Control, TabPage>();  
 public Form1()  
 {  
 InitializeComponent();  
 ClearControlTabs();  
  
 //FetchDataControl a = new FetchDataControl();  
 //splitContainer1.Panel2.Controls.Add(a);  
 //a.Show();  
 //a.BringToFront();  
 //Settings a = new FetchDataControl();  
 //splitContainer1.Panel2.Controls.Add(a);  
 //a.Show();  
 //a.BringToFront();  
 }  
  
 private void AddControlToNewTab(Control c, bool show = true)  
 {  
 TabPage newPage = new TabPage(c.Name);  
 ControlTabs.TabPages.Insert(0, newPage);  
 newPage.Controls.Add(c);  
 c.Show();  
 ControlPageMap.Add(c, newPage);  
 if (show)  
 {  
 ControlTabs.SelectedTab = newPage;  
 }  
 }  
 private void RemoveControlFromTab(Control c)  
 {  
 c.Hide();  
 TabPage pg = ControlPageMap[c];  
 pg.Controls.Remove(c);  
 ControlPageMap.Remove(c);  
 ControlTabs.TabPages.Remove(pg);  
 c.Dispose();  
 }  
 private void ClearControlTabs()  
 {  
  
 ControlTabs.TabPages.Clear();  
  
 }  
  
 private void splitContainer1\_Panel2\_Paint(object sender, PaintEventArgs e)  
 {  
  
 }  
  
 private void splitContainer1\_Panel1\_Paint(object sender, PaintEventArgs e)  
 {  
 }  
  
 private void New\_Click(object sender, EventArgs e)  
 {  
 CreateSelector s = new CreateSelector();  
 //splitContainer1.Panel2.Controls.Add(s);  
 //s.Show();  
  
 s.Name = "Selector";  
 AddControlToNewTab(s);  
  
  
  
 s.ChoiceMade += new EventHandler(Handle\_CreateSelector\_Choice);  
 void Handle\_CreateSelector\_Choice(object sender, EventArgs e)  
 {  
 //MessageBox.Show($"OMG YES u chose {s.Selected}");  
 s.Hide();  
 switch (s.Selected)  
 {  
 case CreateSelector.Choice.NEAT:  
 New\_CreateNeat();  
 break;  
 case CreateSelector.Choice.FFNN:  
 New\_CreateFFNN();  
 break;  
 case CreateSelector.Choice.DataSet:  
 New\_CreateDataSet();  
 break;  
 case CreateSelector.Choice.DataTable:  
 New\_CreateDataTable();  
 break;  
 case CreateSelector.Choice.Invalid:  
 //Must have clicked exit no need to do anything  
 break;  
 }  
 RemoveControlFromTab(s);  
 }  
 }  
 private void New\_CreateNeat()  
 {  
 CreateNEATControl s = new CreateNEATControl();  
 AddControlToNewTab(s);  
 s.FinishedCreating += new EventHandler(Handle\_NeatCreator\_Created);  
 void Handle\_NeatCreator\_Created(object sender, EventArgs e)  
 {  
 RemoveControlFromTab(s);  
 }  
 }  
 private void New\_CreateFFNN()  
 {  
 FFNNCreateControl s = new FFNNCreateControl();  
 AddControlToNewTab(s);  
 s.FinishedCreating += new EventHandler(Handle\_FFNNCreator\_Created);  
 void Handle\_FFNNCreator\_Created(object sender, EventArgs e)  
 {  
 RemoveControlFromTab(s);  
 }  
 }  
 private void New\_CreateDataSet()  
 {  
 DataSetCreateControl s = new DataSetCreateControl();  
 AddControlToNewTab(s);  
 s.FinishedCreating += new EventHandler(Handle\_DatasetCreator\_Created);  
 void Handle\_DatasetCreator\_Created(object sender, EventArgs e)  
 {  
 RemoveControlFromTab(s);  
 }  
 }  
  
  
 private void New\_CreateDataTable()  
 {  
 FetchDataControl s = new FetchDataControl();  
 s.FinishedCreating += (s, e) =>  
 {  
 RemoveControlFromTab((FetchDataControl)s!); //dont like the !  
 };  
 AddControlToNewTab(s);  
 //s.FinishedCreating += new EventHandler(Handle\_DatasetCreator\_Created);  
 //void Handle\_DatasetCreator\_Created(object sender, EventArgs e)  
 //{  
 // RemoveControlFromTab(s);  
 //}  
 }  
  
 private void OpenB\_Click(object sender, EventArgs e)  
 {  
  
 using (OpenFileDialog ofd = new OpenFileDialog())  
 {  
 ofd.Multiselect = false;  
 ofd.Filter = " Data Set (\*.ads;\*.ds;\*dbds)|\*.ads;\*.ds;\*.dbds|FFNN (\*.ffnn)|\*.ffnn|NEAT (\*.neat)|\*.neat";  
 DialogResult result = ofd.ShowDialog();  
 if (result == DialogResult.OK && !string.IsNullOrWhiteSpace(ofd.FileName))  
 {  
 //.ads ArrayDataSet  
 //.ds DataSet  
 //.dbds DB\_DataSet  
 string path = ofd.FileName;  
 string c = ofd.FileName.Split('.').Last();  
 switch (c)  
 {  
 case "ads":  
 DataSetViewer dsva = new DataSetViewer();  
 DataSetViewerEventHandler(dsva);  
 dsva.ProgramaticAssign(path);  
 AddControlToNewTab(dsva);  
 break;  
 case "ds":  
 DataSetViewer dsvb = new DataSetViewer();  
 DataSetViewerEventHandler(dsvb);  
 dsvb.ProgramaticAssign(path);  
 AddControlToNewTab(dsvb);  
 break;  
 case "dbds":  
 DataSetViewer dsvc = new DataSetViewer();  
 DataSetViewerEventHandler(dsvc);  
 dsvc.ProgramaticAssign(path);  
 AddControlToNewTab(dsvc);  
 break;  
 case "ffnn":  
 FFNNControl fnncontrol = new FFNNControl();  
 fnncontrol.FFNN\_Programatic\_Assign(ofd.FileName);  
 fnncontrol.GraphDisplayRequested += OnGraphDisplayRequested;  
 AddControlToNewTab(fnncontrol);  
 fnncontrol.Close += (s, e) =>  
 {  
 fnncontrol.Hide();  
 RemoveControlFromTab(fnncontrol);  
 };  
 break;  
 case "neat":  
 NEATControl nEATControl = new NEATControl();  
 nEATControl.GraphDisplayRequested += OnGraphDisplayRequested;  
 AddControlToNewTab(nEATControl);  
 nEATControl.Neat\_Programatic\_Assign(ofd.FileName);  
 nEATControl.Close += (s, e) =>  
 {  
 nEATControl.Hide();  
 RemoveControlFromTab(nEATControl);  
 };  
 break;  
 }  
  
 }  
 }  
 }  
 void OnGraphDisplayRequested(object? s, GraphDisplay e)  
 {  
 e.Exit += (sender, eventsargs) => RemoveControlFromTab(e);  
 AddControlToNewTab(e);  
 e.Render();  
 }  
  
 private void DataSetViewerEventHandler(DataSetViewer con)  
 {  
 con.Exit += (s, e) =>  
 {  
 RemoveControlFromTab(con);  
 };  
 con.RequestTabForStockChart += (sender, e) =>  
 {  
 StockChart sc = new StockChart();  
 sc.dataset = e;  
 AddControlToNewTab(sc);  
 sc.InitialiseCharts();  
 sc.SetTokens(sc.dataset.tables);  
 sc.Exit += (sender, e) =>  
 {  
 sc.Hide();  
 RemoveControlFromTab(sc);  
 };  
 };  
 }  
  
 private void splitContainer1\_SplitterMoved(object sender, SplitterEventArgs e)  
 {  
  
 }  
  
 private void SettingsB\_Click(object sender, EventArgs e)  
 {  
 SettingsControl a = new SettingsControl();  
 a.Exit += (s, e) => { RemoveControlFromTab(a); };  
 AddControlToNewTab(a);  
 }  
  
 private void PredictionB\_Click(object sender, EventArgs e)  
 {  
 MakePredictionControl s = new MakePredictionControl();  
 s.Name = "Prediction";  
 s.VectorViewRequested += (s, v) =>  
 {  
 VectorViewer vectorViewer = new VectorViewer();  
 vectorViewer.Display(v);  
 vectorViewer.Name = "Vector Viewer";  
 AddControlToNewTab(vectorViewer);  
 vectorViewer.Show();  
 vectorViewer.Close += (snder, ev) =>  
 {  
 RemoveControlFromTab(vectorViewer);  
 };  
 };  
  
 AddControlToNewTab(s);  
 s.Show();  
 s.Exit += (sender, ev) =>  
 {  
 RemoveControlFromTab(s);  
 };  
 }  
  
  
  
 private void credits\_Click(object sender, EventArgs e)  
 {  
 //this method code attributed to https://brockallen.com/2016/09/24/process-start-for-urls-on-net-core/  
  
 string url = @"https://github.com/SaeedMahar2006";  
 try  
 {  
 Process.Start(url);  
 }  
 catch  
 {  
 // comment from source // hack because of this: https://github.com/dotnet/corefx/issues/10361  
 if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))  
 {  
 url = url.Replace("&", "^&");  
 Process.Start(new ProcessStartInfo("cmd", $"/c start {url}") { CreateNoWindow = true });  
 }  
 //else if (RuntimeInformation.IsOSPlatform(OSPlatform.Linux))// not needed this is windows forms  
 //{  
 // Process.Start("xdg-open", url);  
 //}  
 //else if (RuntimeInformation.IsOSPlatform(OSPlatform.OSX))  
 //{  
 // Process.Start("open", url);  
 //}  
 else  
 {  
 throw;  
 }  
 }  
  
 }  
  
 }  
}

## .\WinFormsApp2/Program.cs

namespace WinFormsApp2  
{  
 internal static class Program  
 {  
 /// <summary>  
 /// The main entry point for the application.  
 /// </summary>  
 [STAThread]  
 static void Main()  
 {  
 // To customize application configuration such as set high DPI settings or default font,  
 // see https://aka.ms/applicationconfiguration.  
 ApplicationConfiguration.Initialize();  
 Application.Run(new Form1());  
 }  
 }  
}

## .\WinFormsApp2\DataControls/ArgumentEditorControl.cs

﻿using Microsoft.Diagnostics.Tracing.Parsers.FrameworkEventSource;  
using NeaLibrary.Data.Other;  
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Runtime.Serialization;  
using System.Text;  
using System.Text.RegularExpressions;  
using System.Threading.Tasks;  
using System.Windows.Forms;  
  
namespace WinFormsApp2.DataControls  
{  
 public partial class ArgumentEditorControl : UserControl  
 {  
 public event EventHandler ValueChanged;  
 bool first = true;  
 Regex regex\_filter=new Regex(@".+");  
 string[] categories;  
 //bool isSourceParameterInfo = true;  
 SourceParameterInfo \_sourceParameterInfo;  
 //SettingsItem \_settingsItemInfo;  
 public ArgumentEditorControl()  
 {  
 InitializeComponent();  
 }  
 public void SetInfo(SourceParameterInfo sourceParameterInfo)  
 {  
 //isSourceParameterInfo = true;  
 first = true;  
 if (sourceParameterInfo.IsRegex)  
 {  
 regex\_filter = new Regex(sourceParameterInfo.Regex);  
 textBox1.Show();  
 textBox1.Text = "";  
 domainUpDown1.Hide();  
 numericUpDown.Hide();  
 }  
 else if (sourceParameterInfo.IsCategorical)  
 {  
 categories = sourceParameterInfo.Categories;  
 textBox1.Hide();  
 numericUpDown.Hide();  
 domainUpDown1.Show();  
 domainUpDown1.Items.Clear();  
 foreach (string category in categories) domainUpDown1.Items.Add(category);  
 domainUpDown1.SelectedItem = domainUpDown1.Items[0];  
 first = false;  
 }  
 else if (sourceParameterInfo.IsNumeric)  
 {  
 textBox1.Hide();  
 domainUpDown1.Hide();  
 numericUpDown.Show();  
 numericUpDown.Minimum = (decimal) sourceParameterInfo.BoundsAndStep[0];  
 numericUpDown.Maximum = (decimal) sourceParameterInfo.BoundsAndStep[1];  
 numericUpDown.Increment = (decimal) sourceParameterInfo.BoundsAndStep[2];  
 numericUpDown.Value = 0;//(decimal) 0.5\*(numericUpDown.Minimum+numericUpDown.Maximum);  
 first = false;  
 }  
 else  
 {  
 regex\_filter = new Regex(@".+"); //anything except newline, must be something  
 textBox1.Show();  
 textBox1.Text = "";  
 domainUpDown1.Hide();  
 numericUpDown.Hide();  
 }  
 \_sourceParameterInfo = sourceParameterInfo;  
 }  
 //public void SetInfo(SettingsItem settingsItem)  
 //{  
 // isSourceParameterInfo=false;  
 // first = true;  
  
 //}  
 public string GetInfo()  
 {  
 if (\_sourceParameterInfo.IsRegex)  
 {  
 return textBox1.Text;  
 }  
 else if (\_sourceParameterInfo.IsCategorical)  
 {  
 return domainUpDown1.SelectedItem.ToString();  
 }  
 else  
 {  
 return textBox1.Text;  
 }  
 }  
  
 public object GetValue()  
 {  
 if (\_sourceParameterInfo.IsRegex)  
 {  
 return textBox1.Text;  
 }  
 else if (\_sourceParameterInfo.IsCategorical)  
 {  
 return domainUpDown1.SelectedItem;  
 }  
 else if (\_sourceParameterInfo.IsNumeric)  
 {  
 return numericUpDown.Value;  
 }  
 else  
 {  
 return textBox1.Text;  
 }  
 }  
  
 private void textBox1\_TextChanged(object sender, EventArgs e)  
 {  
  
 }  
 //private bool Verify()  
 //{  
 // return true;  
 //}  
  
 //with reference to https://stackoverflow.com/questions/8915151/c-sharp-validating-input-for-textbox-on-winforms  
 private void textBox1\_Validating(object sender, CancelEventArgs e)  
 {  
 if (!regex\_filter.IsMatch(textBox1.Text)) e.Cancel = true;  
 ValueChanged(this, new EventArgs());  
 }  
  
 private void ArgumentL\_Click(object sender, EventArgs e)  
 {  
  
 }  
 public void setArgumentLabelText(string t)  
 {  
 ArgumentL.Text = t;  
 }  
  
 private void ArgumentEditorControl\_Load(object sender, EventArgs e)  
 {  
  
 }  
  
 private void domainUpDown1\_SelectedItemChanged(object sender, EventArgs e)  
 {  
 try { if (!first) OnValueChanged(domainUpDown1.SelectedItem); } catch { }  
 }  
  
 private void numericUpDown\_ValueChanged(object sender, EventArgs e)  
 {  
 try  
 {  
 if (!first) OnValueChanged(numericUpDown.Value);  
 }catch { }  
 }  
  
 private void OnValueChanged(object val)  
 {  
 EventHandler handler = ValueChanged;  
 if (handler != null) handler(this, EventArgs.Empty);  
 }  
 }  
}

