ANN: Number Recognition

COMPUTATIONAL INTELIGENCE

Eleftherios Trivizakis ΜΠ143, Konstantinos Tzagkarakis ΜΠ141 | Computational Intelligence | 12 Δεκεμβρίου 2016

Content

Introduction

This project is an attempt of implementing an artificial neural network in a Java application for recognizing digits of numbers from 0 to 9.

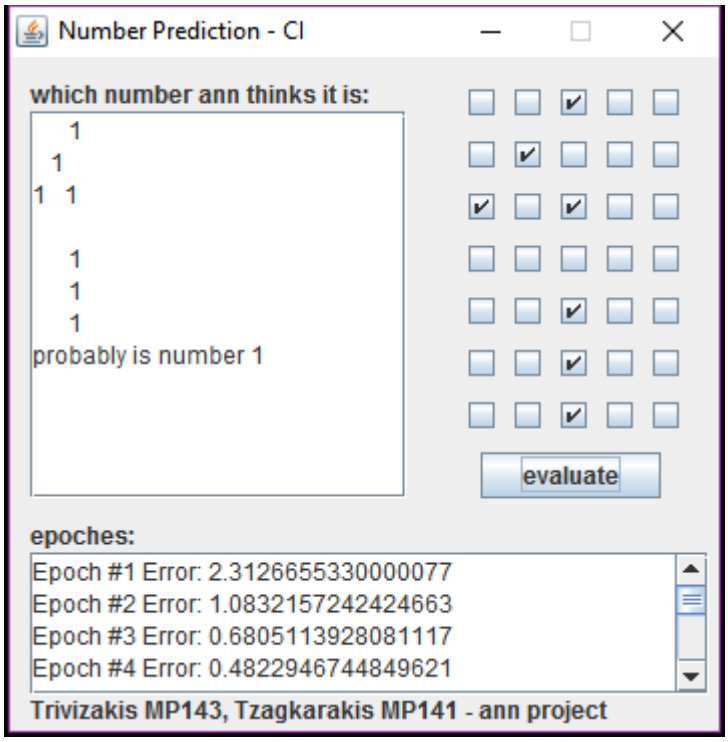
The datasets are created from the high level object automatically by providing only the ideal input, which is an array of ten arrays of pixels, 5 by 7.

After the training, the application should be able to distinguish, from an input of pixels, checkboxes actually, the most probable number even if the user provides a distorted input. These inputs are given in real time.

The output is an estimation out of the ten number digits. The accuracy of the application will be tested under different least square errors but with the same input from the user.

The main idea is that the training occurs as the application start without any interaction needed from the user and the graphical user interface provide the components for the input and the output.

Implementation and Experimental Methods



The platform that we used to create the application is Java Development Kit with the Eclipse IDE. From implementing the artificial neural network, we use the Encong API which is also written in Java. All the tools and libraries are open source as the requirement of the project suggests.

The Encogn API provides all the components, like dataset generators, neurons, error methods, training methods, testing classes and data normalizers.

```
public static MLDataSet generateTraining(){
    MLDataSet result = new BasicMLDataSet();
    for(int i=0;i<idealTarget.length;i++){
        BasicMLData ideal = new BasicMLData(idealTarget.length);

        //setup input
        MLData input = realTrainingData(idealTarget[i]);

        //setup ideal
        for(int j=0;j<idealTarget.length;j++)
        {
            if( j==i )
                ideal.setData(j,1);
            else
                ideal.setData(j,-1);
        }
        result.add(input,ideal);
    }
    return result;
}
```

As a dataset we used the abstract method generateTraining() which creates the MLDataSet for the training. The input for this method is the ideal input for each number in an array of pixels 5 by 7.

-1	-1	1	-1	-1
-1	1	1	-1	-1
1	-1	1	-1	-1
-1	-1	1	-1	-1
-1	-1	1	-1	-1
-1	-1	1	-1	-1

Table 1 - a typical input, number 1

Also, it is worth to mention that the real input [Table 1] consists out of -1 and 1 but in order to be more readable from the user, the -1 is replaced by white spaces, that's why at the output textfield only the 1s are displayed.

