Одеський національний університет імені І. І. Мечникова

Інститут математики, економіки і механіки

Методи математичної фізики

Дипломна робота

бакалавра

на тему: «Генетичні алгоритми у навчанні нечітких нейронних мереж»

«Genetic algorithms for learning fuzzy neural networks»

Виконав: студент денної форми навчання

напряму підготовки 6.040301 Прикладна математика

Заславський Дмитро Володимирович

Керівник: кандидат фіз.-мат. наук, доцент Шумихин С. А.

Рецензент кандидат фіз.-мат. наук, доцент Мойсеенок О. П.

Рекомендовано до захисту:

Протокол засідання кафедри

№_____ від _____ р.

Завідувач кафедри

_____ _____

(підпис) (прізвище, ініціали)

Захищено на засіданні ЕК №_____

протокол №____ від _____ р.

Оцінка_____/___ ___/_____

(за національною шкалою, шкалою ECTS, бали)

Голова ЕК

_____ _____

(підпис) (прізвище, ініціали)

Одеса – 2017

План

Вступ

В даний час, коли обсяги інформації, що збирається швидко набирають зростання і вже неможливо вручну оперативно Обробляти зібрані дані, істотно - важливими елементами соціального життя людини стає автоматизація їх обробки і аналізу.

Необхідно вивчати можливості і ефективність різних форм і методів продажу товарів, виробляти роботу по формуванню запитів населення, підвищенню престижу торговельної фірми. Перспективним з цієї точки зору є застосування інформаційних технологій при обслуговуванні покупців. Автоматизація первинної діагностики та перевірка коректності діагнозу і призначених лікування. Процедура розгляду заяв про користувача з урахуванням неточних питанням і особливостей мови.

На даний момент, в області аналіз даних, успішними інструментами є методи машинного навчання. У даній роботі будуть Розглянуто один з найпотужніших інструментів - нейронні мережі.

Метою дипломної роботи є:

- Вивчення штучних нейронних мереж;
- Вивчення існуючих алгоритмів їх навчання;
- Вивчення систем нечіткого вивода;
- Вивчення генетичних алгоритмів;
- Спроба формування алгоритму навчання нечіткої нейронної мережі засобами генетичних алгоритмів.

Висновки

1. Існує математичне обґрунтування використання нейронних систем за теоремами Колмагорова та Горбані.
2. Що хоча нейронні системи не досягли рівня людини, вони все – таки можуть виконувати досить широкий спектр задач.
3. Оскільки в нашому світі не існує абсолютно точної інформації, то нечітка логіка, а отже і нечіткі нейронні системи будуть розвиватися і активно використовуватися. І хоча все ще існують невирішені проблеми у визначенні та зручність реалізації операції в нечітких безлічі, на даний момент, як і в майбутньому це буде серйозним інструментом для розвитку в області обробки інформації.
4. генетичні алгоритми показали себе досить ефективним засобом для навчання нечітких мереж, однак існують більш ефективні алгоритми навчання.

Список використаної літератури

1 Колмогоров А. Н. / О представлении непрерывных функций нескольких переменных в виде суперпозиции непрерывных функций одного переменного // Докл. АН СССР. — 1958. — Т. 114, № 5. — С. 953–956.

2 Нейроинформатика / [А. Н. Горбань, В. Л. Дунин-Барковский, А. Н. Кирдин, Е. М. Миркес, А. Ю. Новоходько, Д. А. Россиев, С. А. Терехов и др.] — Новосибирск: Наука, 1998. — 296 с.

3 Нейронные сети, генетические алгоритмы и нечеткие системы / Рутковская Д. Пилиньский М., Рутковский Л. – Москва: «Горячая линия», 2006. – 381 с.

4 С.Д.Штовба / "Введение в теорию нечетких множеств и нечеткую логику"-264 с.