## .\WinFormsApp2\DataControls/DataSetCreateControl.cs

﻿using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
using System.Windows.Forms;  
using NeaLibrary.Data;  
using NeaLibrary.DataStructures;  
  
namespace WinFormsApp2.DataControls  
{  
 public partial class DataSetCreateControl : UserControl  
 {  
 string DB\_PATH = NeaLibrary.Tools.Tools.GetGlobalVar("db\_dir");  
 List<string> Assets = new List<string>();  
 List<string> tables;  
 List<string> selected\_tables = new List<string>();  
 List<string> columns;  
 object selected\_cols\_lock = new object();  
 List<string> selected\_columns = new List<string>();  
 List<string> normalise\_columns = new List<string>();  
 List<string> relative\_columns = new List<string>();  
 List<string> value\_column = new List<string>();  
 SQL\_Driver db;  
  
 public object creatinglock = new object();  
 public event EventHandler FinishedCreating;  
  
 //private event EventHandler ColumnsSelectedChange;  
  
  
 class CheckBoxDataBinding  
 {  
 public bool IsChecked { get; set; }  
 public string Name { get; set; }  
 public CheckBoxDataBinding()  
 {  
 this.IsChecked = false;  
 Name = "null";  
 }  
 }  
  
  
 public DataSetCreateControl()  
 {  
 InitializeComponent();  
 OpenDB();  
 GetAssetTypes();  
 //NormaliseRelative();  
 }  
  
 private void OpenDB()  
 {  
 db = new SQL\_Driver(DB\_PATH);  
 DBPathTextBox.Text = DB\_PATH;  
 }  
  
 private void GetAssetTypes()  
 {  
 Assets.Clear();  
 AssetSpecifierUpDown.Items.Clear();  
 foreach (string a in SQL\_Driver.ReadColumn<string>(db.conn, "AssetType", "AssetType"))  
 {  
 Assets.Add(a);  
 AssetSpecifierUpDown.Items.Add(a);  
 }  
 }  
  
 private void LoadTablesCheckBox(string AssetType)  
 {  
 TableSelectionCheckBox.Items.Clear();  
 tables = new List<string>();  
 tables.AddRange(SQL\_Driver.ReadColumn<string>(db.conn, "TokenNames", "Token", $"AssetType='{AssetType}'"));  
 foreach (string s in tables) TableSelectionCheckBox.Items.Add(s);  
  
  
 }  
 private void LoadColumnsCheckBox()  
 {  
 //SELECT name FROM PRAGMA\_TABLE\_INFO('IBM') WHERE type='REAL';  
  
 columns = new List<string>();  
 try  
 {  
 columns.AddRange(SQL\_Driver.ReadColumn<string>(db.conn, $"PRAGMA\_TABLE\_INFO('{tables.First()}')", "name", "type = 'REAL'"));  
  
 //List<string> tables = new List<string>();  
 ColumnSelectionCheckBox.Items.Clear();  
 for (int i = 0; i < columns.Count; i++) { ColumnSelectionCheckBox.Items.Add(columns[i]); }  
 }  
 catch (Exception e)  
 {  
 MessageBox.Show($"Perhaps there are no data tables, try make a data table first: {e.ToString()}");  
 }  
 //((ListBox)ColumnSelectionCheckBox).DataSource = columns;  
 //((ListBox)ColumnSelectionCheckBox).DisplayMember = "Name";  
 //((ListBox)ColumnSelectionCheckBox).ValueMember = "IsChecked";  
  
 //foreach (CheckBoxDataBinding b in ColumnSelectionCheckBox.Items) { b.IsChecked = true; }  
  
  
 }  
  
 private void AssetSpecifierUpDown\_SelectedItemChanged(object sender, EventArgs e)  
 {  
 //TODO  
 LoadTablesCheckBox(AssetSpecifierUpDown.SelectedItem.ToString());  
 LoadColumnsCheckBox();  
 }  
  
 private void ChangeDBButton\_Click(object sender, EventArgs e)  
 {  
 OpenFileDialog OpenDatafileDialogue = new OpenFileDialog();  
 OpenDatafileDialogue.Multiselect = false;  
 OpenDatafileDialogue.Filter = "Sqlite3 database | \*.sqlite3";  
 DialogResult result = OpenDatafileDialogue.ShowDialog();  
 if (result == DialogResult.OK && !string.IsNullOrWhiteSpace(OpenDatafileDialogue.FileName))  
 {  
 DB\_PATH = OpenDatafileDialogue.FileName;  
 OpenDB();  
 DBPathTextBox.Text = OpenDatafileDialogue.FileName;  
 }  
 }  
  
 private void Update\_Normalise\_Relative\_CheckBox()  
 {  
 NormaliseSelectorCheckBox.Items.Clear();  
 RelativeSelectorCheckBox.Items.Clear();  
 foreach (string s in selected\_columns)  
 {  
 RelativeSelectorCheckBox.Items.Add(s);  
 NormaliseSelectorCheckBox.Items.Add(s);  
 }  
 }  
  
  
 private void ColumnSelectionCheckBox\_CheckChanged(object sender, ItemCheckEventArgs e)  
 {  
 //selected\_columns.Clear();  
 //foreach (string s in ColumnSelectionCheckBox.CheckedItems)  
 //{  
 // RelativeSelectorCheckBox.Items.Add(s);  
 // NormaliseSelectorCheckBox.Items.Add(s);  
 //}  
 selected\_columns.Clear();  
  
 foreach (var item in ColumnSelectionCheckBox.CheckedItems) selected\_columns.Add(item.ToString());  
  
 if (e.NewValue == CheckState.Checked) // Found on Stack Overflow  
 {  
 selected\_columns.Add(ColumnSelectionCheckBox.Items[e.Index].ToString());  
 }  
 else  
 {  
 selected\_columns.Remove(ColumnSelectionCheckBox.Items[e.Index].ToString());  
 }  
  
 Update\_Normalise\_Relative\_CheckBox();  
 }  
  
 private void CreateButton\_Click(object sender, EventArgs e)  
 {  
 //OutputDataProducer out\_p = new OutputDataProducer((double)SafetySelectorNUpDown.Value);  
 if (AssetSpecifierUpDown.SelectedItem == null)  
 {  
 MessageBox.Show("Select an asset type");  
 return;  
 }  
 if (selected\_tables.Count==0)  
 {  
 MessageBox.Show("Select a tables");  
 return;  
 }  
 if (selected\_columns.Count == 0)  
 {  
 MessageBox.Show("Select columns");  
 return;  
 }  
 if (ValueColumnUpDown.SelectedItem == null)  
 {  
 MessageBox.Show("Select a vlaue column");  
 return;  
 }  
  
 lock (creatinglock)  
 {  
 ExitB.Enabled = false;  
 ExitB.Update();  
 SaveFileDialog SaveFileDialogue = new SaveFileDialog();  
 string ext = (ExportAsDatasetCheckBox.Checked) ? ".ds" : ".dbds";  
 SaveFileDialogue.DefaultExt = ext;  
 DialogResult result = SaveFileDialogue.ShowDialog();  
 if (result == DialogResult.OK && !string.IsNullOrWhiteSpace(SaveFileDialogue.FileName))  
 {  
 double safety = (double)SafetySelectorNUpDown.Value;  
 DB\_DataSet dbds = new DB\_DataSet(selected\_tables, selected\_columns, normalise\_columns, relative\_columns, new List<string>(), value\_column, safety, 1, offset: (int)InternalDBOffsetNUpDown.Value);  
  
 if (ExportAsDatasetCheckBox.Checked)  
 {  
 NeaLibrary.DataStructures.DataSet ds = dbds.ToDataSet();  
 ds.Save(SaveFileDialogue.FileName);  
 }  
 else  
 {  
 dbds.Save(SaveFileDialogue.FileName);  
 }  
 FinishedCreating(this, new EventArgs());  
 }  
 }  
 }  
  
 private void RelativeSelectorCheckBox\_ItemCheck(object sender, ItemCheckEventArgs e)  
 {  
 relative\_columns.Clear();  
  
 foreach (var item in RelativeSelectorCheckBox.CheckedItems) relative\_columns.Add(item.ToString());  
  
 if (e.NewValue == CheckState.Checked) // Found on Stack Overflow  
 {  
 relative\_columns.Add(RelativeSelectorCheckBox.Items[e.Index].ToString());  
 }  
 else  
 {  
 relative\_columns.Remove(RelativeSelectorCheckBox.Items[e.Index].ToString());  
 }  
 //Console.WriteLine();  
 ValueColumnUpDown.Items.Clear();  
 foreach (string s in relative\_columns) ValueColumnUpDown.Items.Add(s);  
 }  
  
 private void NormaliseSelectorCheckBox\_ItemCheck(object sender, ItemCheckEventArgs e)  
 {  
 normalise\_columns.Clear();  
  
 foreach (var item in NormaliseSelectorCheckBox.CheckedItems) normalise\_columns.Add(item.ToString());  
  
 if (e.NewValue == CheckState.Checked) // Found on Stack Overflow  
 {  
 normalise\_columns.Add(NormaliseSelectorCheckBox.Items[e.Index].ToString());  
 }  
 else  
 {  
 normalise\_columns.Remove(NormaliseSelectorCheckBox.Items[e.Index].ToString());  
 }  
  
 }  
  
 private void TableSelectionCheckBox\_ItemCheck(object sender, ItemCheckEventArgs e)  
 {  
 selected\_tables.Clear();  
  
 foreach (var item in TableSelectionCheckBox.CheckedItems) selected\_tables.Add(item.ToString());  
  
 if (e.NewValue == CheckState.Checked) // Found on Stack Overflow  
 {  
 if (TableSelectionCheckBox.Items.Count == 0) return;  
  
 selected\_tables.Add(TableSelectionCheckBox.Items[e.Index].ToString());  
 }  
 else  
 {  
 selected\_tables.Remove(TableSelectionCheckBox.Items[e.Index].ToString());  
 }  
 }  
  
  
 private void VAlueColumnUpDown\_SelectedItemChanged(object sender, EventArgs e)  
 {  
 value\_column.Clear();  
 value\_column.Add(ValueColumnUpDown.SelectedItem.ToString());  
 }  
  
 private void NormaliseSelectorCheckBox\_SelectedIndexChanged(object sender, EventArgs e)  
 {  
  
 }  
  
 private void ColumnSelectionCheckBox\_SelectedIndexChanged(object sender, EventArgs e)  
 {  
  
 }  
  
 private void ExitB\_Click(object sender, EventArgs e)  
 {  
 //no need to check because button is disabled if something is being created  
 FinishedCreating(this, EventArgs.Empty);  
 }  
 }  
}

## .\WinFormsApp2\DataControls/DataSetViewer.cs

﻿using NeaLibrary.Data;  
using NeaLibrary.DataStructures;  
using ScottPlot;  
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Data.Common;  
  
using System.Diagnostics;  
using System.Diagnostics.Eventing.Reader;  
using System.Drawing;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
using System.Windows.Forms;  
using WinFormsApp2.FFNNControls;  
using System.Threading;  
using System.Windows.Forms;  
using System.Windows.Forms.DataVisualization.Charting;  
  
namespace WinFormsApp2.DataControls  
{  
 public partial class DataSetViewer : UserControl  
 {  
  
  
 private delegate void SafeCallDelegate(string text);  
 private delegate void CountReceivedDelegate(int b, int s, int h);  
 public delegate void RequestTabForStockChartHandler(object sender, DB\_DataSet e);  
 public event EventHandler Exit; public event RequestTabForStockChartHandler RequestTabForStockChart;  
  
 IDataSet dataset;  
 List<(Vector, Vector)> TableInfo = new List<(Vector, Vector)>();  
 object TableInfoLock = new object();  
 object DataChartLock = new object();  
 string DataSetType, InputCol, TableSourced, ValueCol, RelativeCol, Safety;  
 string default\_info =  
 "{0} Data Set\n" +  
 "Data Count: {1}\n" +  
 "Input Dimension: {2}\n" +  
 "Input Columns: {3}\n" +  
 "Output Dimension: {4}\n" +  
 "Tables Sourced: {5}\n" +  
 "Value Column: {6}\n" +  
 "Relative Columns: {7}\n" +  
 "Safety: {8}\n"  
 ;  
 public void DisplayInfo()  
 {  
 string info = String.Format(default\_info, DataSetType, dataset.Count(), dataset.First().Item1.dimension, InputCol, dataset.First().Item2.dimension, TableSourced, ValueCol, RelativeCol, Safety);  
 InformationTB.Text = info;  
 }  
  
 public void PlotCountPoints(int b, int s, int h){  
 lock (DataChartLock)  
 {  
 DataChart.Series["Count"].Points.AddXY("Buy", b);  
 DataChart.Series["Count"].Points.AddXY("Sell", s);  
 DataChart.Series["Count"].Points.AddXY("Hold", h);  
 }  
 }  
  
 public void PlotInfo()  
 {  
 lock (DataChartLock)  
 {  
 DataChart.Series["Count"].Points.Clear();  
 }  
 BackgroundWorker plotCountWorker = new BackgroundWorker();  
 plotCountWorker.DoWork += (sender, e) =>  
 {  
 Tuple<List<(Vector, Vector)>, IDataSet, Chart> tup = e.Argument as Tuple<List<(Vector, Vector)>, IDataSet, Chart>;  
  
 List<(Vector, Vector)> \_TableInfo = tup.Item1;  
 IDataSet \_dataset = tup.Item2;  
 Chart \_DataChart = tup.Item3;  
  
 int buy = 0;  
 int sell = 0;  
 int hold = 0;  
 int total = 0;  
 int looped = 0;  
   
 //Action action = () =>  
 //{  
 lock (TableInfoLock) {  
 \_TableInfo.Clear();  
 foreach ((Vector, Vector) v in \_dataset.Batch(\_dataset.Count()))  
 {  
 \_TableInfo.Add(v);  
  
 if (v.Item2[0] == 1)  
 {  
 buy++;  
 total++;  
 }  
 else if (v.Item2[0] == -1)  
 {  
 sell++;  
 total++;  
 }  
 else if (v.Item2[0] == 0)  
 {  
 hold++;  
 total++;  
 }  
  
 looped++;  
  
 }  
 }  
 //var safeInvoker = new CountReceivedDelegate(PlotCountPoints);  
 //Invoke(safeInvoker, buy, sell, hold);  
 PlotCountPoints(buy,sell,hold);  
  
  
 e.Result = new Tuple<List<(Vector, Vector)>, IDataSet, Chart>(\_TableInfo, \_dataset, \_DataChart);  
  
  
 //\_DataChart.Update();  
 //};  
 //this.Invoke(action);  
 };  
  
 plotCountWorker.RunWorkerCompleted += (sender, e) =>  
 {  
 ChartLoadingLabel.Hide();  
 Tuple<List<(Vector, Vector)>, IDataSet, Chart> tup = e.Result as Tuple<List<(Vector, Vector)>, IDataSet, Chart>;  
 lock (DataChartLock) {  
 DataChart = tup.Item3;  
 }  
 //dataset doesnt need to change  
 lock (TableInfoLock) {  
 TableInfo = tup.Item1; //IT WORKS MashAllah !  
 }  
 };  
  