The generated dataset is the main input for the trainAnn() method, which as its name suggests, trains the artificial neural network so it can identify inputs similar to the ideal input. In each iteration of train object is produced an epoch with different square error.

```
private static void trainAnn(){
    double learningRate = 0.01;
    MLDataSet training = generateTraining();
    MLTrain train = new TrainAdaline(network,training,learningRate);
    double error;
    int epoch = 1;
    do {
        train.iteration();
        error = train.getError();
        annGui.setTaOutput(annGui.getTaOutput()+
                "Epoch #"+epoch+" Error: "+error+"\n");
        epoch++;
    } while(train.getError() > learningRate);

    annGui.setTaOutput(annGui.getTaOutput()+
            "Final error value for our ann: " + error);
}
```

The testing is using the high level neural network's object in conjunction with the input data from the user, so it can recognize the most probable output. More specifically the method winner() of the ann object calculate the output as seen below.

```
public static void testInput(String[] inputToRecogn){
    int output = network.winner(realTrainingData(inputToRecogn));
    for(int i=0; i<inputToRecogn.length; i++){
        annGui.setTextArea(annGui.getTextArea()+
                inputToRecogn[i].replaceAll(" ", "    ").replaceAll("0", "1")+"\n");
    }
    annGui.setTextArea(annGui.getTextArea()+
            "probably is number "+ output +"\n");
}
```

Experimental part

For this part we wanted to investigate how different square errors affect the performance of an artificial neural network. We test the algorithm under 1, 0.1, 0.05, 0.01, 0.001 errors.

Square error	Training Epochs	Performance	
1	Never converges	bad	No recognition
0.1	Never converges	bad	No recognition
0.05	Short, ~25 epochs	good	Recognizes only inputs like the ideal training set
0.01	~50 epochs	the best	Recognizes even distorted inputs
0.001	More than 1000	bad	Rare recognition or only inputs like the ideal training set

Table 2 - training and performance

Indicative, we include in appendices [Graphs 1] the actual number of epochs with the current error. Except the 0.001, simply because it's neither useful nor practical to include 1000+ epochs.

As it can be observed, the only viable square error target should be around 0.01, because the neural network converges to this error relatively fast and the

performance is the optimal. It can recognize even distorted inputs in contrast to the other test errors.

For square errors greater 0.1 the neural network never converges to the wanted state. As a result, those networks barely can recognize the ideal input and not even, in some cases, where the 5*7 inputs are similar, like the 9, 0, 6 ,8.

Summary and Conclusion

In conclusion, we implement a single layer ADELINE artificial neural network for 5*7 number digits. we train the ann with the high level training and dataset objects provided by the open source Encogn API to obtain optimal weights values. I also find useful applications for the ANN I generate. By doing this project, we have practiced using what we have learned in this course.

Our program probably does not beat the state of the art in number digit recognition. However, we have for the first time observed the practical problems of using the artificial neural networks, for instance, designing the architecture of an ann, choosing appropriate activation functions or libraries, the ADALINE pattern, convergence issues, stopping criteria, generalization ability. This experience will definitely be helpful for my future research.

