Fejlesztői dokumentáció

Neurális hálózat

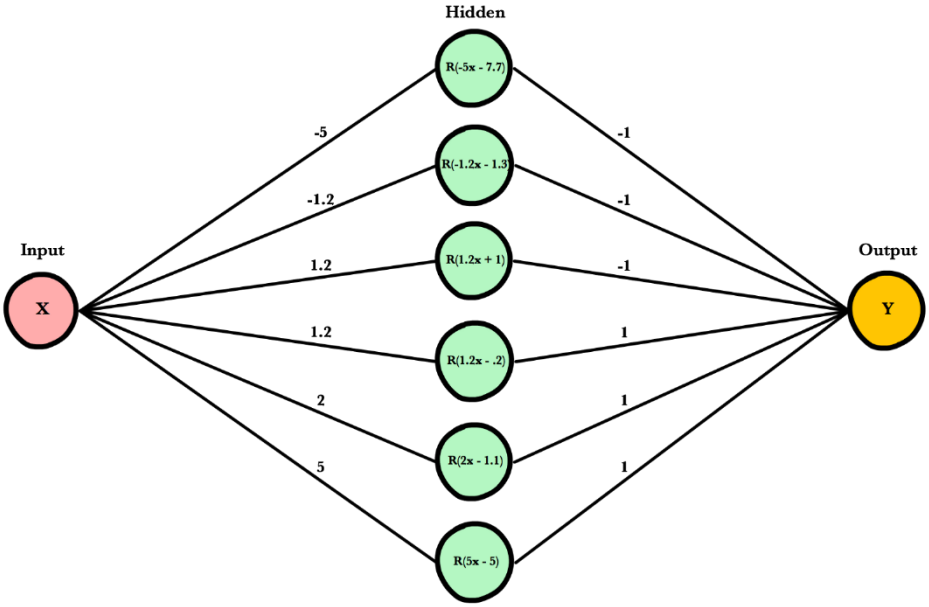
Modell: Bayesi

Mikolics Réka Szilvia, Kondics Milán, Jónás Ámos

2019. június 23.

Neurális hálózat

A neurális hálózat felépítése:



A neurális hálózat rétegekből épül fel (input, hidden, output), ezen rétegek pedig neuronokból. A rétegek közötti átvitelt az súlyozott élek valósítják meg. Ezen az éleken lévő súlyok változtatásával, javításával értjük el azt, hogy a hálózat kimeneti értéke megfelelő legyen.

Jelen esetben három réteggel dolgozunk. Egy bemenő (input), egy rejtett (hidden) és egy kimeneti (output) réteggel. A bemenő rétegbe érkezik a bemenő adat, amely tovább halad a belső rejtett rétegbe az ezeket összekötő éleken keresztül. A rejtett rétegből pedig a kimeneti rétegbe kerül az ezeket összekötő éleken keresztül.

Az első, azaz bemeneti rétegben egy neuront használunk, amely a mi esetünkben a függvényközelítésnél az x koordinátát kapja meg. A rejtett rétegben 20 neuront használunk, tapasztalataink alapján ez tűnt a leghatékonyabbnak. A kimeneti rétegben egy neuront használunk, amely a pont y koordinátáját adja meg.

A hálózat tanításának működési elve:

A hálózat kap egy bizonyos mennyiségű input és output párt, jelen esetben a függvény egy pontjának x és y koordinátáit. Ezeket átáramoltatja a hálózaton és kiszámolja az élek súlyának hibáit. A javítás pedig úgy zajlik, hogy a hálózat összes élét egyszerre javítja, vagyis minden élhez hozzá ad egy random generált számot egy nagyon kicsi intervallumról. Ezután újra hibát számolunk. Ha a javítás után a hiba csökkent, akkor a hálózat elfogadja a javítást, ha nem, akkor egy új értékkel próbál javítani. Ez addig fut, amíg az hálózat átlagos hibája kicsi nem lesz.