 ChartLoadingLabel.Show();  
 plotCountWorker.RunWorkerAsync(argument: new Tuple<List<(Vector, Vector)>, IDataSet, Chart>(TableInfo, dataset, DataChart));  
 //fillInformationTAble();  
  
 }  
 //void plotInfoDb()  
 //{  
 // //TableInfo.Clear();  
 // int buy = 0;  
 // int sell = 0;  
 // int hold = 0;  
 // int total = 0;  
 // int looped = 0;  
 // ((DB\_DataSet)dataset).ToListComplete += (object? s, List<(Vector, Vector)> e) =>  
 // {  
 // TableInfo = e;  
 // Thread t = new Thread(() =>  
 // {  
 // foreach ((Vector, Vector) v in TableInfo)  
 // {  
  
 // if (v.Item2[0] == 1)  
 // {  
 // buy++;  
 // total++;  
 // }  
 // else if (v.Item2[0] == -1)  
 // {  
 // sell++;  
 // total++;  
 // }  
 // else if (v.Item2[0] == 0)  
 // {  
 // hold++;  
 // total++;  
 // }  
  
 // looped++;  
  
 // }  
  
  
 // DataChart.Series["Count"].Points.AddXY("Buy", buy);  
 // DataChart.Series["Count"].Points.AddXY("Sell", sell);  
 // DataChart.Series["Count"].Points.AddXY("Hold", hold);  
 // Update();  
 // }  
 // );  
 // t.Start();  
 // };  
 // ((DB\_DataSet)dataset).ToList();  
 //}  
 void addRowToTable(Vector i, Vector o)  
 {  
 Invoke(() =>  
 {  
 Table.RowCount += 1;  
 Label l2 = new Label();  
 l2.BackColor = Color.Transparent;  
 l2.AutoSize = true;  
 l2.Text = "{" + String.Join(", ", i) + "}";  
 Table.Controls.Add(l2);  
 //Table.SetColumn(l2, 0);  
  
  
 Label l = new Label();  
 l.BackColor = Color.Transparent;  
 l.AutoSize = true;  
 l.Text = "{" + String.Join(", ", o) + "}";  
 Table.Controls.Add(l);  
 });  
 }  
  
 void fillInformationTAble()  
 {  
 //BackgroundWorker FillTableWorker = new BackgroundWorker();  
  
   
  
  
 //FillTableWorker.DoWork += (sender, e) =>  
 //{  
 //Tuple<List<(Vector, Vector)>, TableLayoutPanel> tup = e.Argument as Tuple<List<(Vector, Vector)>, TableLayoutPanel>;  
 //List<(Vector, Vector)> TableInfo = tup.Item1;  
 //TableLayoutPanel \_Table = tup.Item2;  
 int count = 0;  
 //Label label = new Label();  
 //label.AutoSize=true;  
 //string text = "FIRST 50 RECORDS\n";  
 Table.Controls.Clear();  
 Table.RowCount = 0;  
 lock (TableInfoLock) {  
 foreach ((Vector, Vector) v in TableInfo)  
 {  
 Table.RowCount += 1;  
 Label l2 = new Label();  
 l2.BackColor = Color.Transparent;  
 l2.AutoSize = true;  
 l2.Text = "{" + String.Join(", ", v.Item1) + "}";  
   
 //Table.SetColumn(l2, 0);  
  
  
 Label l = new Label();  
 l.BackColor = Color.Transparent;  
 l.AutoSize = true;  
 l.Text = "{" + String.Join(", ", v.Item2) + "}";  
  
 //\_Table.Controls.Add(l);  
 //\_Table.Controls.Add(l2);  
  
 if (InvokeRequired)  
 {  
 Invoke((MethodInvoker)delegate ()  
 {  
 Table.Controls.Add(l);  
 Table.Controls.Add(l2);  
  
 });  
 }  
 count++; if (count > 50) break;  
 }  
 }  
  
 //for (int r = 0; r < Table.RowCount; r++)  
 //{  
 // for (int c = 0; c < Table.ColumnCount; c++)  
 // {  
 // if (r % 2 == 1)  
 // {  
 // Control con = this.Table.GetControlFromPosition(c, r);  
  
 // if (con != null)  
 // {  
 // TableLayoutPanelCellPosition l = new TableLayoutPanelCellPosition();  
  
 // }  
 // }  
  
 // }  
 //}  
  
 if (InvokeRequired)  
 {  
 Invoke((MethodInvoker)delegate ()  
 {  
 Table.AutoScroll = true;  
 Table.RowStyles.Clear();  
 });  
 }  
  
  
  
 //};  
  
 //FillTableWorker.RunWorkerCompleted += (sender, e) =>  
 //{  
  
 //};  
  
 //FillTableWorker.RunWorkerAsync(argument: new Tuple<List<(Vector,Vector)>, TableLayoutPanel>(TableInfo,Table));  
 }  
  
 public DataSetViewer()  
 {  
 InitializeComponent();  
 }  
  
 public void ProgramaticAssign( string path )  
 {//the . is needed here  
   
 string c = Path.GetExtension( path );  
 switch (c)  
 {  
 case ".ads":  
 dataset = ArrayDataSet.Load(path);  
 DataSetType = "Array";  
 InputCol = "N/A";  
 TableSourced = "N/A";  
 ValueCol = "N/A";  
 RelativeCol = "N/A";  
 Safety = "N/A";  
 break;  
 case ".ds":  
 dataset = NeaLibrary.DataStructures.DataSet.Load(path);  
 DataSetType = "Standard";  
 InputCol = "N/A";  
 TableSourced = "N/A";  
 ValueCol = "N/A";  
 RelativeCol = "N/A";  
 Safety = "N/A";  
 break;  
 case ".dbds":  
 dataset = DB\_DataSet.Load(path);  
 DataSetType = "Database";  
 InputCol = String.Join(", ", ((DB\_DataSet)dataset).cols);  
 TableSourced = String.Join(", ", ((DB\_DataSet)dataset).tables);  
 ValueCol = String.Join(", ", ((DB\_DataSet)dataset).ValueColumn);  
 RelativeCol = String.Join(", ", ((DB\_DataSet)dataset).relative\_cols);  
 Safety = ((DB\_DataSet)dataset).safety.ToString();  
 //stockChart();  
 DB\_DataSet\_StockView\_B.Visible = true;  
  
 break;  
 }  
 SourceTB.Text = Path.GetFileName(path);  
 DisplayInfo();  
 PlotInfo();  
 }  
  
 private void CangeSourceB\_Click(object sender, EventArgs e)  
 {  
 OpenFileDialog OpenDatafileDialogue = new OpenFileDialog();  
 OpenDatafileDialogue.Multiselect = false;  
 OpenDatafileDialogue.Filter = "Supported Encodings | \*.ads;\*.ds;\*.dbds";  
 DialogResult result = OpenDatafileDialogue.ShowDialog();  
 if (result == DialogResult.OK && !string.IsNullOrWhiteSpace(OpenDatafileDialogue.FileName))  
 {  
 //.ads ArrayDataSet  
 //.ds DataSet  
 //.dbds DB\_DataSet  
 string c = OpenDatafileDialogue.FileName.Split('.').Last();  
 switch (c)  
 {  
 case "ads":  
 dataset = ArrayDataSet.Load(OpenDatafileDialogue.FileName);  
 DataSetType = "Array";  
 InputCol = "N/A";  
 TableSourced = "N/A";  
 ValueCol = "N/A";  
 RelativeCol = "N/A";  
 Safety = "N/A";  
 break;  
 case "ds":  
 dataset = NeaLibrary.DataStructures.DataSet.Load(OpenDatafileDialogue.FileName);  
 DataSetType = "Standard";  
 InputCol = "N/A";  
 TableSourced = "N/A";  
 ValueCol = "N/A";  
 RelativeCol = "N/A";  
 Safety = "N/A";  
 break;  
 case "dbds":  
 dataset = DB\_DataSet.Load(OpenDatafileDialogue.FileName);  
 DataSetType = "Database";  
 InputCol = String.Join(", ", ((DB\_DataSet)dataset).cols);  
 TableSourced = String.Join(", ", ((DB\_DataSet)dataset).tables);  
 ValueCol = String.Join(", ", ((DB\_DataSet)dataset).ValueColumn);  
 RelativeCol = String.Join(", ", ((DB\_DataSet)dataset).relative\_cols);  
 Safety = ((DB\_DataSet)dataset).safety.ToString();  
 //stockChart();  
 DB\_DataSet\_StockView\_B.Visible = true;  
  
 break;  
 }  
 SourceTB.Text = OpenDatafileDialogue.FileName;  
 DisplayInfo();  
 PlotInfo();  
 //fillInformationTAble();  
 //if (dataset is DB\_DataSet)  
 //{  
 // plotInfoDb();  
 //}  
 //else  
 //{  
 // PlotInfoNonDb();  
 //}  
 }  
 }  
 //StockChart stockChart()  
 //{  
 // StockChart sc = new StockChart();  
 // //this.Controls.Add(sc);  
 // //this.Controls[sc.Name].BringToFront();  
 // sc.dataset = (DB\_DataSet)dataset;  
 // sc.InitialiseCharts();  
 // sc.SetTokens(sc.dataset.tables);  
 // return sc;  
 //}  
  
 private void Table\_Paint(object sender, PaintEventArgs e)  
 {  
 //Table.CellPaint += colourRows;  
 //void colourRows(object? sender, TableLayoutCellPaintEventArgs e)  
 //{//https://stackoverflow.com/questions/34064499/how-to-set-cell-color-in-tablelayoutpanel-dynamically  
 // if ((e.Row) % 2 == 1) e.Graphics.FillRectangle(Brushes.LightSteelBlue, e.CellBounds);  
 //}  
 }  
  
 private void DB\_DataSet\_StockView\_B\_Click(object sender, EventArgs e)  
 {  
 //DB\_DataSet\_StockView\_B.Visible = false;  
 EventArgs ev = new EventArgs();  
 RequestTabForStockChart(this, (DB\_DataSet)dataset);  
 }  
  
 private void CloseButton\_Click(object sender, EventArgs e)  
 {  
 Exit(this, e);  
 }  
 }  
}

## .\WinFormsApp2\DataControls/FetchDataControl.cs

﻿  
using NeaLibrary.Data;  
using NeaLibrary.Data.Other;  
//using System.Text.Json.Nodes;  
using Newtonsoft.Json.Linq;  
using System.ComponentModel;  
using System.Data.SQLite;  
using System.Text.RegularExpressions;  
  
namespace WinFormsApp2.DataControls  
{  
 public partial class FetchDataControl : UserControl  
 {  
 MultiMap<int, string> sourcesInfo = new MultiMap<int, string>();  
 Regex parameterMatcher = new Regex(@"\{(0|[1-9][0-9]\*)\}"); //tested and verified on regex storm net tester  
 //matches curly brackets with natural number including 0 inside. does NOT match {000123} which is correct  
 //does NOT match {-123} which is correct  
 Dictionary<string, string> ArgumentEditorControlName\_to\_Values = new Dictionary<string, string>();  
 Dictionary<string, ArgumentEditorControl> CurlyBracesFromParameters\_to\_ArgumentEditorControl = new Dictionary<string, ArgumentEditorControl>();  
 string empty\_request;  
 string final;  
  
 int source;  
 List<string> for\_worker\_args = new List<string>();  
  
 List<BackgroundWorker> workers = new List<BackgroundWorker>();  
 object manageWorkersLock = new object();  
  
 public event EventHandler FinishedCreating;  
  
 //BackgroundWorker fetcher\_worker = new BackgroundWorker();  
 //Queue<>  
 public FetchDataControl()  
 {  
 InitializeComponent();  
 loadSources();  
 }  
  
 void loadSources()  
 {  
 sourcesInfo.Clear();  
 SourceSelectorUpDown.Items.Clear();  
 SQL\_Driver sQL\_Driver = new SQL\_Driver(NeaLibrary.Tools.Tools.GetGlobalVar("db\_dir"));  
 using (SQLiteDataReader r = SQL\_Driver.Query(sQL\_Driver.conn, "SELECT \* FROM Sources"))  
 {  
 while (r.Read())  
 {  
 try  
 {  
 int sourceId = r.GetInt32(0);  
 sourcesInfo.Add(sourceId);  
 for (int i = 1; i < r.FieldCount; i++)  
 {  
 //Type t = r.GetFieldType(i);  
 object v = r.GetValue(i);  
 sourcesInfo.Add(sourceId, v.ToString()); // sources v should never be null, ignore warning  
 }  
 SourceSelectorUpDown.Items.Add(sourceId);  
 }  
 catch  
 {  
 continue;  
 }  
 }  
 }  
 }  
  
 private void OnValueChnaged(object sender, EventArgs e)  
 {  
 final = empty\_request;  
 //values[((ArgumentEditorControl)sender).Name] = ((ArgumentEditorControl)sender).GetInfo();  
 foreach (string k in ArgumentEditorControlName\_to\_Values.Keys)  
 {  
 ArgumentEditorControlName\_to\_Values[k] = ((ArgumentEditorControl)CurlyBracesFromParameters\_to\_ArgumentEditorControl[k]).GetInfo();  
 if (ArgumentEditorControlName\_to\_Values[k] != "") final = final.Replace(k, ArgumentEditorControlName\_to\_Values[k]);  
 }  
 richTextBox1.Text = final.ToString();  
 richTextBox1.Invalidate();  
 for\_worker\_args.Clear();  
 for (int i = 0; i < ArgumentEditorControlName\_to\_Values.Count; i++)  
 {  
 try  
 {  
 for\_worker\_args.Add(ArgumentEditorControlName\_to\_Values[$"{{{i}}}"]);  
 }  
 catch (KeyNotFoundException)  
 {  
 break;  
 }  
 }  
 }  
  
 private void SourceSelectorUpDown\_SelectedItemChanged(object sender, EventArgs e)  
 {  
 CurlyBracesFromParameters\_to\_ArgumentEditorControl.Clear();  
 //selected item can not be null if this called  
 if (SourceSelectorUpDown.SelectedItem == null) throw new Exception("No item is selected");  
 source = (int)SourceSelectorUpDown.SelectedItem;  
 ArgumentP.Controls.Clear();  
 ArgumentEditorControlName\_to\_Values.Clear();  
 ArgumentP.RowCount = 1;  
 ArgumentP.ColumnCount = 1;  
 string sourcepinfo = sourcesInfo[(int)SourceSelectorUpDown.SelectedItem][7];  
 Dictionary<int, SourceParameterInfo> sp = new Dictionary<int, SourceParameterInfo>();  
 List<int> covered\_args = new List<int>();  
 if (sourcepinfo != "{}")  
 {  
 JObject temp = JObject.Parse(sourcepinfo);  
 foreach (KeyValuePair<string, JToken?> o in temp)  
 {  
 //key should be the parameter position  
 try  
 {  
 if (o.Value == null) continue; //the try loops should save us but defensive programming is good   
 SourceParameterInfo t = o.Value.ToObject<SourceParameterInfo>(); //warning if he item can not be converted, if thats the case exception is handled by try statemnet  
 sp.Add(Convert.ToInt32(o.Key), t);  
 //control\_tracker  
 }  
 catch { }  
  