5 Fausett L. /“Fundamentals of Neural Networks: Architectures, Algorithms And Applications”.-2001.-449.

Додатки

Код програми:

1) Нечітке трикутне число

```
class FuzzyDouble
   {
      double mediana;
      double fuzzyLeft;
      double fuzzyRigth;
      public FuzzyDouble() { }
      public FuzzyDouble(double A)
      {
         mediana = A;
         fuzzyLeft = 0;
         fuzzyRigth = 0;
      }
      public FuzzyDouble(double FuzzyLeft, double Mediana, double
FuzzyRigth)
      {
         mediana = Mediana;
         //fuzzyLeft = Mediana - FuzzyLeft;
         //fuzzyRigth = FuzzyRigth - Mediana;
         fuzzyLeft = FuzzyLeft;
         fuzzyRigth = FuzzyRigth;
      }
      public static FuzzyDouble operator +(FuzzyDouble obj1, FuzzyDouble
obj2)
      {
         FuzzyDouble result = new FuzzyDouble();
         result.mediana = obj1.mediana + obj2.mediana;
         result.fuzzyLeft = obj1.mediana + obj2.mediana-obj1.fuzzyLeft -
obj2.fuzzyLeft;
         result.fuzzyRigth = obj1.fuzzyRigth + obj2.fuzzyRigth;
         return result;
      }
      public static FuzzyDouble operator -(FuzzyDouble obj1, FuzzyDouble
obj2)
      {
         FuzzyDouble result = new FuzzyDouble();
         result.mediana = obj1.mediana - obj2.mediana;
         result.fuzzyLeft = obj1.fuzzyLeft - obj2.fuzzyLeft;
```

```
            result.fuzzyRigth = obj1.fuzzyRigth - obj2.fuzzyRigth;
            return result;
        }
        public static FuzzyDouble operator *(FuzzyDouble obj1, FuzzyDouble
obj2)
        {
            FuzzyDouble result = new FuzzyDouble();
            result.mediana = obj1.mediana * obj2.mediana;
            result.fuzzyLeft = obj1.fuzzyLeft * obj2.fuzzyLeft;
            result.fuzzyRigth = obj1.fuzzyRigth * obj2.fuzzyRigth;
            return result;
        }
        public static FuzzyDouble operator /(FuzzyDouble obj1, FuzzyDouble
obj2)
        {
            FuzzyDouble result = new FuzzyDouble();
            result.mediana = obj1.mediana / obj2.mediana;
            result.fuzzyLeft = obj1.fuzzyLeft / obj2.fuzzyRigth;
            result.fuzzyRigth = obj1.fuzzyRigth / obj2.fuzzyLeft;
            return result;
        }


        public double Mediana
        {
            get { return mediana; }
            set { mediana = value; }
        }
        public override string ToString()
        {
            return String.Format("({0}, {1}, {2})", mediana - fuzzyLeft, mediana,
mediana + fuzzyRigth);
        }
    }
```

2) Нейрон

```
public interface INeuron<T>
    {
        /// <summary>
        /// Weights of the neuron
        /// </summary>
        List<double> Weights { get; }

        /// <summary>
```

```
        /// Offset/bias of neuron (default is 0)
        /// </summary>
        double Bias { get; set; }

        /// <summary>
        /// Last calculated state in Activate
        /// </summary>
        T LastState { get; set; }

        /// <summary>
        /// Last calculated NET in NET
        /// </summary>
        T WeightedSum { get; set; }

        IList<INeuron<T>> Children { get; }

        IList<INeuron<T>> Parents { get; }

        IFunction ActivationFunction { get; set; }

        double ActivationFunctionDerivative { get; }

        double ErrorDerivative { get; set; }

        /// <summary>
        /// Compute NET of the neuron by input vector
        /// </summary>
        /// <param name="inputVector">input vector (must be the same dimension
as was set in SetDimension)</param>
        /// <returns>NET of neuron</returns>
        T computeWeightedSum(T[] inputVector);
        /// <summary>
        /// Compute state of neuron
        /// </summary>
        /// <param name="inputVector">input vector (must be the same dimension
as was set in SetDimension)</param>
        /// <returns>State of neuron</returns>
        T Activate(T[] inputVector);
    }
```