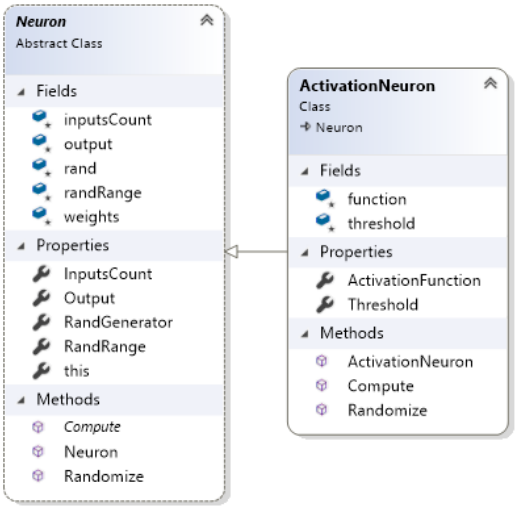
Az n. mintapár hibája:

$$\mathcal{E}(n) = \frac{1}{2}\sum_{j}\left(d_j(n) - y_j^{(2)}\right)^2,$$

Ahol d_j a j. kívánt output, y_j a j. hálózat által adott output.

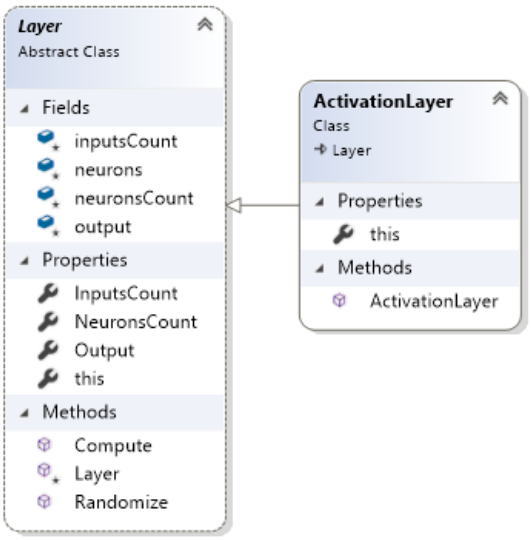
A program felépítése

A program két projektből áll. Az első, a **NeuralNetwork** a neurális hálózat alapjait valósítja meg.



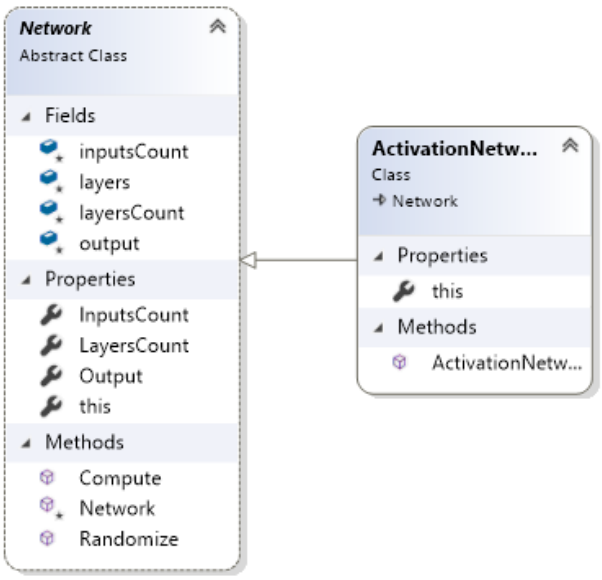
A neuronok a neurális hálózat alapjait képezik. Rendelkeznek egy a beléjük menő élek súlyaival, a belőlük kimenő outputtal, amelyet a Compute függvény számol ki.

Az ActivationNeuron osztály a belső réteggel rendelkező, úgynevezett többrétegű neurális hálózatok alapjai. A különbség egy sima neuron és egy ActivationNeuron között az az, hogy az ActivationNeuron rendelkezik egy ActivationFunctionnal.



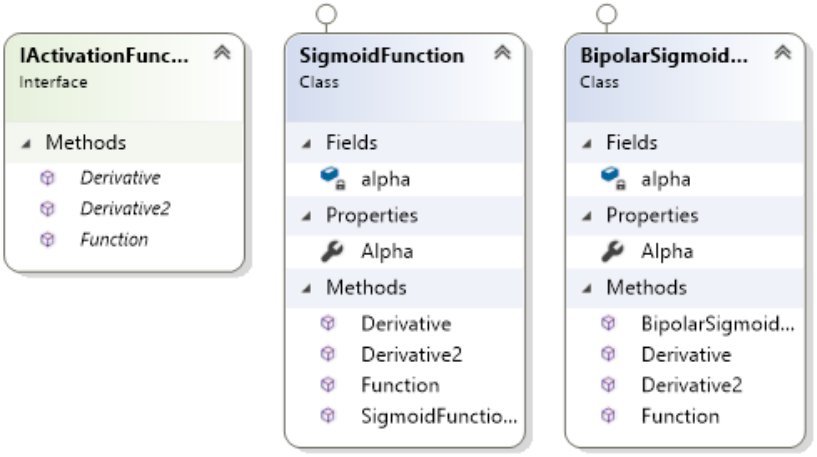
A Layer osztály a hálózat rétegeit valósítja meg. Összefogja a réteghez tartozó neuronokat. Rendelkezik egy Compute függvénnyel, amely a réteg kimeneti értékét határozza meg.