 }  
 }  
 foreach (Match m in parameterMatcher.Matches(sourcesInfo[(int)SourceSelectorUpDown.SelectedItem][1]))  
 {  
 SourceParameterInfo DEFAULT = new SourceParameterInfo();  
 DEFAULT.Name = m.Value;  
 DEFAULT.IsRegex = false;  
 DEFAULT.IsCategorical = false;  
 DEFAULT.Regex = "";  
 DEFAULT.Categories = new string[0];  
 ArgumentEditorControl a = new ArgumentEditorControl();  
 //a.ValueChanged += OnValueChnaged;  
 ArgumentP.RowCount++;  
 int n = Convert.ToInt32(m.Value.Substring(1, m.Value.Length - 2));  
 if (sp.ContainsKey(n) && !covered\_args.Contains(n))  
 {  
 a.setArgumentLabelText($"{m.Value} => {sp[n].Name}");  
 a.SetInfo(sp[n]);  
 covered\_args.Add(n);  
  
 a.Name = m.Value;  
 ArgumentEditorControlName\_to\_Values.Add(a.Name, "");  
 ArgumentP.Controls.Add(a, 0, ArgumentP.RowCount - 1);  
 a.Show();  
 a.ValueChanged += OnValueChnaged;  
 CurlyBracesFromParameters\_to\_ArgumentEditorControl.Add(m.Value, a);  
 }  
 else if (!sp.ContainsKey(n) && !covered\_args.Contains(n))  
 {  
 a.setArgumentLabelText(m.Value);  
 a.SetInfo(DEFAULT);  
 covered\_args.Add(n);  
  
 a.Name = m.Value;  
 ArgumentEditorControlName\_to\_Values.Add(a.Name, "");  
 ArgumentP.Controls.Add(a, 0, ArgumentP.RowCount - 1);  
 a.Show();  
 a.ValueChanged += OnValueChnaged;  
 CurlyBracesFromParameters\_to\_ArgumentEditorControl.Add(m.Value, a);  
 }  
 //if (!covered\_args.Contains(n))  
 //{  
  
 //}  
 }  
 Label label = new Label();  
 label.AutoSize = true;  
 label.UseMnemonic = false;  
 label.Text = String.Join("", sourcesInfo[(int)SourceSelectorUpDown.SelectedItem][0], sourcesInfo[(int)SourceSelectorUpDown.SelectedItem][1]);  
 empty\_request = String.Join("", sourcesInfo[(int)SourceSelectorUpDown.SelectedItem][0], sourcesInfo[(int)SourceSelectorUpDown.SelectedItem][1]);  
 ArgumentP.Controls.Add(label, 0, 0);  
 label.Show();  
 ArgumentP.RowStyles.Clear();  
 }  
  
 BackgroundWorker ManageWorkers()  
 {  
 lock (manageWorkersLock)  
 {  
 List<BackgroundWorker> to\_free = new List<BackgroundWorker>();  
 bool one\_unbusy = false;  
 BackgroundWorker worker = null;  
 if (workers.Count != 0)  
 {  
 foreach (BackgroundWorker w in workers)  
 {  
 if (w.IsBusy)  
 {  
  
 }  
 else  
 {  
 if (one\_unbusy)  
 {  
 to\_free.Add(w);  
 }  
 else  
 {  
 worker = w;  
 }  
 one\_unbusy = true;  
 }  
 }  
 foreach (BackgroundWorker w in to\_free)  
 {  
 w.Dispose();  
 workers.Remove(w);  
 }  
 }  
 if (!one\_unbusy)  
 {  
 worker = new BackgroundWorker();  
 workers.Add(worker);  
 }  
 return worker!; //shouldnt be null by this point  
 }  
 }  
  
 private void SubmitB\_Click(object sender, EventArgs e)  
 {  
 ManageWorkers();  
  
 BackgroundWorker fetcher\_worker = ManageWorkers();  
 fetcher\_worker.DoWork += workerWork;  
 fetcher\_worker.ProgressChanged += (sender, e) =>  
 {  
 //CandleChart = e.Result as Chart;  
  
 };  
 fetcher\_worker.RunWorkerCompleted += (sender, e) =>  
 {  
 if (e.Error != null)  
 {  
 MessageBox.Show("Error encountered while fetching: " + e.Error.ToString());  
 }  
 };  
  
 if (!fetcher\_worker.IsBusy)  
 {  
 //SubmitB.Enabled = false;  
 fetcher\_worker.RunWorkerAsync(argument: new Tuple<int, List<string>, bool>(source, for\_worker\_args, UseCalcCB.Checked));  
 }  
 else MessageBox.Show("Worker busy before even assigned work, you can't have reached this point");  
 }  
 void workerWork(object s, DoWorkEventArgs e)  
 {  
 Tuple<int, List<string>, bool> h = e.Argument as Tuple<int, List<string>, bool>;  
 NeaLibrary.Data.Data\_Handler dh = new Data\_Handler();  
 string? table;  
 try { table = dh.Fetch(h.Item1, h.Item2.ToArray()); }  
 catch (NullReferenceException ex)  
 {  
 throw new Exception("Null Reference Encountered, perhaps this is not a valid request." +  
 "\nPlease check input" + ex.ToString());  
 }  
 if (h.Item3)  
 {  
 //calc  
 try  
 {  
 //shouldnt be by this point but check so compiler is happy  
 if (table != null) Calculator.Calculate(table);  
 }  
 catch (Exception ex)  
 {  
 throw new Exception("Calculator error. Other values were saved." +  
 "\nPlease check DB and Calculator" + ex.ToString());  
 }  
 }  
 }  
  
 private void ArgumentP\_Paint(object sender, PaintEventArgs e)  
 {  
  
 }  
  
 private void richTextBox1\_TextChanged(object sender, EventArgs e)  
 {  
 }  
  
 private void label2\_Click(object sender, EventArgs e)  
 {  
 }  
  
 private void ExitB\_Click(object sender, EventArgs e)  
 {  
 lock (manageWorkersLock)  
 {  
  
 if (workers.Any((worker) => worker.IsBusy)) { MessageBox.Show("Unfinished workers"); return; }  
 try { FinishedCreating(this, EventArgs.Empty); }  
 catch (Exception ex)  
 {  
 MessageBox.Show(ex.Message);  
 }  
 }  
 }  
 }  
}

## .\WinFormsApp2\DataControls/MakePredictionControl.cs

﻿using NeaLibrary.Data;  
using NeaLibrary.Data.Other;  
using NeaLibrary.DataStructures;  
using NeaLibrary.NeuralNetwork;  
using NeaLibrary.NeuralNetwork.FFNN;  
using NeaLibrary.NeuralNetwork.NEAT;  
using NeaLibrary.Tools;  
  
namespace WinFormsApp2.DataControls  
{  
 public partial class MakePredictionControl : UserControl  
 {  
 public event EventHandler Exit;  
 public event EventHandler<Vector> VectorViewRequested;  
 string neuralnetworktype = "";  
 INeuralNetwork? model;  
 NNSpecification? NNSpec;  
 string info\_template =  
"""  
Neural Network Type: {0}  
Count: {1}  
Last trained: {2}  
  
Mapping  
{3}  
""";  
 public MakePredictionControl()  
 {  
 InitializeComponent();  
 }  
  
 private void MakePredictionB\_Click(object sender, EventArgs e)  
 {  
 if (NNSpec == null || model == null) return;  
 //for (int i =0; i<NNSpec.InputDimension; i++)  
 //{  
 // InputTablePanel;  
 //}  
 Vector result = new Vector(NNSpec.OutputDimension);  
 Vector input = new Vector(NNSpec.InputDimension);  
 foreach (ArgumentEditorControl control in InputTablePanel.Controls)  
 {  
 if (control != null)  
 {  
 int n = Convert.ToInt32(control.Name);  
 input[n] = Convert.ToDouble(control.GetValue());  
  
 }  
 }  
 Vector output = model.BestPrediction(input);  
  
 if (output.dimension == 1)  
 {  
 double pred = output[0];  
 //pred should be in [-1,1] but to ensure this putthrough ReLu2  
 pred = Tools.ReLu2(pred);  
 PredictionLabel.Text = pred.ToString();  
 TrackBar.Value = (int)(pred \* 10);  
 }  
 else  
 {  
 VectorViewRequested(this, output);  
 }  
 }  
  
 private void LoadInputArgumentEditorControls()  
 {  
 if (model != null)  
 {  
 if (neuralnetworktype == "FFNN")  
 {  
 NNSpec = model.GetNNSpecification();  
 }  
 else  
 {  
 NNSpec = model.GetNNSpecification();  
 }  
 InfoLabel.Text = String.Format(info\_template, neuralnetworktype, "", ((NNSpec.LastTrainedOn != null) ? NNSpec.LastTrainedOn : "uknown"), ((NNSpec.InputMapCacheDescription != null) ? NNSpec.InputMapCacheDescription.ToString() : ""));  
 InputTablePanel.Controls.Clear();  
 InputTablePanel.RowStyles.Clear();  
 InputTablePanel.ColumnStyles.Clear();  
 for (int i = 0; i < NNSpec.InputDimension; i++)  
 {  
 var ArgumentControl = new ArgumentEditorControl();  
 SourceParameterInfo sp = new SourceParameterInfo();  
 sp.Name = $"input n.{i.ToString()}";  
 ArgumentControl.setArgumentLabelText(sp.Name);  
 sp.IsNumeric = true;  
 sp.BoundsAndStep = new double[] { -1e6, +1e6, 1e-3 };  
  
 ArgumentControl.SetInfo(sp);  
 ArgumentControl.Name = i.ToString();  
 InputTablePanel.Controls.Add(ArgumentControl, 0, i);  
 }  
 }  
 }  
  
 private void ChangeB\_Click(object sender, EventArgs e)  
 {  
 OpenFileDialog OpenSourceDialogue = new OpenFileDialog();  
 OpenSourceDialogue.Multiselect = false;  
 OpenSourceDialogue.Filter = "Neural Network | \*.neat;\*.ffnn";  
 DialogResult result = OpenSourceDialogue.ShowDialog();  
 if (result == DialogResult.OK && !string.IsNullOrWhiteSpace(OpenSourceDialogue.FileName))  
 {  
 string c = OpenSourceDialogue.FileName.Split('.').Last();  
 int count = -1;  
 switch (c)  
 {  
 case "neat":  
 neuralnetworktype = "NEAT";  
 model = NEAT\_Handler.Load(OpenSourceDialogue.FileName);//new NEAT\_Handler(Neat.Load(OpenSourceDialogue.FileName));  
 count = ((NEAT\_Handler)model).NumberOfCreatures();  
 break;  
 case "ffnn":  
 neuralnetworktype = "FFNN";  
 model = FFNN\_Client.Load(OpenSourceDialogue.FileName);  
 count = ((FFNN\_Client)model).GetClients().Count;  
 break;  
 }  
 SourceTB.Text = OpenSourceDialogue.FileName;  
 LoadInputArgumentEditorControls();  
 }  
 }  
  
 private void PredictionLabel\_Click(object sender, EventArgs e)  
 {  
  
 }  
  
 private void label3\_Click(object sender, EventArgs e)  
 {  
 }  
  
 private void InputTablePanel\_Paint(object sender, PaintEventArgs e)  
 {  
 }  
  
 private void ExitB\_Click(object sender, EventArgs e)  
 {  
 Exit(this, EventArgs.Empty);  
 }  
  
 private void InfoLabel\_Click(object sender, EventArgs e)  
 {  
  
 }  
 }  
}

## .\WinFormsApp2\DataControls/StockChart.cs

﻿using NeaLibrary.Data;  
using Skender.Stock.Indicators;  
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Data.SQLite;  
using System.Drawing;  
using System.Linq;  
using System.Security.Policy;  
using System.Text;  
using System.Threading.Tasks;  
using System.Windows.Forms;  
using System.Windows.Forms.DataVisualization.Charting;  
using NeaLibrary.DataStructures;  
  
using System.Reflection;  
  
namespace WinFormsApp2.DataControls  
{  
 public partial class StockChart : UserControl  
 {  
 public const int CHUNK\_SIZE = 30;  
 string SQLCommand;  
 public DB\_DataSet dataset;  
 List<string> choices = new List<string>();  
 public delegate void OnPointToBePlotted(string table, Quote q, double marker);  
 public event EventHandler Exit;  
 public StockChart()  
 {  
 InitializeComponent();  
 }  
  
 public void setDataset(DB\_DataSet ds)  
 {  
 dataset = ds;  
 }  
 public void InitialiseCharts()  
 {  
 //CandleChart.Series.Add("Price");  
 //CandleChart.Series["Price"].AxisLabel =   
 ClearChart();  
 CandleChart.ChartAreas["ChartArea1"].AxisX.ScrollBar.Enabled = true;  
 CandleChart.ChartAreas["ChartArea1"].AxisX.IntervalAutoMode = IntervalAutoMode.VariableCount;  
  
 //CandleChart.ChartAreas["ChartArea1"].AxisX2.Enabled = AxisEnabled.True;  
 //CandleChart.ChartAreas["ChartArea1"].AxisX2.MajorGrid.Enabled = false;  
 //CandleChart.ChartAreas["ChartArea1"].AxisX2.MinorGrid.Enabled = false;  
  
 CandleChart.ChartAreas["ChartArea1"].AxisY.IsStartedFromZero = false;  
 CandleChart.ChartAreas["ChartArea1"].AxisX.LabelStyle.Format = "dd-MM-yy";  
 CandleChart.ChartAreas["ChartArea1"].AxisX.Interval = 1;  
 CandleChart.ChartAreas["ChartArea1"].AxisX.IntervalType = DateTimeIntervalType.Auto;  
  
  
 //CandleChart.ChartAreas["ChartArea1"].AxisX2.Interval = 1;  
 //CandleChart.ChartAreas["ChartArea1"].AxisX2.IsInterlaced = false;  
 //CandleChart.ChartAreas["ChartArea1"].AxisX2.InterlacedColor = Color.Transparent;  
 //CandleChart.ChartAreas["ChartArea1"].AxisX2.IntervalAutoMode = IntervalAutoMode.VariableCount;  
  
 CandleChart.ChartAreas["ChartArea1"].AxisX2.IntervalType = DateTimeIntervalType.Auto;  
  
 // enable autoscroll  
 CandleChart.ChartAreas["ChartArea1"].CursorX.AutoScroll = true;  
  