3) Шар нейронної мережі

```
public interface ILayer<T>
    {
        /// <summary>
```

```
        /// Get last output of the layer
        /// </summary>
        T[] LastOutput { get; }

        /// <summary>
        /// Get neurons of the layer
        /// </summary>
        INeuron<T>[] Neurons { get; }

        /// <summary>
        /// Get input dimension of neurons
        /// </summary>
        int InputDimension { get; }


        /// <summary>
        /// Compute output of the layer
        /// </summary>
        /// <param name="inputVector">Input vector</param>
        /// <returns>Output vector</returns>
        T[] Compute(double[] inputVector);
    }
```

4) Нечітка нейронна мережа

```
public class MultiLayerNetwork:IMultilayerNeuralNetwork
    {
        Layer[] layers;
        ILearningStrategy<IMultilayerNeuralNetwork> learnStrategy;
        int inputDimension=0;
        int conectionsCount = 0;

        public MultiLayerNetwork(int[] NeuronsInLayers, int InputDimension)
        {
            Random rnd = new Random();
            inputDimension = InputDimension;
            layers = new Layer[NeuronsInLayers.Length];
            layers[0] = new Layer(NeuronsInLayers[0], InputDimension);
            double[] weights = new double[InputDimension];
            for (int i = 0; i < InputDimension; i++)
            {
                weights[i] = 1;
            }
```

```
        for (int i = 0; i < NeuronsInLayers[0]; i++)
        {
            layers[0].AddNeuron(new MLNeuron(weights.ToList()));
            conectionsCount += weights.Length;
        }
        for (int i = 1; i < NeuronsInLayers.Length; i++)
        {
            layers[i] = new Layer(NeuronsInLayers[i], NeuronsInLayers[i - 1]);
            for (int j = 0; j < NeuronsInLayers[i]; j++)
                layers[i].AddNeuron(new MLNeuron(NeuronsInLayers[i - 1],
rnd));
            conectionsCount += NeuronsInLayers[i - 1];
        }
        LearningConfig config = new LearningConfig();
        config.BatchSize = -1;
        config.MaxEpoches = 1000;
        config.MinError = 0.000001;
        config.MinErrorChange = 0.0000000001;
        learnStrategy = new GeneticLearning(this, config);
    }


    public ILayer<double>[] Layers
    {
        get { return layers; }
    }
    public int InputDimension
    {
        get { return inputDimension; }
    }
    public int ConectionsCount
    {
        get { return conectionsCount; }
    }
    public FuzzyDouble [] ComputeOutput(FuzzyDouble [] inputVector)
    {
        FuzzyDouble [] outputVector = layers[0].Compute(inputVector);
        for (int i = 1; i < layers.Length; i++)
        {
            outputVector = layers[i].Compute(outputVector);
        }
        return outputVector;
    }
```

```
        public void Train(IList< FuzzyDouble > Data)
        {
            learnStrategy.train(Data);
        }
        public void Train(IList< FuzzyDouble > Train, IList< FuzzyDouble >
Validate, IList< FuzzyDouble > Check)
        {
            learnStrategy.train(Train,Validate,Check);
        }
```