Appendices

	A	B	C	D
1	1	0.1	0.05	0.01
2	Epoch #1 Error: 3.2851768000285535E15	Epoch #1 Error: 478.23556349104303	Epoch #1 Error: 3.9547777210772073	Epoch #1 Error: 2.501976778950257
3	Epoch #2 Error: 1.4177919265447292E34	Epoch #2 Error: 169895.6468891488	Epoch #2 Error: 2.150503237309855	Epoch #2 Error: 0.8051968033556564
4	Epoch #3 Error: 6.1187135602121826E52	Epoch #3 Error: 5.8413013742195594E7	Epoch #3 Error: 1.8533873763339201	Epoch #3 Error: 0.4859427149102776
5	Epoch #4 Error: 2.6406498017058217E71	Epoch #4 Error: 1.8044338594394786E10	Epoch #4 Error: 1.1036647686558368	Epoch #4 Error: 0.3423477790183136
6	Epoch #5 Error: 1.1296227624969776E90	Epoch #5 Error: 5.420759640759826E12	Epoch #5 Error: 0.7659166215011599	Epoch #5 Error: 0.2659514372445983
7	Epoch #6 Error: 4.918387925902353E108	Epoch #6 Error: 1.6122081128132186E15	Epoch #6 Error: 0.47005031732916036	Epoch #6 Error: 0.21930043280988874
8	Epoch #7 Error: 2.1225741482201032E127	Epoch #7 Error: 4.779114367068641E17	Epoch #7 Error: 0.3575399223870116	Epoch #7 Error: 0.18774308653068959
9	Epoch #8 Error: 9.160981424047155E145	Epoch #8 Error: 1.4151530987669345E20	Epoch #8 Error: 0.29462595612522974	Epoch #8 Error: 0.16452382378839428
10	Epoch #9 Error: 3.9533407068200242E164	Epoch #9 Error: 4.188865449674195E22	Epoch #9 Error: 0.27280490569201293	Epoch #9 Error: 0.14640132051151685
11	Epoch #10 Error: 1.706141045958429E183	Epoch #10 Error: 1.2398260860005588E25	Epoch #10 Error: 0.181639438179597	Epoch #10 Error: 0.13165199575387062
12	Epoch #11 Error: 7.3831834050686576E201	Epoch #11 Error: 3.6694318512195587E27	Epoch #11 Error: 0.12901359259329215	Epoch #11 Error: 0.11928793241247616
13	Epoch #12 Error: 3.1777249592059526E220	Epoch #12 Error: 1.0860049053887298E30	Epoch #12 Error: 0.09472891817419975	Epoch #12 Error: 0.10870305632686955
14	Epoch #13 Error: 1.3714088889498186E239	Epoch #13 Error: 3.214127531165944E32	Epoch #13 Error: 0.09681686127647417	Epoch #13 Error: 0.09949787345928433
15	Epoch #14 Error: 5.918582136149191E257	Epoch #14 Error: 9.512483039273057E34	Epoch #14 Error: 0.09722562507980786	Epoch #14 Error: 0.09140239237430463
16	Epoch #15 Error: 2.5542791945403826E276	Epoch #15 Error: 2.815398839789481E37	Epoch #15 Error: 0.09138477124700316	Epoch #15 Error: 0.08421898838682065
17	Epoch #16 Error: 1.1023488489690084E295	Epoch #16 Error: 8.222111706748409E39	Epoch #16 Error: 0.08856266683420072	Epoch #16 Error: 0.07780051932339904
18	Epoch #17 Error: Infinity	Epoch #17 Error: 2.4659577612297126E42	Epoch #17 Error: 0.07857206178147903	Epoch #17 Error: 0.07203333507036881
19	Epoch #18 Error: Infinity	Epoch #18 Error: 7.298207004348898E44	Epoch #18 Error: 0.07174870161269693	Epoch #18 Error: 0.06682732719927382
20	Epoch #19 Error: Infinity	Epoch #19 Error: 2.159985028413813E47	Epoch #19 Error: 0.06432851902791936	Epoch #19 Error: 0.062109413416891666
21	Epoch #20 Error: Infinity	Epoch #20 Error: 6.392596037639489E49	Epoch #20 Error: 0.059421407163252555	Epoch #20 Error: 0.05781929088270778
22	Epoch #21 Error: Infinity	Epoch #21 Error: 1.8919419317204705E52	Epoch #21 Error: 0.05657859042667562	Epoch #21 Error: 0.05390649605113536
23	Epoch #22 Error: Infinity	Epoch #22 Error: 5.599359401779876E54	Epoch #22 Error: 0.053075807753832536	Epoch #22 Error: 0.05032326970968811