 // let's zoom to [0,blockSize] (e.g. [0,100])  
 CandleChart.ChartAreas["ChartArea1"].AxisX.ScaleView.Zoomable = true;  
 CandleChart.ChartAreas["ChartArea1"].AxisX.ScaleView.SizeType = DateTimeIntervalType.Number;  
 //CandleChart.ChartAreas["ChartArea1"].AxisX2.ScaleView.Zoomable = true;  
 //CandleChart.ChartAreas["ChartArea1"].AxisX2.ScaleView.SizeType = DateTimeIntervalType.Number;  
 int position = 0;  
 int size = 30;  
 CandleChart.ChartAreas["ChartArea1"].AxisX.ScaleView.Zoom(position, size);  
 //CandleChart.ChartAreas["ChartArea1"].AxisX2.ScaleView.Zoom(position, size);  
  
 // disable zoom-reset button (only scrollbar's arrows are available)  
 CandleChart.ChartAreas["ChartArea1"].AxisX.ScrollBar.ButtonStyle = ScrollBarButtonStyles.SmallScroll;  
 //CandleChart.ChartAreas["ChartArea1"].AxisX2.ScrollBar.ButtonStyle = ScrollBarButtonStyles.None;  
  
  
 // set scrollbar small change to blockSize (e.g. 100)  
 CandleChart.ChartAreas["ChartArea1"].AxisX.ScaleView.SmallScrollSize = 30;  
 //CandleChart.ChartAreas["ChartArea1"].AxisX2.ScaleView.SmallScrollSize = 30;  
  
 //CandleChart.ChartAreas["ChartArea1"].AxisX2.StripLines.Clear();  
  
  
 //CandleChart.Series["Price"].ChartType = SeriesChartType.Candlestick;  
  
 //CandleChart.Series["Price"]["OpenCloseStyle"] = "Triangle";  
  
 //CandleChart.Series["Price"]["ShowOpenClose"] = "Both";  
  
 //CandleChart.Series["Price"]["PointWidth"] = "1.0";  
  
 //CandleChart.Series["Price"]["PriceUpColor"] = "Green";  
 //CandleChart.Series["Price"]["PriceDownColor"] = "Red";  
  
 IndicatorChart.ChartAreas["ChartArea1"].AxisX.ScrollBar.Enabled = true;  
  
 }  
  
 void ClearChart()  
 {  
 if (CandleChart == null) return;  
 Invoke(() => { CandleChart.Series.Clear(); });  
 }  
  
  
 void PlotPoint(string table, Quote q, double marker)  
 {  
 // adding date and high  
 int i = CandleChart.Series[table].Points.AddXY(q.Date, q.High);  
 // adding low  
 CandleChart.Series[table].Points[i].YValues[1] = (double)q.Low;  
 //adding open  
 CandleChart.Series[table].Points[i].YValues[2] = (double)q.Open;  
 // adding close  
 CandleChart.Series[table].Points[i].YValues[3] = (double)q.Close;  
 //plotpoint(table, q);  
 CandleChart.Series[table + " Close"].Points.AddXY(q.Date, q.Close);  
  
 if (marker == 1)  
 {  
 //buy  
 CandleChart.Series[table + " Close"].Points[i].MarkerStyle = MarkerStyle.Circle;  
 CandleChart.Series[table + " Close"].Points[i].MarkerColor = Color.Green;  
 CandleChart.Series[table + " Close"].Points[i].MarkerSize = 10;  
 }  
 else if (marker == -1)  
 {  
 //sell  
 CandleChart.Series[table + " Close"].Points[i].MarkerStyle = MarkerStyle.Circle;  
 CandleChart.Series[table + " Close"].Points[i].MarkerColor = Color.Red;  
 CandleChart.Series[table + " Close"].Points[i].MarkerSize = 10;  
 }  
 }  
  
  
 public void PlotAsset(string table)  
 {  
 if (PlotterWorker.IsBusy) return;  
 if (CandleChart.Series.IsUniqueName(table))  
 {  
 CandleChart.Series.Add(table);  
 CandleChart.Series.Add(table + " Close");  
 }  
 //new Series()  
  
 CandleChart.Series[table].ChartType = SeriesChartType.Candlestick;  
 CandleChart.Series[table + " Close"].ChartType = SeriesChartType.Line;  
  
 CandleChart.Series[table]["OpenCloseStyle"] = "Triangle";  
  
 CandleChart.Series[table]["ShowOpenClose"] = "Both";  
  
 CandleChart.Series[table]["PointWidth"] = "0.9";  
  
 CandleChart.Series[table]["PriceUpColor"] = "Green";  
 CandleChart.Series[table]["PriceDownColor"] = "Red";  
  
 CandleChart.Series[table].BorderWidth = 3;  
 //CandleChart.Series[table]. = 3;  
 //CandleChart.Series[table].BorderColor = Color.Black;  
  
  
 //CandleChart.Series[table].AxisLabel = "Date";  
  
 CandleChart.Series[table].SetCustomProperty("PriceUpColor", "Green");  
 CandleChart.Series[table].SetCustomProperty("PriceDownColor", "Red");  
  
  
 PlotterWorker.RunWorkerCompleted += (sender, e) =>  
 {  
 if (e.Error != null) return;  
 CandleChart = e.Result as Chart;  
 if (CandleChart == null) { MessageBox.Show($"check if data table has missing values"); return; }  
 CandleChart.Update();  
 };  
  
 try  
 {  
 PlotterWorker.RunWorkerAsync(argument: new Tuple<Chart, string, DB\_DataSet>(CandleChart, table, dataset));  
   
 }  
 catch(Exception e)  
 {  
 MessageBox.Show($"Check if some tables have missing data \n{e.Message}");  
 }  
  
 //Thread t = new Thread(() =>  
 //{  
 // int i = 0;  
  
 // foreach (Quote q in SQL\_Driver.ReadRow\_AsQuote(dataset.driver.conn, table, new List<string> { "Date", "open", "high", "low", "close", "volume" }))  
 // {  
 // // adding date and high  
 // CandleChart.Series[table].Points.AddXY(q.Date, q.High);  
 // // adding low  
 // CandleChart.Series[table].Points[i].YValues[1] = (double)q.Low;  
 // //adding open  
 // CandleChart.Series[table].Points[i].YValues[2] = (double)q.Open;  
 // // adding close  
 // CandleChart.Series[table].Points[i].YValues[3] = (double)q.Close;  
 // i++;  
 // if (i > 50) break;  
 // }  
 // Update();  
 //}); t.Start();  
  
 }  
  
 public void PlotPrice()  
 {  
  
 }  
  
  
 private void SQLCommandTB\_TextChanged(object sender, EventArgs e)  
 {  
  
 }  
 private void ChangeSQlTB(string s)  
 {  
 Invoke(() => SQLCommandTB.Text = s);  
 }  
 private void ChangeRecordsPlotted(int s)  
 {  
 Invoke(() => TotalRecordsTB.Text = s.ToString());  
 }  
 private void PlotterWorkerWork(object sender, DoWorkEventArgs e)  
 {  
  
 Tuple<Chart, string, DB\_DataSet> tup = e.Argument as Tuple<Chart, string, DB\_DataSet>; // uhhh  
 if (tup != null)  
 {  
 string table = tup.Item2;  
  
  
 Chart MyChart = tup.Item1;  
 DB\_DataSet dataset = tup.Item3;  
 SQLiteConnection conn = dataset.driver.conn;  
  
 ChangeSQlTB($"SELECT {String.Join(", ", dataset.cols)} FROM {table} WHERE 1=1 ORDER BY Date ASC LIMIT -1 OFFSET {dataset.GetInternalDbOffset()};");  
  
 IEnumerable<Vector> input = SQL\_Driver.ReadRow\_AsVector(conn, /\*"(" + from\_string + ")"\*/ table, dataset.cols, $"1=1 ORDER BY Date ASC LIMIT -1 OFFSET {dataset.GetInternalDbOffset()}");  
 OutputDataProducer output = new OutputDataProducer(dataset.safety, 3, input, dataset.GetValueIndexes(), dataset.GetValueIndexes().Count() / 2);  
  
  
 IEnumerator<Vector> signalSource = output.GetEnumerator();  
 IEnumerator<Quote> quoteSource = SQL\_Driver.ReadRow\_AsQuote(conn, table, new List<string> { "Date", "open", "high", "low", "close", "volume" }, dataset.GetInternalDbOffset()).GetEnumerator();  
 //signalSource.MoveNext(); //its 1 ahead idk why  
 int i = 0;  
 while (signalSource.MoveNext() && quoteSource.MoveNext())  
 {  
 Quote q = quoteSource.Current;  
  
 ////table = "Price";  
 //// adding date and high  
 //int i= MyChart.Series[table].Points.AddXY(q.Date, q.High);  
 //// adding low  
 //MyChart.Series[table].Points[i].YValues[1] = (double)q.Low;  
 ////adding open  
 //MyChart.Series[table].Points[i].YValues[2] = (double)q.Open;  
 //// adding close  
 //MyChart.Series[table].Points[i].YValues[3] = (double)q.Close;  
 ////plotpoint(table, q);  
 //if (signalSource.Current[0] == 1)  
 //{  
 // //buy  
 // MyChart.Series[table].Points[i].MarkerStyle = MarkerStyle.Circle;  
 // MyChart.Series[table].Points[i].MarkerColor = Color.Green;  
 //}  
 //else if (signalSource.Current[0] == -1)  
 //{  
 // //sell  
 // MyChart.Series[table].Points[i].MarkerStyle = MarkerStyle.Circle;  
 // MyChart.Series[table].Points[i].MarkerColor = Color.Red;  
 //}  
  
 var safeInvoker = new OnPointToBePlotted(PlotPoint);  
 Invoke(safeInvoker, table, q, signalSource.Current[0]);  
 i++;  
 if (i > 500) break;  
 }  
 ChangeRecordsPlotted(i);  
 e.Result = MyChart;  
 }  
 }  
   
  
 private void CandleChart\_Click(object sender, EventArgs e)  
 {  
  
 }  
 public void LockTokenSelector()  
 {  
 TokenSelector.ReadOnly = false; //invoke? nah  
 }  
 public void UnLockTokenSelector()  
 {  
 TokenSelector.ReadOnly = true;  
 }  
 public void SetTokens(IEnumerable<string> tokens)  
 {  
 TokenSelector.Items.Clear();  
 Invoke(() => { foreach (string token in tokens) { TokenSelector.Items.Add(token); } });  
 }  
 private void TokenSelector\_SelectedItemChanged(object sender, EventArgs e)  
 {  
 bool nothingwrong = true;  
 if (!PlotterWorker.IsBusy)  
 {  
 if (TokenSelector.SelectedItem != null)  
 {  
 TokenSelector.Enabled = false;  
 PlotterWorker.RunWorkerCompleted += (o, e) =>  
 {  
 if (e.Error != null)  
 {  
 MessageBox.Show($"check if data table has missing values {e.Error.Message}");  
 nothingwrong = false;  
 }  
 TokenSelector.Enabled = true;  
 };  
 if (nothingwrong)  
 {  
 ClearChart();  
 PlotAsset(TokenSelector.SelectedItem.ToString());  
 }  
 }  
 }  
 }  
  
 private void ExitB\_Click(object sender, EventArgs e)  
 {  
 Exit(this, e);  
 }  
 }  
}

## .\WinFormsApp2\DataControls/VectorViewer.cs

﻿using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
using System.Windows.Forms;  
  
using NeaLibrary.DataStructures;  
  
namespace WinFormsApp2.DataControls  
{  
 public partial class VectorViewer : UserControl  
 {  
 public event EventHandler Close;  
 public VectorViewer()  
 {  
 InitializeComponent();  
 }  
  
 private void CloseButton\_Click(object sender, EventArgs e)  
 {  
 try { Close(this, EventArgs.Empty); }  
 catch { }  
 }  
 public void Display(Vector v)  
 {  
 Table.Controls.Clear();  
 Table.ColumnCount = 1;  
 Table.RowCount = 1;  
 foreach (double d in v)  
 {  
 TextBox textBox = new TextBox();  
 textBox.ReadOnly = true;  
 textBox.Text = d.ToString();  
 Table.Controls.Add(textBox, 1, Table.RowCount);  
 Table.RowCount++;  
 textBox.Dock = DockStyle.Fill;  
 textBox.Show();  
 }  
 Table.RowStyles.Clear();  
 Table.Update();  
 }  
 }  
}

## .\WinFormsApp2\FFNNControls/FFNNControl.cs

﻿using NeaLibrary.Data;  
using NeaLibrary.DataStructures;  
  
using NeaLibrary.NeuralNetwork.FFNN;  
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
using System.Windows.Forms;  
using WinFormsApp2.MainControls;  
  
namespace WinFormsApp2.FFNNControls  
{  
 public partial class FFNNControl : UserControl  
 {  
 SimpleTextBinding FFNNSource\_TextBinding = new SimpleTextBinding();  
 SimpleTextBinding DataSourceTextBox\_Binding = new SimpleTextBinding();  
 IDataSet? dataset;  
 FFNN\_Client? client;  
  
 CancellationTokenSource cancelTask = new CancellationTokenSource();  
 CancellationToken cancelTaskToken;  
  
 public event EventHandler<FFNN\_Client> ClientReleased;  
 public event EventHandler<GraphDisplay> GraphDisplayRequested;  
 public event EventHandler Close;  
 public FFNNControl()  
 {  
 InitializeComponent();  
 BindLabels();  
 //Thread tr = new Thread(() => { PlotFFNN(); });  
 //tr.Start();  
 cancelTaskToken = cancelTask.Token;  
 }  
  
 void OnGraphDisplayRequested(GraphDisplay g)  
 {  
 EventHandler<GraphDisplay> handler = GraphDisplayRequested;  
 if (handler != null && client != null)  
 {  
 g.SetGraph(client.ToGraph());  
 handler(this, g);  
 }  
 }  
  
 private void BindLabels()  
 {  
 DataSourceLocation.DataBindings.Add("Text", DataSourceTextBox\_Binding, "Text");  
 FFNNSourceTextBox.DataBindings.Add("Text", FFNNSource\_TextBinding, "Text");  
 }  
 private void SetName(string name)  
 {  
 this.Name = name;  
 }  
 class SimpleTextBinding : INotifyPropertyChanged //Binding code from Stack Overflow  
 {  
 private string text = "null";  
 public string Text  
 {  
 get { return text; }  
 set  
 {  
 text = value;  
 InvokePropertyChanged(new PropertyChangedEventArgs("Text"));  
 }  
 }  
  
 public event PropertyChangedEventHandler? PropertyChanged;  
  
 public void InvokePropertyChanged(PropertyChangedEventArgs e)  
 {  
 PropertyChangedEventHandler handler = PropertyChanged!;  
 if (handler != null) handler(this, e);  
 }  
 }  
  
 protected void ReleaseClient()  
 {  
 EventHandler<FFNN\_Client> handler = ClientReleased;  
 if (handler != null)  
 {  
 handler(this, this.client!); //shouldnt be null when called  
 client = null;  
 }  
 FFNNChart.Series["Error"].Points.Clear();  
  