Az ActivationLayer a többrétegű hálózatokhoz megvalósított réteg típus.



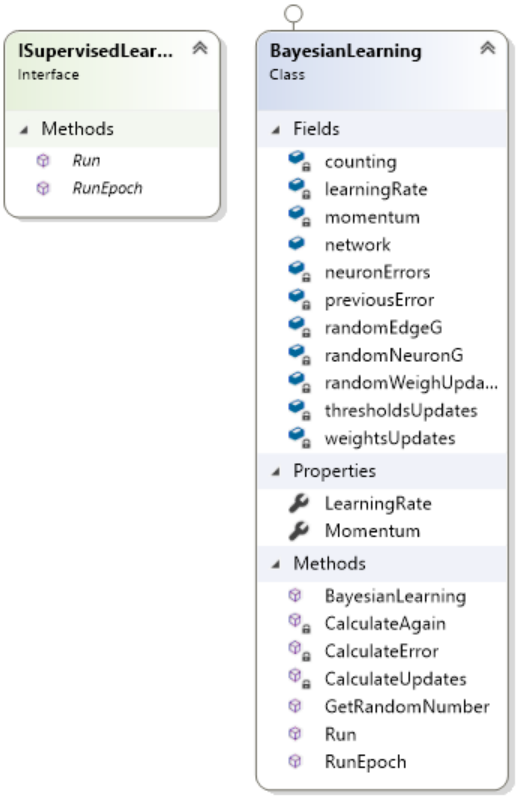
A Network osztály magát a neurális hálózatot valósítja meg, amely a rétegek összessége. Rendelkezik Compute függvénnyel, amely a hálózat kimenetét adja meg.

Az ActivationNetwork a többrétegű hálózatok megvalósítására használt osztály.



Az IActivationFunction az előzőleg látott Activation előtagú neuron, réteg illetve hálózat osztályhoz tartozozó függvények interfésze.

Ebből megvalósíthatunk bármilyen függvényt.



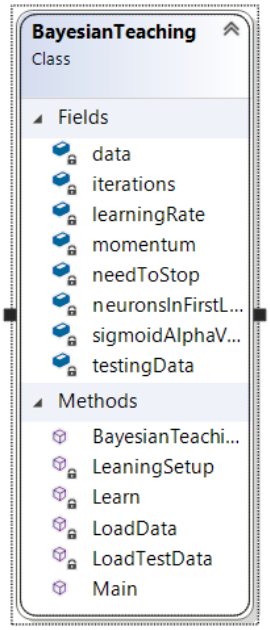
Az ISupervisedLearning interfész azon tanulási metódusok interfésze, amelyeknél a tanulás megadott input-output párokon alapszik.

Ez az interfész az alapja a BayesianLearning osztálynak, amely a fent ismertetett tanulási módszert valósítja meg.

Rendelkezik RunEpoch függvénnyel, amely a hálózat tanítására szolgál. Bemeneti paraméterei az input és output párok. Ez a függvény hívja meg a Run függvényt.

A Run függvény kezeli a hálózatot, itt hívódnak meg a CalculateUpdates (hálózat javításának számolása), Compute (hálózaton az input átáramoltatása), CalculateError (hiba kiszámítása) függvények, illetve ez kezeli hogy a javítás elfogadott-e vagy sem.

A második projekt a **NeuralNetworkTeaching**. Ebben a projektben kapnak helyet azon osztályok, amely a neurális hálózat tanítását kezelik.



Jelen esetben ez a tanítás a BayesianTeaching, amely a BayesianLearning osztályt kezeli.

Ezen osztályban történik a tanítási adatok beolvasása, a tanulás beállítása (iterációk száma, tanulás gyorsasága, stb.), illetve a tesztelési folyamat.