5) Оператор кроссінговера

```
public class Crossingover
    {
        public MultiLayerNetwork[] copulation(IMultilayerNeuralNetwork N1,
IMultilayerNeuralNetwork N2, int k)
        {
            MultiLayerNetwork[]  childs = new MultiLayerNetwork[2];
            int[] neurons = new int[N1.Layers.Length];
            for (int j = 0; j < N1.Layers.Length; j++)
            {
                neurons[j] = N1.Layers[j].Neurons.Length;
            }
            for (int j = 0; j < childs.Length; j++)
            {
                childs[j] = new MultiLayerNetwork(neurons, N1.InputDimension);
            }

            int i = 0;

            for (int layer = 0; layer < N1.Layers.Length; layer++)
            {
                for (int j = 0; j < N1.Layers[layer].Neurons.Length; j++)
                {
                    for (int n = 0; n < N1.Layers[layer].Neurons[j].Weights.Count;
n++, i++)
                    {
                        if (i < k)
                        {
                            double a = N1.Layers[layer].Neurons[j].Weights[n], b =
N2.Layers[layer].Neurons[j].Weights[n];
                            childs[0].Layers[layer].Neurons[j].Weights[n] = a;
                            childs[1].Layers[layer].Neurons[j].Weights[n] = b;
                        }
```

```
                else
                {
                    double a = N1.Layers[layer].Neurons[j].Weights[n], b =
N2.Layers[layer].Neurons[j].Weights[n];
                    childs[0].Layers[layer].Neurons[j].Weights[n] = b;
                    childs[1].Layers[layer].Neurons[j].Weights[n] = a;
                }
            }
        }
    }
    return childs;
}
```

6) Оператор мутаціі

```
internal class Mutation : IMutation
  {
    List<double> mutationsRange;
    Random _rnd;
    int maxMutations;
    internal Mutation(Random rnd, List<double> MutationsRange, int
MaxMutations)
    {
      _rnd = rnd;
      mutationsRange = MutationsRange;
      maxMutations = MaxMutations;
    }
    public void mutate(IMultilayerNeuralNetwork network)
    {
      int mutations = _rnd.Next(1, maxMutations);
      for (int i = 0; i < mutations; i++)
      {
        double rate = _rnd.NextDouble();
        if (rate < 0.2)
        {
          int layer = _rnd.Next(0, network.Layers.Length);
          int neuron = _rnd.Next(0, network.Layers[layer].Neurons.Length);
          int weigth = _rnd.Next(0,
network.Layers[layer].Neurons[neuron].Weights.Count);
          network.Layers[layer].Neurons[neuron].Weights[weigth] =
mutationsRange[_rnd.Next(0, mutationsRange.Count)];
        }
      }
```

```
        }
    }
```

7) Оператор селекції

```
int[] selectParents(double[] errors)
        {
            int[] parentsIndexes = new int[2];
            double random = rnd.NextDouble();
            for(int i=0;i<errors.Length;i++)
            {
                if (random < errors[i])
                {
                    parentsIndexes[0] = i;
                    break;
                }
            }
            do
            {
                random = rnd.NextDouble();
                for (int i = 0; i < errors.Length; i++)
                {
                    if (random < errors[i])
                    {
                        parentsIndexes[1] = i;
                        break;
                    }
                }
            } while (parentsIndexes[1] == parentsIndexes[0]);
            return parentsIndexes;
        }
```