 }  
  
 private void SelectDataSourceB\_Click(object sender, EventArgs e)  
 {  
 using (OpenFileDialog OpenDatafileDialogue = new OpenFileDialog())  
 {  
 OpenDatafileDialogue.Multiselect = false;  
 OpenDatafileDialogue.Filter = "Dataset Files | \*.ads;\*.ds;\*dbds";  
 DialogResult result = OpenDatafileDialogue.ShowDialog();  
 if (result == DialogResult.OK && !string.IsNullOrWhiteSpace(OpenDatafileDialogue.FileName))  
 {  
 //.ads ArrayDataSet  
 //.ds DataSet  
 //.dbds DB\_DataSet  
 string c = OpenDatafileDialogue.FileName.Split('.').Last();  
 switch (c)  
 {  
 case "ads":  
 dataset = ArrayDataSet.Load(OpenDatafileDialogue.FileName);  
 break;  
 case "ds":  
 dataset = NeaLibrary.DataStructures.DataSet.Load(OpenDatafileDialogue.FileName);  
 break;  
 case "dbds":  
 dataset = DB\_DataSet.Load(OpenDatafileDialogue.FileName);  
 break;  
 }  
 DataSourceTextBox\_Binding.Text = OpenDatafileDialogue.FileName;  
 }  
 }  
 }  
  
 private void SelectFFNNB\_Click(object sender, EventArgs e)  
 {  
 using (OpenFileDialog OpenDatafileDialogue = new OpenFileDialog())  
 {  
 OpenDatafileDialogue.Multiselect = false;  
 OpenDatafileDialogue.Filter = "FFNN Files | \*.ffnn";  
 DialogResult result = OpenDatafileDialogue.ShowDialog();  
 if (result == DialogResult.OK && !string.IsNullOrWhiteSpace(OpenDatafileDialogue.FileName))  
 {  
 if (client != null)  
 {  
 ReleaseClient();  
 }  
  
 client = FFNN\_Client.Load(OpenDatafileDialogue.FileName);  
  
 LearningRateNUpDown.Value = (decimal)client.GetLearningRate();  
  
 FFNNSource\_TextBinding.Text = OpenDatafileDialogue.FileName;  
 ClientsCountLabel.Text = client.GetClients().Count().ToString();  
  
 client.NextGeneration += (sender, e) => PlotPoint\_FFNNChart(e.Item1, e.Item2);  
 client.Unpause();  
  
 SetName(OpenDatafileDialogue.SafeFileName);  
 }  
 }  
 }  
 public void FFNN\_Programatic\_Assign(string path)  
 {  
 if (client != null)  
 {  
 ReleaseClient();  
 }  
  
 client = FFNN\_Client.Load(path);  
  
 LearningRateNUpDown.Value = (decimal)client.GetLearningRate();  
  
 FFNNSource\_TextBinding.Text = path;  
 ClientsCountLabel.Text = client.GetClients().Count().ToString();  
  
 client.NextGeneration += (sender, e) => PlotPoint\_FFNNChart(e.Item1, e.Item2);  
 client.Unpause();  
 SetName(path.Split('\\').Last());  
 }  
  
 private void TrainB\_Click(object sender, EventArgs e)  
 {  
 if (dataset != null && client != null)  
 {  
 try  
 {  
 Task.Run(() =>  
 {  
 bool relevant = true;  
 ClientReleased += (sender, e) => relevant = false;  
 while (relevant)  
 {  
 while (!client.GetPause())  
 {  
 if (!relevant) break;  
 client.Train(dataset: dataset);  
 }  
 }  
 MessageBox.Show("Successfully terminated");  
 Clear\_FFNNChart();  
 }, cancelTaskToken  
 ); //we DONT want to wait  
 }  
 catch (Exception ex)  
 {  
 MessageBox.Show(ex.Message);  
 }  
 }  
  
 }  
  
 private void PauseB\_Click(object sender, EventArgs e)  
 {  
 if (client != null)  
 {  
 bool state = client.GetPause();  
 if (state)  
 {  
 //Paused  
 client.TogglePause();  
 PauseB.Text = "Pause";  
 }  
 else  
 {  
 //Unpaused  
 client.TogglePause();  
 PauseB.Text = "Unpause";  
 }  
 }  
 }  
  
 private void SaveB\_Click(object sender, EventArgs e)  
 {  
 if (dataset != null && client != null)  
 {  
 using (SaveFileDialog SaveLocationDialogue = new SaveFileDialog())  
 {  
 SaveLocationDialogue.AddExtension = true;  
 SaveLocationDialogue.DefaultExt = ".ffnn";  
 //SaveLocationDialogue.InitialDirectory=  
 DialogResult result = SaveLocationDialogue.ShowDialog();  
 if (result == DialogResult.OK && !string.IsNullOrWhiteSpace(SaveLocationDialogue.FileName))  
 {  
  
 string SaveLocation = SaveLocationDialogue.FileName;  
 client.Save(SaveLocation);  
 }  
 }  
 }  
 }  
 private void PlotPoint\_FFNNChart(double x, double y)  
 {  
 //FFNNChart.ChartAreas["ChartArea1"].AxisY.Maximum = 5;  
 //FFNNChart.ChartAreas["ChartArea1"].AxisY.Minimum = 0;  
 //FFNNChart.ChartAreas["ChartArea1"].AxisX.Maximum = 10000;  
 //FFNNChart.ChartAreas["ChartArea1"].AxisX.Minimum = 0;  
 //FFNNChart.ChartAreas["ChartArea1"].AxisX.IsLogarithmic = true;  
 //FFNNChart.ChartAreas["ChartArea1"].AxisY.Maximum = 1;  
  
 Action action = () => { FFNNChart.Series["Error"].Points.AddXY(x, y); Update(); };  
 this.Invoke(action);  
 }  
  
 private void Clear\_FFNNChart()  
 {  
 Action action = () => { FFNNChart.Series["Error"].Points.Clear(); Update(); };  
 this.Invoke(action);  
 }  
  
 private void LearningRateNUpDown\_ValueChanged(object sender, EventArgs e)  
 {  
 if (client != null)  
 {  
  
 if (Decimal.ToDouble(LearningRateNUpDown.Value) != client.GetLearningRate())  
 {  
 client.SetLearningRate(Decimal.ToDouble(LearningRateNUpDown.Value));  
 }  
 }  
 }  
  
 private void CloseButton\_Click(object sender, EventArgs e)  
 {  
 if (client != null) { client.Pause(); ReleaseClient(); }  
 Close(this, e);  
 }  
  
 private void ViewGraphB\_Click(object sender, EventArgs e)  
 {  
 GraphDisplay graphDisplay = new GraphDisplay();  
 OnGraphDisplayRequested(graphDisplay);  
 }  
  
 //private async Task PlotFFNN()  
 //{  
 // while (true)  
 // {  
 // if (client != null)  
 // {  
 // if (dataset != null)  
 // {  
 // //if (FFNNChart.Series["Accuracy"].Points.Count != 0)  
 // //{  
  
 // //FFNNChart.ChartAreas["Chart1"].AxisX.Interval = 1000;  
 // PlotPoint\_FFNNChart(client.generations + 1, client.lowest\_error);  
 // //}  
 // }  
 // }  
 // }  
 //}  
  
 }  
}

## .\WinFormsApp2\FFNNControls/FFNNCreateControl.cs

﻿using NeaLibrary.Data;  
using NeaLibrary.DataStructures;  
using NeaLibrary.NeuralNetwork.FFNN;  
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
using System.Windows.Forms;  
  
namespace WinFormsApp2.FFNNControls  
{  
 public partial class FFNNCreateControl : UserControl  
 {  
  
 List<HiddenLayerDimensionSpecifier> HiddenLayers = new List<HiddenLayerDimensionSpecifier>();  
 public FFNNCreateControl()  
 {  
 InitializeComponent();  
 }  
 public event EventHandler FinishedCreating;  
 private void label5\_Click(object sender, EventArgs e)  
 {  
  
 }  
 private void ReRenderHiddenLayersTable()  
 {  
 HiddenLayerP.RowStyles.Clear();  
 HiddenLayerP.RowCount = HiddenLayers.Count;  
 int row = 0;  
 foreach (HiddenLayerDimensionSpecifier layer in HiddenLayers)  
 {  
 HiddenLayerP.Controls.Add(layer, 0, row);  
 row++;  
 }  
 }  
 private void RemoveHiddenLayer(object sender, EventArgs e)  
 {  
 HiddenLayers.Remove((HiddenLayerDimensionSpecifier)sender);  
 ReRenderHiddenLayersTable();  
 }  
  
 private void AddHiddenLayer\_B\_Click(object sender, EventArgs e)  
 {  
 HiddenLayerP.RowStyles.Clear();  
 HiddenLayerP.Visible = true;  
 HiddenLayerDimensionSpecifier c = new HiddenLayerDimensionSpecifier();  
 c.Show();  
 c.RemoveHiddenLayer += new EventHandler(RemoveHiddenLayer);  
 HiddenLayers.Add(c);  
 HiddenLayerP.RowCount += 1;  
 HiddenLayerP.Controls.Add(c);  
 //.Controls.Add(c);  
 }  
  
 private void Create\_B\_Click(object sender, EventArgs e)  
 {  
 List<int> topology = new List<int>  
 {  
 (int)InputDimension\_NUpDown.Value  
 };  
 foreach (HiddenLayerDimensionSpecifier h in HiddenLayers)  
 {  
 topology.Add((int)h.numericUpDown1.Value);  
 }  
 topology.Add((int)OutputDimension\_NUpDown.Value);  
 Vector tp = new Vector(topology.Count);  
 for (int i = 0; i < tp.dimension; i++)  
 {  
 tp[i] = topology[i];  
 }  
 SaveFFNNDialogue.DefaultExt = ".ffnn";  
  
 DialogResult result = SaveFFNNDialogue.ShowDialog();  
 if (result == DialogResult.OK && !string.IsNullOrWhiteSpace(SaveFFNNDialogue.FileName))  
 {  
 FFNN\_Client client = new FFNN\_Client(tp, (double)LearningRate\_NUpDown.Value, (int)Clients\_NUpDown.Value);  
 client.Save(SaveFFNNDialogue.FileName);  
 FinishedCreating(this, new EventArgs());  
 }  
  
  
 }  
  
 private void HiddenLayerP\_Paint(object sender, PaintEventArgs e)  
 {  
  
 }  
  
 private void CloseButton\_Click(object sender, EventArgs e)  
 {  
 FinishedCreating(this, EventArgs.Empty);  
 }  
 }  
}

## .\WinFormsApp2\FFNNControls/HiddenLayerDimensionSpecifier.cs

﻿using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
using System.Windows.Forms;  
  
namespace WinFormsApp2.FFNNControls  
{  
 public partial class HiddenLayerDimensionSpecifier : UserControl  
 {  
 public event EventHandler RemoveHiddenLayer;  
 public HiddenLayerDimensionSpecifier()  
 {  
 InitializeComponent();  
 }  
  
 private void button1\_Click(object sender, EventArgs e)  
 {  
 RemoveHiddenLayer(this,e);  
   
 this.Hide();  
 this.Dispose();  
  
  
 }  
 }  
}

## .\WinFormsApp2\MainControls/Class1.cs

﻿using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace WinFormsApp2.MainControls  
{  
 public class SimpleTextBinding : INotifyPropertyChanged //from stack overflow  
 {  
 private string text = "null";  
 public string Text  
 {  
 get { return text; }  
 set  
 {  
 text = value;  
 InvokePropertyChanged(new PropertyChangedEventArgs("Text"));  
 }  
 }  
  
  
  
 public event PropertyChangedEventHandler? PropertyChanged;  
  
  
 /// <summary>  
 /// Stack overflow accreddited   
 /// </summary>  
 /// <param name="e"></param>  
  
 public void InvokePropertyChanged(PropertyChangedEventArgs e)  
 {  
 PropertyChangedEventHandler handler = PropertyChanged!;  
 if (handler != null) handler(this, e);  
 }  
 }  
}

## .\WinFormsApp2\MainControls/CreateSelector.cs

﻿using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
using System.Windows.Forms;  
using WindowsFormsApp1.NEATControls;  
using static System.Windows.Forms.VisualStyles.VisualStyleElement.Button;  
  
namespace WinFormsApp2.MainControls  
{  
 public partial class CreateSelector : UserControl  
 {  
 public Choice Selected = Choice.Invalid;  
 public event EventHandler ChoiceMade;  
  
 public CreateSelector()  
 {  
 InitializeComponent();  
 }  
  
 public enum Choice  
 {  
 DataSet,  
 DataTable,  
 FFNN,  
 NEAT,  
 Invalid  
 }  
  
  
  
  
  
  
  
  
 private void Create\_B\_Click(object sender, EventArgs e)  
 {  
 if (NEAT\_RB.Checked == true)  
 {  
 Selected = Choice.NEAT;  
 ChoiceMade(sender, e);  
 }  
 else if (DataTable\_RB.Checked == true)  
 {  
 Selected = Choice.DataTable;  
 ChoiceMade(sender, e);  
 }  
 else if (FFNN\_RB.Checked == true)  
 {  
 Selected = Choice.FFNN;  
 ChoiceMade(sender, e);  
 }  
 else if (DataSet\_RB.Checked == true)  
 {  
 Selected = Choice.DataSet;  
 ChoiceMade(sender, e);  
 }  
 else  
 {  
 MessageBox.Show("Please select a value");  
 }  
 }  
  
 private void DataTable\_RB\_CheckedChanged(object sender, EventArgs e)  
 {  
  
 }  
  
 private void ExitB\_Click(object sender, EventArgs e)  
 {  
 ChoiceMade(sender, e);  
 }  
  
 //public async int ShowAndGetChoice()  
 //{  
 // Visible = true;  
 // //while (SELECTED==0) { continue; }  
 // Choice c = await  
 // return SELECTED;  
 //}  
 }  
}

## .\WinFormsApp2\MainControls/GraphDisplay.cs

﻿using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Drawing.Drawing2D;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
using System.Windows.Forms;  
using Microsoft.Diagnostics.Tracing.Parsers.MicrosoftWindowsTCPIP;  
using NeaLibrary.DataStructures;  
namespace WinFormsApp2.MainControls  
{  
 public partial class GraphDisplay : UserControl  
 {  
 public event EventHandler Exit;  
 Graph? graph;  
 Pen blackPen = new Pen(Color.Black);  
 Pen redPen = new Pen(Color.Red);  
 Brush redBrush = new SolidBrush(Color.Red);  
 public GraphDisplay()  
 {  
 InitializeComponent();  
 }  
 public void SetGraph(Graph graph)  
 {  
 this.graph = graph;  
 }  
  
 public void Render()  
 {  
 if (graph == null)  
 {  
  
 }  
 else  
 {  
 panel1.Invalidate();  
 Update();  
 }  
 }  
  
 private void panel1\_Paint(object sender, PaintEventArgs e)  
 {  
 void DrawLine(int to, int from, bool black = false, float penwidth = 1.4f)  
 {  
 try  
 {  
 //graph shouldsnt be null when this is caleld  
 double width = panel1.Width;  
 double height = panel1.Height;  
 //blackPen.Width = 1.4f;  
 int x1 = (int)(graph.NodesCoordinates[to].Item1 \* width); //simple truncation  
 int y1 = (int)(graph.NodesCoordinates[to].Item2 \* height);  
 int x2 = (int)(graph.NodesCoordinates[from].Item1 \* width);  
 int y2 = (int)(graph.NodesCoordinates[from].Item2 \* height);  
  