24	Epoch #23 Error: Infinity	Epoch #23 Error: 1.657176955764914E57	Epoch #23 Error: 0.05000037982411022	Epoch #23 Error: 0.04704802516340798
25	Epoch #24 Error: Infinity	Epoch #24 Error: 4.904553941907647E59	Epoch #24 Error: 0.04695238595582369	Epoch #24 Error: 0.044034254235854256
26	Epoch #25 Error: Infinity	Epoch #25 Error: 1.4515432088899526E62	Final error value for our ann: 0.04695238595582369	Epoch #25 Error: 0.04125985526477129
27	Epoch #26 Error: Infinity	Epoch #26 Error: 4.295962775981054E64		Epoch #26 Error: 0.038700037390397696
28	Epoch #27 Error: Infinity	Epoch #27 Error: 1.2714259807494321E67		Epoch #27 Error: 0.03633475479496485
29	Epoch #28 Error: Infinity	Epoch #28 Error: 3.762890603092293E69		Epoch #28 Error: 0.034145252863128424
30	Epoch #29 Error: Infinity	Epoch #29 Error: 1.1136587500857498E72		Epoch #29 Error: 0.03211522350055888
31	Epoch #30 Error: Infinity	Epoch #30 Error: 3.2959656360330866E74		Epoch #30 Error: 0.03023020638017321
32	Epoch #31 Error: Infinity	Epoch #31 Error: 9.754694254020326E76		Epoch #31 Error: 0.028477734105764142
33	Epoch #32 Error: Infinity	Epoch #32 Error: 2.886979889940104E79		Epoch #32 Error: 0.026845159512014303
34	Epoch #33 Error: Infinity	Epoch #33 Error: 8.544256926769386E81		Epoch #33 Error: 0.02532340831553801
35	Epoch #34 Error: NaN	Epoch #34 Error: 2.528743885660276E84		Epoch #34 Error: 0.023903883289392303
36	Final error value for our ann: NaN	Epoch #35 Error: 7.484027625041697E86		Epoch #35 Error: 0.022575374743088107
37		Epoch #36 Error: 2.2149601812985856E89		Epoch #36 Error: 0.021223408503240747
38		Epoch #37 Error: 6.555325875987647E91		Epoch #37 Error: 0.020170289039180705
39		Epoch #38 Error: 1.9401129118675777E94		Epoch #38 Error: 0.019079981109169694
40		Epoch #39 Error: 5.741925421041034E96		Epoch #39 Error: 0.018056931975466938
41		Epoch #40 Error: 1.6993705541117269E99		Epoch #40 Error: 0.017098217957288885
42		Epoch #41 Error: 5.029429403221283E101		Epoch #41 Error: 0.016193289532213152
43		Epoch #42 Error: 1.4885011289555889E104		Epoch #42 Error: 0.015344009825181266
44		Epoch #43 Error: 4.405342943880242E106		Epoch #43 Error: 0.01454461770821722
45		Epoch #44 Error: 1.3037979809866492E109		Epoch #44 Error: 0.013791664451429032
46		Epoch #45 Error: 3.858690049895642E111		Epoch #45 Error: 0.01308199351309657
47		Epoch #46 Error: 1.1420137805941725E114		Epoch #46 Error: 0.012412708730445284
48		Epoch #47 Error: 3.379884646415855E116		Epoch #47 Error: 0.01178114848676257
49		Epoch #48 Error: 1.0003049540431114E119		Epoch #48 Error: 0.011118486270771911
50		Epoch #49 Error: 2.960495654870719E121		Epoch #49 Error: 0.010621592737287015
51		Epoch #50 Error: 8.761803365334096E123		Epoch #50 Error: 0.010089253074317866
52		Epoch #51 Error: 2.593128532356709E126		Epoch #51 Error: 0.009585891539061598
53		Epoch #52 Error: 7.674579427252467E128		Final error value for our ann: 0.009585891539061598
54		Epoch #53 Error: 2.271355710889896E131		

Graphs 1 - number of epochs with different square errors

References

1. http://neuroph.sourceforge.net/
2. http://www.heatonresearch.com/aifh/vol3/
3. https://github.com/encog/encog-java-core/releases
4. http://www.cs.utsa.edu/~bylander/cs4793/learnsc32.pdf
5. http://neuralnetworksanddeeplearning.com
6. https://www.youtube.com/watch?v=IEFRtz68m-8