8) Алгоритм навчання нейронної мережі з застосуванням генетичних алгоритмів

```
class GeneticLearning: ILearningStrategy<IMultilayerNeuralNetwork>

    {

        private LearningConfig config;

        private Random rnd;

        private static Logger logger = LogManager.GetCurrentClassLogger();
```

```
private IMultilayerNeuralNetwork network;
List<double> weigthesRange;
private double currentTrainError = Single.MaxValue;
private double lastTrainError = 0;
private double currentValidateError = Single.MaxValue;
private double lastValidateError1 = 0;
private double lastValidateError2 = 0;
private double lastValidateError3 = 0;
private double currentCheckError = 0;
Crossingover crossingover;
Mutation mutation;
int elite = 2;
private IMultilayerNeuralNetwork networkL1;
private IMultilayerNeuralNetwork networkL2;
internal GeneticLearning(IMultilayerNeuralNetwork Network,
LearningConfig Config)
{
    config = Config;
    network = Network;
    //_random = new Random(Helper.GetSeed());
    rnd = new Random();
}
public void train(IList<DoubleData> train, IList<DoubleData> validate,
IList<DoubleData> check)
{
    int epochNumber = 0;
```

```
logger.Info("Start learning...");
MultiLayerNetwork[] population = Initialize(128);
string Train = @"E:\Дима\NeuralNetworks\result.txt";
System.IO.StreamWriter srT = new System.IO.StreamWriter(Train, false);
do
{
    lastTrainError = currentTrainError;
    DateTime dtStart = DateTime.Now;
    population=Iteration(population,train);
    //recalculating error on all train
    currentTrainError = 0;
    for (int i = 0; i < train.Count; i++)
    {
        double[] realOutput = population[0].ComputeOutput(train[i].Input);
        currentTrainError += config.ErrorFunction.Calculate(train[i].Output,
realOutput);
    }
    currentTrainError /= train.Count;
    lastValidateError3 = lastValidateError2;
    lastValidateError2 = lastValidateError1;
    lastValidateError1 = currentValidateError;
    //recalculating error on all validate
    currentValidateError = 0;
    for (int i = 0; i < validate.Count; i++)
    {
```

```
                double[] realOutput =
population[0].ComputeOutput(validate[i].Input);

                currentValidateError +=
config.ErrorFunction.Calculate(validate[i].Output, realOutput);

            }

            currentValidateError /= validate.Count;

            currentCheckError = 0;

            for (int i = 0; i < check.Count; i++)

            {

                double[] realOutput = population[0].ComputeOutput(check[i].Input);

                currentCheckError +=
config.ErrorFunction.Calculate(check[i].Output, realOutput);

            }

            currentCheckError /= check.Count;

            epochNumber++;

            Console.WriteLine("Eposh #" + epochNumber.ToString() +

                            " finished; current error is " + currentTrainError.ToString()
+

                            "; it takes: " +

                            (DateTime.Now - dtStart).Duration().ToString());

            logger.Trace("Eposh #" + epochNumber.ToString() +

                            " finished; current error is " + currentTrainError.ToString()
+

                            "; it takes: " +

                            (DateTime.Now - dtStart).Duration().ToString());

            string ss = epochNumber.ToString() + "," +
currentTrainError.ToString().Replace(",", ".") + "," +
```

```
currentValidateError.ToString().Replace(",", ".") + "," +
currentCheckError.ToString().Replace(",", ".");
            srT.WriteLine(ss);
        }           while (epochNumber < config.MaxEpoches &&
currentTrainError > config.MinError);
        srT.Close();
        UpdateWeightes(population[0]);
    }
    public void train(IList<DoubleData> train)
    {
        int epochNumber = 0;
        logger.Info("Start learning...");
        MultiLayerNetwork[] population = Initialize(128);
        string Train = @"E:\Дима\NeuralNetworks\train.txt";
        System.IO.StreamWriter srT = new System.IO.StreamWriter(Train, false);
        do
        {
            lastTrainError = currentTrainError;
            DateTime dtStart = DateTime.Now;
            population = Iteration(population, train);
            //recalculating error on all train
            currentTrainError = 0;
            for (int i = 0; i < train.Count; i++)
            {
                double[] realOutput = population[0].ComputeOutput(train[i].Input);
```

```
                currentTrainError += config.ErrorFunction.Calculate(train[i].Output,
realOutput);
            }
            currentTrainError /= train.Count;
            epochNumber++;
            Console.WriteLine("Eposh #" + epochNumber.ToString() +
                        " finished; current error is " + currentTrainError.ToString()
+
                        "; it takes: " +
                        (DateTime.Now - dtStart).Duration().ToString());
            logger.Trace("Eposh #" + epochNumber.ToString() +
                        " finished; current error is " + currentTrainError.ToString()
+
                        "; it takes: " +
                        (DateTime.Now - dtStart).Duration().ToString());


            srT.WriteLine(epochNumber.ToString() + "," +
currentTrainError.ToString().Replace(",", "."));
        } while (epochNumber < config.MaxEpoches && currentTrainError >
config.MinError);
        srT.Close();
        UpdateWeightes(population[0]);
    }
    public void UpdateWeightes(MultiLayerNetwork newNetwork)
    {
        // Update weights and bias
        for (int layerIndex = 0; layerIndex < network.Layers.Length; layerIndex++)
```

```
        {
            for (int neuronIndex = 0; neuronIndex <
network.Layers[layerIndex].Neurons.Length; neuronIndex++)
            {
                network.Layers[layerIndex].Neurons[neuronIndex].Bias =
newNetwork.Layers[layerIndex].Neurons[neuronIndex].Bias;
                for (int weightIndex = 0; weightIndex <
network.Layers[layerIndex].Neurons[neuronIndex].Weights.Count;
weightIndex++)
                {

network.Layers[layerIndex].Neurons[neuronIndex].Weights[weightIndex] =
newNetwork.Layers[layerIndex].Neurons[neuronIndex].Weights[weightIndex];
                }
            }
        }
    }
    public MultiLayerNetwork[] Iteration(MultiLayerNetwork[] population,
IList<DoubleData> data)
    {
        MultiLayerNetwork[] childs = new
MultiLayerNetwork[population.Length];
        double[] errors = new double[population.Length];
        for (int n = 0; n < population.Length; n++)
        {
            for (int i = 0; i < data.Count; i++)
            {
                double[] realOutput = population[n].ComputeOutput(data[i].Input);
```

```
                errors[n] += config.ErrorFunction.Calculate(data[i].Output,
realOutput);
            }
            errors[n] /= data.Count;
        }
        Array.Sort(errors, population);
        double sum = Sum(errors);
        for (int i=0;i< errors.Length ; i++)
        {
            errors[i] = errors[i] / sum;
        }
        for (int i = 0; i < errors.Length; i++)
        {
            errors[i] = 1 / errors[i];
        }
        sum = Sum(errors);
        for (int i = 0; i < errors.Length; i++)
        {
            errors[i] = errors[i] / sum;
        }
        for (int i = 1; i < errors.Length; i++)
        {
            errors[i] += errors[i-1];
        }
        for (int i = 0; i <  elite; i++)
        {
```

```
            childs[i] = population[i];
        }
        for (int i =  elite; i < population.Length; i+=2)
        {
            int[] indexes= selectParents(errors);
            MultiLayerNetwork[] temp =
crossingover.copulation(population[indexes[0]],
population[indexes[1]],rnd.Next(0,network.ConectionsCount));
            foreach(MultiLayerNetwork n in temp)
            {
                mutation.mutate(n);
            }
            childs[i] = temp[0];
            childs[i + 1] = temp[1];
        }
        return childs;
    }
    MultiLayerNetwork[] Initialize(int N)
    {
        MultiLayerNetwork[] population = new MultiLayerNetwork[N];
        crossingover = new Crossingover();

        weigthesRange = new List<double>();
        int M = 1000000;
        for(int i=0;i<M;i++)
        {
```

```
            weigthesRange.Add(((double)i / M) * 2 - 1);
        }
        mutation = new Mutation(rnd, weigthesRange,10);
        int[] neurons = new int[network.Layers.Length];
        for (int j = 0; j < network.Layers.Length; j++)
        {
            neurons[j] = network.Layers[j].Neurons.Length;
        }
        for (int j = 0; j < population.Length; j++)
        {
            population[j] = new MultiLayerNetwork(neurons,
network.InputDimension);
        }


        for (int layer = 0; layer < network.Layers.Length; layer++)
        {
            for (int j = 0; j < network.Layers[layer].Neurons.Length; j++)
            {
                for (int n = 0; n < network.Layers[layer].Neurons[j].Weights.Count;
n++)
                {
                    for (int k = 0; k < N; k++)
                    {
                        double A = weigthesRange[rnd.Next(0, M)];
                        population[k].Layers[layer].Neurons[j].Weights[n] = A;
                    }
```

```
            }
        }
    }
    return population;
}
```