 Pen p;  
 if (black) { p = blackPen; }  
 else  
 {  
 p = new Pen(colourscale(graph[to, from]));  
 }  
 p.DashStyle = DashStyle.Solid;  
 p.StartCap = LineCap.ArrowAnchor;  
 p.Width = penwidth;  
 e.Graphics.DrawLine(p, new Point(x1, y1), new Point(x2, y2));  
 }  
 catch { }  
 }  
  
  
 //https://stackoverflow.com/questions/55601338/get-a-color-value-within-a-gradient-based-on-a-value  
 //https://stackoverflow.com/questions/3722307/is-there-an-easy-way-to-blend-two-system-drawing-color-values  
 Color ColourInterpolate(Color c1, Color c2, double fraction)  
 {  
 byte r = (byte)(c2.R \* fraction + (1 - fraction) \* (c1.R));  
 byte g = (byte)(c2.G \* fraction + (1 - fraction) \* (c1.G));  
 byte b = (byte)(c2.B \* fraction + (1 - fraction) \* (c1.B));  
 return Color.FromArgb(r, g, b);  
 }  
  
 Color colourscale(double value)  
 {//red yellow green blue  
 if (value < -2)  
 {  
 return Color.Red;  
 }  
 else if (value >= -2 && value < -0.5)  
 {  
 return ColourInterpolate(Color.Red, Color.Yellow, (value + 2) / 1.5); // --2 = +2  
 }  
 else if (value >= -0.5 && value < 0.5)  
 {  
 return ColourInterpolate(Color.Yellow, Color.Green, (value + 0.5)); // - - 0.5 /1  
 }  
 else if (value >= 0.5 && value < 2)  
 {  
 return ColourInterpolate(Color.Green, Color.Blue, (value + 0.5));  
 }  
 else  
 {  
 return Color.Blue;  
 }  
 }  
  
 //DateTimeOffset.  
  
 void DrawCircle(int node, double r = 4)  
 {  
 //graph shouldsnt be null when this is caleld  
 double width = panel1.Width;  
 double height = panel1.Height;  
 blackPen.Width = 1;  
 try  
 {  
 int x1 = (int)(graph.NodesCoordinates[node].Item1 \* width); //simple truncation  
 int y1 = (int)(graph.NodesCoordinates[node].Item2 \* height);  
  
 Vector r\_v = new Vector(2);  
 r\_v[0] = -1;  
 r\_v[1] = -1;  
 r\_v = (r) \* r\_v;  
  
 x1 = (int)(x1 + r\_v[0]);  
 y1 = (int)(y1 + r\_v[1]);  
  
 e.Graphics.FillEllipse(redBrush, (int)x1, (int)y1, (int)r \* 2, (int)r \* 2);  
 }  
 catch { }  
 }  
 if (graph != null)  
 {  
 //foreach (KeyValuePair<int,(double,double)> v in graph.Nodes)  
 //{  
  
 //}  
 //foreach (var v in graph.Edges())  
 //{  
  
 //}  
  
 for (int from = 0; from < graph.NodeCount; from++)  
 {  
 DrawCircle(from);  
 for (int to = 0; to < graph.NodeCount; to++)  
 {  
 if (graph.adjacencyMatrix[to, from] != 0) DrawLine(to, from);  
 }  
 }  
 }  
 }  
  
 private void ExitB\_Click(object sender, EventArgs e)  
 {  
 Exit(this, EventArgs.Empty);  
 }  
 }  
}

## .\WinFormsApp2\MainControls/Settings.cs

﻿using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
using System.Windows.Forms;  
  
namespace WinFormsApp2.MainControls  
{  
 public partial class Settings : UserControl  
 {  
 public Settings()  
 {  
 InitializeComponent();  
 }  
 }  
}

## .\WinFormsApp2\MainControls/SettingsControl.cs

﻿using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
//using System.Text;  
using System.Threading.Tasks;  
using System.Windows.Forms;  
using NeaLibrary.Data.Other;  
using NeaLibrary.Tools;  
using Newtonsoft.Json;  
using Newtonsoft.Json.Linq;  
using ScottPlot.Drawing.Colormaps;  
using WinFormsApp2.DataControls;  
  
namespace WinFormsApp2.MainControls  
{  
 public partial class SettingsControl : UserControl  
 {  
  
 public event EventHandler Exit;  
 //List<SettingsItem> \_settings = new List<SettingsItem>();  
 public SettingsControl()  
 {  
 InitializeComponent();  
 loadSettings();  
  
 }  
 void OnSettingsChanged(object sender, object e)  
 {  
 //Tuple<string,SettingsItem> t=sender as Tuple<string,SettingsItem>;  
 //if (t!=null)  
 //{  
 // ((SettingsSelector)(SettingsScrollPanel.Controls[t.Item1])).SetInfo(t.Item2);  
 //}  
 SettingsSelector s = (SettingsSelector)sender;  
 //e is the value  
 //Console.WriteLine();  
 string val = (string)e;  
 Tools.SetGlobalVar(s.Name, val);//controls name should match the setting name  
 }  
 void OnToolsSettingsChanged(object sender, object e)  
 {  
  
 }  
 void loadSettings()  
 {  
 //\_settings.Clear();  
 SettingsScrollPanel.Controls.Clear();  
 SettingsScrollPanel.RowCount = 1;  
 SettingsScrollPanel.ColumnCount = 1;  
 SettingsScrollPanel.RowStyles.Clear();  
 string s = Tools.GetAllSettings();  
 JObject j = JObject.Parse(s);  
 foreach (KeyValuePair<string, JToken> pair in j)  
 {  
 SettingsItem settingsItem = pair.Value.ToObject<SettingsItem>()!;  
 string SettingName = pair.Key;  
 SettingsSelector control = new SettingsSelector();  
 control.Name = SettingName;  
 control.SetLabel(pair.Key);  
 control.SetInfo(settingsItem);  
 control.SettingsSelectorValueChanged += OnSettingsChanged;  
 //\_settings.Add(settingsItem);  
 SettingsScrollPanel.Controls.Add(control);  
  
 control.Show();  
 }  
 SettingsScrollPanel.RowStyles.Clear();  
 Tools.SettingsChanged += OnToolsSettingsChanged;  
 }  
  
 private void SettingsScrollPanel\_Paint(object sender, PaintEventArgs e)  
 {  
  
 }  
  
 private void label1\_Click(object sender, EventArgs e)  
 {  
  
 }  
  
 private void ExitB\_Click(object sender, EventArgs e)  
 {  
 Exit(this, e);  
 }  
 }  
}

## .\WinFormsApp2\MainControls/SettingsSelector.cs

﻿using NeaLibrary.Data;  
using NeaLibrary.Data.Other;  
using NeaLibrary.Tools;  
using NeaLibrary.DataStructures;  
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Text;  
using System.Text.RegularExpressions;  
using System.Threading.Tasks;  
using System.Windows.Forms;  
using static System.Windows.Forms.VisualStyles.VisualStyleElement;  
  
namespace WinFormsApp2.MainControls  
{  
 public partial class SettingsSelector : UserControl  
 {  
 public event EventHandler<object> SettingsSelectorValueChanged;  
 bool first = true;  
 Regex rg;  
 Type t;  
 object[] categories;  
 string fileExt;  
 //bool isSourceParameterInfo = true;  
 // SourceParameterInfo \_sourceParameterInfo;  
 public SettingsItem? settingsItem = null;  
 public SettingsSelector()  
 {  
 InitializeComponent();  
 }  
 public void SetLabel(string s)  
 {  
 ArgumentL.Text = s;  
 }  
 private void hide()  
 {  
 generalTB.Visible = false;  
 PathSelectorB.Visible = false;  
 PathSelectorTB.Visible = false;  
 categorySelectorUpDown.Visible = false;  
 numericUpDown1.Visible = false;  
 }  
 public void SetInfo(SettingsItem settingsItem)  
 {  
 first = true;  
 this.settingsItem = settingsItem;  
 if (settingsItem.PathSelector)  
 {  
 hide();  
 PathSelectorB.Visible = true;  
 PathSelectorTB.Visible = true;  
 fileExt = settingsItem.FileExtension;  
 PathSelectorTB.Text = settingsItem.Value;  
 }  
 else if (settingsItem.Categorical)  
 {  
 hide();  
   
 categories = settingsItem.Categories;  
 try  
 {  
 t = Type.GetType(settingsItem.TypeName)!;  
 }  
 catch  
 {  
 MessageBox.Show("Invalid type encountered");  
 }  
 finally  
 {  
 if (t == null) MessageBox.Show("Not found type");  
 }  
 foreach (object o in settingsItem.Categories)  
 {  
 try  
 {  
 var c = (Convert.ChangeType(o, t));  
 categorySelectorUpDown.Items.Add(c);  
 }  
 catch  
 {  
 MessageBox.Show($"Can not cast to type {settingsItem.TypeName}");  
 }  
 }  
 try { categorySelectorUpDown.SelectedItem = Convert.ChangeType(settingsItem.Value, t); } catch {/\*shouldnt fail as it worked in loop above unless settings were corrupted\*/ }  
 categorySelectorUpDown.Visible = true;  
 }  
 else if (settingsItem.IsNumeric)  
 {  
 hide();  
 numericUpDown1.Minimum = (decimal)settingsItem.BoundsAndStep[0];  
 first = true;  
 numericUpDown1.Maximum = (decimal)settingsItem.BoundsAndStep[1];  
 first = true;  
 numericUpDown1.Increment = (decimal)settingsItem.BoundsAndStep[2];  
 numericUpDown1.Value = Decimal.Parse(settingsItem.Value);  
 numericUpDown1.Visible = true;  
 }  
 else//simple text box  
 {  
 if (settingsItem.IsRegex)  
 {  
 rg = new Regex(settingsItem.Regex);  
 generalTB.Validating += generalTB\_Validating;  
 }  
 generalTB.Text = settingsItem.Value;  
 generalTB.Visible = true;  
 }  
 }  
 private void PathSelectorB\_Click(object sender, EventArgs e)  
 {  
 OpenFileDialog OpenDatafileDialogue = new OpenFileDialog();  
 OpenDatafileDialogue.Multiselect = false;  
 OpenDatafileDialogue.Filter = $"Supported Encodings | {fileExt}";  
 DialogResult result = OpenDatafileDialogue.ShowDialog();  
 if (result == DialogResult.OK && !string.IsNullOrWhiteSpace(OpenDatafileDialogue.FileName))  
 {  
 string c = OpenDatafileDialogue.FileName;  
 //Tools.SetGlobalVar();  
  
 PathSelectorTB.Text = c;  
 string clean = c.Replace("\\", "/");  
 settingsItem.Value = clean;  
 SettingsSelectorValueChanged(this, clean);  
 }  
 }  
  
 private void generalTB\_TextChanged(object sender, EventArgs e)  
 {  
 if (first) { first = false; return; }  
 SettingsSelectorValueChanged(this, generalTB.Text);  
 }  
  
 // reference to https://stackoverflow.com/questions/8915151/c-sharp-validating-input-for-textbox-on-winforms  
 private void generalTB\_Validating(object sender, CancelEventArgs e)  
 {  
 //if (first) { first = false; return; }  
 if (!rg.IsMatch(generalTB.Text))  
 {  
 e.Cancel = true;  
 }  
 }  
  
 private void categorySelectorUpDown\_SelectedItemChanged(object sender, EventArgs e)  
 {  
 if (first) { first = false; return; }  
 if (categorySelectorUpDown.SelectedItem == null) return;  
 SettingsSelectorValueChanged(this, categorySelectorUpDown.SelectedItem.ToString()!);  
 }  
  
 private void numericUpDown1\_ValueChanged(object sender, EventArgs e)  
 {  
 if (first) { first = false; return; }  
 SettingsSelectorValueChanged(this, numericUpDown1.Value.ToString());  
 }  
 }  
}

## .\WinFormsApp2\NEATControls/CreateNEATControl.cs

﻿using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.IO;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
using System.Windows.Forms;  
using NeaLibrary.NeuralNetwork.NEAT;  
  
namespace WindowsFormsApp1.NEATControls  
{  
 public partial class CreateNEATControl : UserControl  
 {  
 private string SaveLocation = "";  
 public event EventHandler FinishedCreating;  
 //public event EventHandler Close;  
 public CreateNEATControl()  
 {  
 InitializeComponent();  
 }  
  
 private void ChangeSaveLocationButton\_Click(object sender, EventArgs e)  
 {  
  
 DialogResult result = SaveLocationDialogue.ShowDialog();  
  
 if (result == DialogResult.OK && !string.IsNullOrWhiteSpace(SaveLocationDialogue.FileName))  
 {  
  
 SaveLocation = SaveLocationDialogue.FileName;  
  
 }  
  
 }  
  
 private void ConfirmCreate\_Click(object sender, EventArgs e)  
 {  
 SaveLocationDialogue.DefaultExt = ".neat";  
 DialogResult result = SaveLocationDialogue.ShowDialog();  
  
 if (result == DialogResult.OK && !string.IsNullOrWhiteSpace(SaveLocationDialogue.FileName))  
 {  
  
 SaveLocation = SaveLocationDialogue.FileName;  
  
  
 //Neat n = new Neat((int)InputSelector.Value, (int)OutputSelector.Value, (int)ClientsNumSpecifier.Value);  
  
 NEAT\_Handler n = new NEAT\_Handler((int)InputSelector.Value, (int)OutputSelector.Value, (int)ClientsNumSpecifier.Value);  
  
 n.ChangeParameter(NEAT\_Handler.Action.kill\_rate, (double)KillRateSpecifier.Value);  
 n.ChangeParameter(NEAT\_Handler.Action.specie\_distance, (double)SpecieDistanceSpecifier.Value);  
 n.ChangeParameter(NEAT\_Handler.Action.probability\_mutate\_node, (double)NewNodeMutate.Value);  
 n.ChangeParameter(NEAT\_Handler.Action.probability\_mutate\_link, (double)NewConnectionMutate.Value);  
 n.ChangeParameter(NEAT\_Handler.Action.probability\_mutate\_link\_toggle, (double)ConnectionToggleMutate.Value);  
 n.ChangeParameter(NEAT\_Handler.Action.probability\_mutate\_weight\_random, (double)WeightRandomMutate.Value);  
 n.ChangeParameter(NEAT\_Handler.Action.probability\_mutate\_weight\_shift, (double)WeightShiftMutate.Value);  
 n.ChangeParameter(NEAT\_Handler.Action.weight\_shift\_strength, (double)WeightShiftStrength.Value);  
  
 n.Save(SaveLocation);  
  
 FinishedCreating(this, new EventArgs());  
 }  
 }  
  
 private void CloseButton\_Click(object sender, EventArgs e)  
 {  
 FinishedCreating(this, EventArgs.Empty);  
 }  
 }  
}