A program részletes ismertetése

Neuron.cs

```
1.  //Neuron konstruktora
2.  public Neuron(int inputs)
3.  {
4.      //súlyok elérése
5.      inputsCount = Math.Max(1, inputs);
6.      weights = new double[inputsCount];
7.      //súlyok randomizálása
8.      Randomize();
9.  }
```

Itt hozzuk létre magát a neuront, amelyhez tartoznak élek (inputsCount), amely biztosan minimum egy darab. Ezen élekhez tartoznak súlyok, amelyet random állítunk be a Randomize() metódussal.

```
1.  //Neuron randomizálása
2.  //A súlyokat random értékekkel inicializálja az adott intervallumon belül
3.  public virtual void Randomize()
4.  {
5.      double d = randRange.Length;
6.
7.      // súlyok randomizálása
8.      for (int i = 0; i < inputsCount; i++)
9.          weights[i] = rand.NextDouble() * d + randRange.Min;
10. }
```

ActivationNeuron.cs

```
1.  public override double Compute(double[] input)
2.  {
3.      if (input.Length != inputsCount)
4.          throw new ArgumentException();
5.
6.      //összeg
7.      double sum = 0.0;
8.
9.      //súlyozott összeg kiszámítása
10.     for (int i = 0; i < inputsCount; i++)
11.     {
12.         sum += weights[i] * input[i];
13.     }
14.     sum += threshold;
15.
16.     return (output = function.Function(sum));
17. }
```

A neuron kimeneti értékének kiszámítása.

Layer.cs

```
1.  protected Layer(int neuronsCount, int inputsCount)
2.  {
3.      this.inputsCount = Math.Max(1, inputsCount);
4.      this.neuronsCount = Math.Max(1, neuronsCount);
5.      // neuronok létrehozása
6.      neurons = new Neuron[this.neuronsCount];
7.      // output tömb létrehozása
8.      output = new double[this.neuronsCount];
9.  }
```

A rétegeket megvalósító osztály. A rétegek neuronok összessége, így a konstruktor egyik paramétere is ez, illetve a rétegekhez tartozó élek. Itt példányosítjuk, azaz létrehozzuk a réteghez tartozó neuronokat és a réteg neuronjaihoz tartozó kimenetet.

```
1.  public virtual double[] Compute(double[] input)
2.  {
3.      // kiszámoljuk az összes neuronnál
4.      for (int i = 0; i < neuronsCount; i++)
5.          output[i] = neurons[i].Compute(input);
6.
7.      return output;
8.  }
```

A Compute függvény a kimeneti értéket számítja ki, létrehoz egy output tömböt, amibe belerakja a réteghez tartozó neuronok kimenetét.

Network.cs

```
1.  protected Network(int inputsCount, int layersCount)
2.  {
3.      this.inputsCount = Math.Max(1, inputsCount);
4.      this.layersCount = Math.Max(1, layersCount);
5.      // layerek összessége
6.      layers = new Layer[this.layersCount];
7.  }
```

A neurális háló osztálya, itt hozzuk lére a hálózathoz tartozó rétegeket.

```
1.  public virtual double[] Compute(double[] input)
2.  {
3.      output = input;
4.
5.      //kiszámoljuk az összes rétegre
6.      foreach (Layer layer in layers)
7.      {
8.          output = layer.Compute(output);
9.      }
10.
11.     return output;
12. }
```

A Compute a hálózat kimenetét számolja ki.

BayesianLearning.cs

A RunEpoch a függvényt kell meghívni a neurális hálózat tanításához :

```
1.  public double RunEpoch(double[][] input, double[][] output)
2.  {
3.      double error = 0.0;
4.
5.      // az összes példára futtatjuk a tanulást
6.      for (int i = 0, n = input.Length; i < n; i++)
7.      {
8.          error += Run(input[i], output[i], input, output);
9.      }
10.
11.     //átlagos errort adjuk vissza
12.     return error;
13. }
```

A Run függvény kezeli a tanulást:

```
1.  public double Run(double[] input, double[] output, double[][] inp, double[][] outp)
2.   {
3.       while (true)
4.       {
5.           //random számot hozzáad az összes élhez
6.           CalculateUpdates(input);
7.           //kiszámolja a hálózat kimenetét így
8.           network.Compute(input);
9.           //a hálózat átlagos hibája
10.          double avgerror = AvgError(inp, outp) / 200;
11.          //az output hibája
12.          error = CalculateError(output);
13.          //ha javult, vagy elértük a kívánt hibaarányt, akkor elfogadunk, ha nem, akkor
14.          //töröljük a változtatásokat
15.          if (avgerror < previousError || avgerror < 0.03)
16.          {
17.              Console.WriteLine("AVG ERRROR: " + avgerror);
18.              previousError = avgerror;
19.              break;
20.          }
21.          else
22.          {
23.              deleteChanges();
24.          }
25.      }
26.
27.      return error;
28.  }
```