## .\WinFormsApp2\NEATControls/NEATControl.cs

﻿using NeaLibrary.Data;  
using NeaLibrary.DataStructures;  
  
using NeaLibrary.NeuralNetwork.NEAT;  
using System.ComponentModel;  
using System.Xml.Linq;  
using WinFormsApp2.MainControls;  
  
namespace WindowsFormsApp1  
{  
 public partial class NEATControl : UserControl  
 {  
 private NEAT\_Handler neat\_handler;  
 private IDataSet dataset;  
 private string datasource\_string = "null";  
 public event EventHandler Close;  
 public event EventHandler<(NEAT\_Handler, string name)> ClientChange;  
 public event EventHandler<GraphDisplay> GraphDisplayRequested;  
  
 bool loading\_parameters = true;  
  
 private BackgroundWorker neatWorker = new BackgroundWorker();  
 public NEATControl()  
 {  
 InitializeComponent();  
 BindLabels();  
  
 //Thread tr = new Thread(() => { PlotNeat(); });  
 //tr.Start();  
  
 }  
  
 //private void NEAT\_Chart\_Click(object sender, EventArgs e)  
 //{  
  
 //}  
 private void PlotPoint\_NeatChart(double x, double y)  
 {  
 Action action = () => { NEAT\_Chart.Series["Performance"].Points.AddXY(x, y); Update(); };  
 this.Invoke(action);  
 }  
 private void BindLabels()  
 {  
 DataSourceLocation.DataBindings.Add("Text", DataSourceTextBox\_Binding, "Text");  
 NeatSourceTextBox.DataBindings.Add("Text", NeatSourceTextBox\_Binding, "Text");  
 }  
 //private async Task PlotNeat()  
 //{  
 // while (true)  
 // {  
 // if (neat\_handler != null)  
 // {  
 // if (neat\_handler.client.PAUSED == false)  
 // {  
 // PlotPoint\_NeatChart(neat\_handler.client.generations, neat\_handler.client.best\_acc);  
 // }  
 // }  
 // }  
 //}  
  
 class SimpleTextBinding : INotifyPropertyChanged //from so  
 {  
 private string text = "null";  
 public string Text  
 {  
 get { return text; }  
 set  
 {  
 text = value;  
 InvokePropertyChanged(new PropertyChangedEventArgs("Text"));  
 }  
 }  
  
  
  
 public event PropertyChangedEventHandler? PropertyChanged;  
  
 public void InvokePropertyChanged(PropertyChangedEventArgs e)  
 {  
 PropertyChangedEventHandler handler = PropertyChanged!;  
 if (handler != null) handler(this, e);  
 }  
 }  
 public void ReleaseClient()  
 {  
 neat\_handler.Pause();  
 try { neat\_handler.NextGeneration -= OnNextGeneration; }  
 catch  
 {  
 //shouldnt happen  
 }  
 Action action = () => { NEAT\_Chart.Series["Performance"].Points.Clear(); Update(); };  
 this.Invoke(action);  
 }  
 public void SetName(string s)  
 {  
 Invoke(() => this.Name = s);  
 }  
 public void Neat\_Programatic\_Assign(string path)  
 {  
 if (neat\_handler != null)  
 {  
 ReleaseClient();  
 }  
  
 neat\_handler = (NEAT\_Handler)NEAT\_Handler.Load(path);//new NEAT\_Handler(Neat.Load(path)!);  
 fetch\_parameters();  
 NeatSourceTextBox.Text = path;  
 //ClientsCountLabel.Text = client.clients.Count().ToString();  
  
 neat\_handler.NextGeneration += OnNextGeneration;  
 SetName(path.Split('\\').Last());  
 OnClientChange(neat\_handler, path.Split('\\').Last());  
 }  
  
 void fetch\_parameters()  
 {  
  
 if (neat\_handler != null)  
 {  
 loading\_parameters = true;  
  
 CreaturesNUpDown.Value = neat\_handler.client!.max\_clients;  
 KillRateNUpDown.Value = (decimal)neat\_handler.client.kill\_rate;  
 SpecieDistanceNUpDown.Value = (decimal)neat\_handler.client.specie\_distance;  
 C1NUpDown.Value = (decimal)neat\_handler.client.c1;  
 C2NUpDown.Value = (decimal)neat\_handler.client.c2;  
 C3NUpDown.Value = (decimal)neat\_handler.client.c3;  
  
 MNodeNUpDown.Value = (decimal)neat\_handler.client.probability\_mutate\_node;  
 MConnNUpDown.Value = (decimal)neat\_handler.client.probability\_mutate\_link;  
 MToggleNUpDown.Value = (decimal)neat\_handler.client.probability\_mutate\_link\_toggle;  
 MWeightShiftNUpDown.Value = (decimal)neat\_handler.client.probability\_mutate\_weight\_shift;  
 WeightShiftStrengthNUpDown.Value = (decimal)neat\_handler.client.weight\_shift\_strength;  
  
 loading\_parameters = false;  
 }  
 }  
 void OnClientChange(NEAT\_Handler n, string name)  
 {  
 EventHandler<(NEAT\_Handler, string)> handler = ClientChange;  
 if (handler != null)  
 {  
 //fetch\_parameters();  
 handler(this, (n, name));  
 }  
 }  
 void OnGraphDisplayRequested(GraphDisplay g)  
 {  
 EventHandler<GraphDisplay> handler = GraphDisplayRequested;  
 if (handler != null)  
 {  
 Graph? temp = (neat\_handler.client.GetBest().Phenotype != null) ? neat\_handler.client.GetBest().Phenotype : neat\_handler.client.GetBest().ToGraph();  
 if (temp == null) throw new Exception("No phonetype for the best creature");  
 g.SetGraph(temp);  
 handler(this, g);  
 }  
 }  
  
 private void SelectDataSourceB\_Click(object sender, EventArgs e)  
 {  
 OpenDatafileDialogue.Multiselect = false;  
 OpenDatafileDialogue.Filter = "Supported Encodings | \*.ads;\*.ds;\*dbds";  
 DialogResult result = OpenDatafileDialogue.ShowDialog();  
 if (result == DialogResult.OK && !string.IsNullOrWhiteSpace(OpenDatafileDialogue.FileName))  
 {  
 //.ads ArrayDataSet  
 //.ds DataSet  
 //.dbds DB\_DataSet  
 string c = OpenDatafileDialogue.FileName.Split('.').Last();  
 switch (c)  
 {  
 case "ads":  
 dataset = ArrayDataSet.Load(OpenDatafileDialogue.FileName);  
 break;  
 case "ds":  
 dataset = NeaLibrary.DataStructures.DataSet.Load(OpenDatafileDialogue.FileName);  
 break;  
 case "dbds":  
 dataset = DB\_DataSet.Load(OpenDatafileDialogue.FileName);  
 break;  
 }  
 datasource\_string = OpenDatafileDialogue.FileName;  
 DataSourceTextBox\_Binding.Text = datasource\_string;  
 }  
 }  
  
 private void NeatSourceTextBox\_TextChanged(object sender, EventArgs e)  
 {  
  
 }  
  
 private void SelectNeatB\_Click(object sender, EventArgs e)  
 {  
 OpenDatafileDialogue.Multiselect = false;  
 OpenDatafileDialogue.Filter = "Supported Encodings | \*.neat";  
 DialogResult result = OpenDatafileDialogue.ShowDialog();  
 if (result == DialogResult.OK && !string.IsNullOrWhiteSpace(OpenDatafileDialogue.FileName))  
 {  
 //.ads ArrayDataSet  
 //.ds DataSet  
 //.dbds DB\_DataSet  
  
  
 NEAT\_Handler temp = (NEAT\_Handler)NEAT\_Handler.Load(OpenDatafileDialogue.FileName); //new NEAT\_Handler(Neat.Load(OpenDatafileDialogue.FileName)!);  
  
 ReleaseClient();  
  
 neat\_handler = temp;  
  
 string s = OpenDatafileDialogue.FileName;  
 NeatSourceTextBox\_Binding.Text = s;  
  
 neat\_handler.NextGeneration += OnNextGeneration;  
 fetch\_parameters();  
  
 }  
 }  
  
 void OnNextGeneration(object? sender, (int, double) e)  
 {  
 PlotPoint\_NeatChart(e.Item1, e.Item2);  
 }  
  
 private void TrainB\_Click(object sender, EventArgs e)  
 {  
 if (dataset != null && neat\_handler != null)  
 {  
 neat\_handler.Unpause();  
 neat\_handler.dataset = dataset;  
 PauseB.Enabled = true;  
  
 if (neatWorker.IsBusy)  
 {  
 MessageBox.Show("Already training");  
 return;  
 //return;  
 }  
  
 //Task.Run(() => neat\_handler.Train(0.9));  
  
 neatWorker.DoWork += (seder, ev) =>  
 {  
 Console.WriteLine();  
 //try thanks to stupid documentation  
 //{  
 //throw new NotImplementedException();  
 neat\_handler.Train(dataset); //fix the task thing  
 Console.WriteLine();  
 //}catch (Exception ex)  
 //{  
 // throw ex;  
 //}  
 };  
  
 neatWorker.RunWorkerCompleted += (sdr, evnts) =>  
 {  
 if (evnts.Error != null)  
 {  
 MessageBox.Show("Error encountered while training: " + evnts.Error.Message + evnts.Error.ToString());  
 }  
 };  
  
 neatWorker.RunWorkerAsync();  
 }  
  
 }  
 private void PauseB\_Click(object sender, EventArgs e)  
 {  
  
 if (dataset != null && neat\_handler != null)  
 {  
 neat\_handler.Pause();  
 PauseB.Enabled = false;  
 }  
 }  
  
 //private void SaveB\_Click(object sender, EventArgs e)  
 //{  
 // string SaveLocation;  
  
 // if (dataset != null && neat\_handler != null)  
 // {  
 // using (SaveFileDialog SaveLocationDialogue = new SaveFileDialog())  
 // {  
  
 // //SaveLocationDialogue.InitialDirectory=  
 // DialogResult result = SaveLocationDialogue.ShowDialog();  
 // if (result == DialogResult.OK && !string.IsNullOrWhiteSpace(SaveLocationDialogue.FileName))  
 // {  
  
 // SaveLocation = SaveLocationDialogue.FileName;  
 // neat\_handler.client.Save(SaveLocation);  
 // }  
 // }  
 // }  
 //}  
  
 private void SaveB\_Click\_1(object sender, EventArgs e)  
 {  
 string SaveLocation;  
  
 if (dataset != null && neat\_handler != null)  
 {  
 using (SaveFileDialog SaveLocationDialogue = new SaveFileDialog())  
 {  
 //SaveLocationDialogue.InitialDirectory=  
 SaveLocationDialogue.Filter = "NEAT (\*.neat)|\*.neat"; // credit to https://stackoverflow.com/questions/1213339/save-file-with-appropriate-extension-in-a-save-file-prompt  
 SaveLocationDialogue.DefaultExt = "neat";  
 SaveLocationDialogue.AddExtension = true;  
 DialogResult result = SaveLocationDialogue.ShowDialog();  
 if (result == DialogResult.OK && !string.IsNullOrWhiteSpace(SaveLocationDialogue.FileName))  
 {  
  
 SaveLocation = SaveLocationDialogue.FileName;  
 neat\_handler.Save(SaveLocation);  
 }  
 }  
 }  
 }  
  
 private void CloseButton\_Click(object sender, EventArgs e)  
 {  
  
 Close(this, e);  
 }  
  
 private void ViewAsGraphB\_Click(object sender, EventArgs e)  
 {  
 if (neat\_handler != null)  
 {  
 GraphDisplay d = new GraphDisplay();  
 OnGraphDisplayRequested(d);  
 }  
 }  
  
 private void ParametersLabel\_Click(object sender, EventArgs e)  
 {  
  
 }  
  
 private void NEATControl\_Load(object sender, EventArgs e)  
 {  
  
 }  
  
 private void CreaturesNUpDown\_ValueChanged(object sender, EventArgs e)  
 {  
 if (loading\_parameters) return;  
 neat\_handler.ChangeParameter(NEAT\_Handler.Action.creature\_count, (double)CreaturesNUpDown.Value); //cast to int later so its fine  
 }  
  
 private void KillRateNUpDown\_ValueChanged(object sender, EventArgs e)  
 {  
 if (loading\_parameters) return;  
 neat\_handler.ChangeParameter(NEAT\_Handler.Action.kill\_rate, (double)KillRateNUpDown.Value);  
 }  
  
 private void SpecieDistanceNUpDown\_ValueChanged(object sender, EventArgs e)  
 {  
 if (loading\_parameters) return;  
 neat\_handler.ChangeParameter(NEAT\_Handler.Action.specie\_distance, (double)SpecieDistanceNUpDown.Value);  
 }  
  
 private void C1NUpDown\_ValueChanged(object sender, EventArgs e)  
 {  
 if (loading\_parameters) return;  
 neat\_handler.ChangeParameter(NEAT\_Handler.Action.c1, (double)C1NUpDown.Value);  
 }  
  
 private void C2NUpDown\_ValueChanged(object sender, EventArgs e)  
 {  
 if (loading\_parameters) return;  
 neat\_handler.ChangeParameter(NEAT\_Handler.Action.c2, (double)C2NUpDown.Value);  
 }  
  
 private void C3NUpDown\_ValueChanged(object sender, EventArgs e)  
 {  
 if (loading\_parameters) return;  
 neat\_handler.ChangeParameter(NEAT\_Handler.Action.c3, (double)C3NUpDown.Value);  
 }  
  
 private void MNodeNUpDown\_ValueChanged(object sender, EventArgs e)  
 {  
 if (loading\_parameters) return;  
 neat\_handler.ChangeParameter(NEAT\_Handler.Action.probability\_mutate\_node, (double)MNodeNUpDown.Value);  
 }  
  
 private void MConnNUpDown\_ValueChanged(object sender, EventArgs e)  
 {  
 if (loading\_parameters) return;  
 neat\_handler.ChangeParameter(NEAT\_Handler.Action.probability\_mutate\_link, (double)MConnNUpDown.Value);  
 }  
  
 private void MWeightShiftNUpDown\_ValueChanged(object sender, EventArgs e)  
 {  
 if (loading\_parameters) return;  
 neat\_handler.ChangeParameter(NEAT\_Handler.Action.probability\_mutate\_weight\_shift, (double)MWeightShiftNUpDown.Value);  
 }  
  
 private void WeightShiftStrengthNUpDown\_ValueChanged(object sender, EventArgs e)  
 {  
 if (loading\_parameters) return;  
 neat\_handler.ChangeParameter(NEAT\_Handler.Action.weight\_shift\_strength, (double)WeightShiftStrengthNUpDown.Value);  
 }  
  
 private void MToggleNUpDown\_ValueChanged(object sender, EventArgs e)  
 {  
 if (loading\_parameters) return;  
 neat\_handler.ChangeParameter(NEAT\_Handler.Action.probability\_mutate\_link\_toggle, (double)MToggleNUpDown.Value);  
 }  
  
  
 }  
}