A CalculateError számolja a hibát $\mathcal{E}(n) = \frac{1}{2}\sum_j \left(d_j(n) - y_j^{(2)}\right)^2,$ alapján:

```
1.  private double CalculateError(double[] desiredOutput)
2.  {
3.      ActivationLayer layer;
4.      double error = 0, e;
5.      double output;
6.      int layersCount = network.LayersCount;
7.      layer = network[layersCount - 1];
8.      for (int i = 0; i < desiredOutput.Length; i++)
9.      {
10.         output = layer[i].Output;
11.         e = desiredOutput[i] - output;
12.         error += (e * e);
13.         counting += 1;
14.     }
15.     return error / 2.0;
16. }
```

A javító függvény:

Végig megy az összes élen, és hozzáad egy random kicsi számot.

```
1.  private void CalculateUpdates(double[] input)
2.  {
3.      ActivationNeuron neuron;
4.      ActivationNeuron[] edge;
5.      ActivationLayer layer;
6.
7.      neuronChanges = new double[network.LayersCount,100];
8.      for (int i = 0, n = network.LayersCount; i < n; i++)
9.      {
10.         layer = network[i];
11.         for (int j = 0, m = layer.NeuronsCount; j < m; j++)
12.         {
13.             neuron = layer[j];
```

```
            for (int k = 0, s = neuron.InputsCount; k < s; k++)
            {
                double randomWeight = GetRandomNumber(-0.5, 0.5);
                neuronChanges[i,k] = randomWeight;
                neuron[k] += randomWeight;
            }
        }
    }

}
```

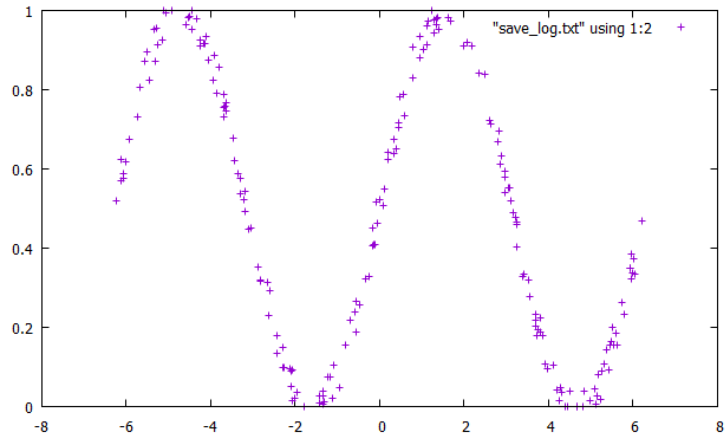
BayesianTeaching.cs

A tanulási függvénye:

Kezeljük a bemenő input és output adatokat. Példányosítjuk a hálózatot és a tanuló osztályt.

```
private void Learn()
        {
            int samples = data.GetLength(0);
            double[][] input = new double[samples][];
            double[][] output = new double[samples][];

            for (int i = 0; i < samples; i++)
            {
                input[i] = new double[1];
                output[i] = new double[1];
                input[i][0] = (data[i, 0]);
                output[i][0] = (data[i, 1]);
            }

            ActivationNetwork network = new ActivationNetwork(
                new BipolarSigmoidFunction(sigmoidAlphaValue),
                1, neuronsInFirstLayer, 1);
            BayesianLearning teacher = new BayesianLearning(network);
            teacher.LearningRate = learningRate;

            int iteration = 1;
            while (!needToStop)
            {
                double error = teacher.RunEpoch(input, output) / samples;
                Console.WriteLine("The avg error is: " + error);
                iteration++;

                //Mikor álljunk meg? Ha az iterációk száma eléri a megadottat, vagy a hiba kisebb mint valami
                if ((iterations != 0) && (iteration > iterations) || error < 1)
                {
                    for (int i = 0, n = input.Length; i < n; i++)
                    {
                        WriteData(input[i], teacher);
                    }
                    break;

                }
            }
                        …
                        …
```

A bemenő paraméterek kirajzolása:



A neurális hálózat által adott paraméterek kirajzolása: