

```
1 ////////////////////////////////////////////////////////////////////
2 // Artificial Neural Network Test Program //
3 ////////////////////////////////////////////////////////////////////
4 //                               By Malcolm Stagg //
5 //                               //
6 // Supports Dynamic Rerouting of the //
7 // neural network, partially-connected //
8 // neural networks, and fully-connected //
9 // neural networks. //
10 //                               //
11 // Backpropagation learning is currently //
12 // supported. //
13 //                               //
14 // Training cycles may be repeated any //
15 // number of times, with results exported //
16 // to a comma-delimited file. //
17 ////////////////////////////////////////////////////////////////////
18
19 #include <stdio.h>
20 #include <stdlib.h>
21 #include <math.h>
22 #include <time.h>
23
24 #include "ocrsets.h"
25
26 /* [node][layer] */
27 double * fpinput;
28 double * fpoutput;
29 double * fpmean;
30 double * fpsd;
31
32 double * bpininput;
33 double * bpoutput;
34
35 /* [weight][layer] 2000, 2*/
36
37 double * weight;
38 double * learning_rate;
39 double * delta_w;
40
41 /* [weight][end 0=input 1=output][layer] 2000,2,2*/
42
43 unsigned int * weightconnection;
44
45 /* [layer] */
46
47 unsigned int neurons[3];
48 unsigned int synapses[2];
49
50 unsigned int monitor_synapses[100][2];
51
52 /* get time as seed for random generator */
53
54 time_t seconds;
55
56 unsigned int dataorder[50];
57
58 /* functions */
59
60 double get_fpinput(unsigned int node, unsigned int layer) {
61     return fpinput[layer*100+node];
62 }
```

```
63
64 void set_fpinput(unsigned int node, unsigned int layer, double value) {
65     fpinput[layer*100+node] = value;
66 }
67
68 double get_fpoutput(unsigned int node, unsigned int layer) {
69     return fpoutput[layer*100+node];
70 }
71 void set_fpoutput(unsigned int node, unsigned int layer, double value) {
72     fpoutput[layer*100+node] = value;
73 }
74
75 double get_fpmean(unsigned int node, unsigned int layer) {
76     return fpmean[layer*100+node];
77 }
78 void set_fpmean(unsigned int node, unsigned int layer, double value) {
79     fpmean[layer*100+node] = value;
80 }
81
82 double get_fpsd(unsigned int node, unsigned int layer) {
83     return fpsd[layer*100+node];
84 }
85 void set_fpsd(unsigned int node, unsigned int layer, double value) {
86     fpsd[layer*100+node] = value;
87 }
88
89 double get_bpinput(unsigned int node, unsigned int layer) {
90     return bpinput[layer*100+node];
91 }
92 void set_bpinput(unsigned int node, unsigned int layer, double value) {
93     bpinput[layer*100+node] = value;
94 }
95
96 double get_bpoutput(unsigned int node, unsigned int layer) {
97     return bpoutput[layer*100+node];
98 }
99 void set_bpoutput(unsigned int node, unsigned int layer, double value) {
100     bpoutput[layer*100+node] = value;
101 }
102
103 double get_weight(unsigned int node, unsigned int layer) {
104     return weight[layer*5000+node];
105 }
106 void set_weight(unsigned int node, unsigned int layer, double value) {
107     weight[layer*5000+node] = value;
108 }
109
110 double get_learning_rate(unsigned int node, unsigned int layer) {
111     return learning_rate[layer*5000+node];
112 }
113 void set_learning_rate(unsigned int node, unsigned int layer, double
value) {
114     learning_rate[layer*5000+node] = value;
115 }
116
117 double get_delta_w(unsigned int node, unsigned int layer) {
118     return delta_w[layer*5000+node];
119 }
120 void set_delta_w(unsigned int node, unsigned int layer, double value) {
121     delta_w[layer*5000+node] = value;
122 }
123
```

```
124 unsigned int get_weightconnection(unsigned int node, unsigned int which,
125 unsigned int layer) {
126     return weightconnection[(layer*10000)+(which*5000)+node];
127 }
128 void set_weightconnection(unsigned int node, unsigned int which,
129 unsigned int layer, unsigned int value) {
130     weightconnection[(layer*10000)+(which*5000)+node] = value;
131 }
132 void neuronfp(unsigned int node, unsigned int layer) {
133     double fpout, fpmn;
134     fpout = tanh(get_fpinput(node,layer));
135     set_fpoutput(node,layer,fpout);
136     /* also calculate mean (moving average), and a modification of
137 standard deviation */
138     fpmn = (get_fpmean(node,layer) * .9) + (fpout * .1);
139     set_fpmean(node,layer,fpmn);
140     set_fpsd(node,layer,(get_fpsd(node,layer) *.9) + ((fpout-fpmn) *
141 .1));
142 }
143 void neuronbp(unsigned int node, unsigned int layer) {
144     double sech_in;
145     sech_in = 1/cosh(get_fpinput(node,layer));
146     sech_in *= sech_in; /* sech^2 */
147     /* printf("\n sech^2(%f)= %f \n", get_fpinput(node,layer), sech_in);
148 */
149     //getch();
150     set_bpoutput(node,layer,sech_in * get_bpinput(node,layer));
151 }
152 void synapsefp(unsigned int which_weight, unsigned int layer) {
153     unsigned int which_node_in;
154     unsigned int which_node_out;
155     which_node_in = get_weightconnection(which_weight,0,layer);
156     which_node_out = get_weightconnection(which_weight,1,layer);
157     set_fpinput(which_node_out,layer+1,
158 get_fpinput(which_node_out,layer+1) +
159 get_fpoutput(which_node_in,layer)*get_weight(which_weight,layer));
160 }
161 void synapsebp(unsigned int which_weight, unsigned int layer) {
162     unsigned int which_node_in;
163     unsigned int which_node_out;
164     double bpin, bpout, fpout, w;
165     which_node_in = get_weightconnection(which_weight,0,layer);
166     which_node_out = get_weightconnection(which_weight,1,layer);
167 }
```

```
179     bpin = get_bpinput(which_node_in,layer);
180     bpout = get_bpoutput(which_node_out,layer+1);
181     fpout = get_fpoutput(which_node_in,layer);
182     w = get_weight(which_weight,layer);
183
184     set_bpinput(which_node_in,layer, bpin + bpout * w);
185
186     set_delta_w(which_weight,layer,
187     fpout*bpout*get_learning_rate(which_weight,layer));
188
189 }
190
191 void learn(unsigned int which_weight, unsigned int layer) {
192     set_weight(which_weight,layer, get_weight(which_weight,layer) +
193     get_delta_w(which_weight,layer));
194 }
195 void add_synapse(unsigned int layer, unsigned int from_node, unsigned
196 int to_node, float init_weight, float init_learning_rate) {
197     synapses[layer]++;
198     set_weightconnection(synapses[layer],0,layer, from_node);
199     set_weightconnection(synapses[layer],1,layer, to_node);
200     set_weight(synapses[layer],layer, init_weight);
201     set_learning_rate(synapses[layer],layer, init_learning_rate);
202 }
203 void remove_synapse(unsigned int which_weight, unsigned int layer) {
204     unsigned int n;
205
206     synapses[layer]--;
207     //printf("Removing Synapse...");
208     for (n=which_weight;n<=synapses[layer];n++) {
209         set_weightconnection(n,0,layer, get_weightconnection(n+1,0,layer));
210         set_weightconnection(n,1,layer, get_weightconnection(n+1,1,layer));
211         set_weight(n,layer, get_weight(n+1,layer));
212         set_learning_rate(n,layer, get_learning_rate(n+1,layer));
213     }
214     //printf(" Removed\n");
215
216 }
217
218 void forwards_propagate() {
219     unsigned int n;
220
221     /* input data should be in fpoutput[0] */
222
223     /* clear layer1 data */
224     for (n=0;n<neurons[1];n++) {
225         set_fpinput(n,1, 0);
226     }
227
228     /* clear layer2 data */
229     for (n=0;n<neurons[2];n++) {
230         set_fpinput(n,2, 0);
231     }
232
233     /* propagate through layer0 synapses (input) */
234     for (n=0;n<synapses[0];n++) {
235         synapsefp(n,0);
236     }
237
```

```
238     /* propagate through layer1 neurons (hidden) */
239     for (n=0;n<neurons[1];n++) {
240         neuronfp(n,1);
241     }
242
243     /* propagate through layer1 synapses (hidden) */
244     for (n=0;n<synapses[1];n++) {
245         synapsefp(n,1);
246     }
247
248     /* propagate through layer2 neurons (output) */
249     for (n=0;n<neurons[2];n++) {
250         neuronfp(n,2);
251     }
252
253     /* output data is now available in fpoutput[2] */
254 }
255
256
257 void backwards_propagate() {
258     unsigned int n;
259
260     /* input data should be in bpinput[2] */
261
262     /* clear layer0 data */
263     for (n=0;n<neurons[0];n++) {
264         set_bpinput(n,0, 0);
265     }
266
267     /* clear layer1 data */
268     for (n=0;n<neurons[1];n++) {
269         set_bpinput(n,1, 0);
270     }
271
272     /* propagate through layer2 neurons (output) */
273     for (n=0;n<neurons[2];n++) {
274         neuronbp(n,2);
275     }
276
277     /* propagate through layer1 synapses (hidden) */
278     for (n=0;n<synapses[1];n++) {
279         synapsebp(n,1);
280     }
281
282     /* propagate through layer1 neurons (hidden) */
283     for (n=0;n<neurons[1];n++) {
284         neuronbp(n,1);
285     }
286
287     /* propagate through layer0 synapses (input) */
288     for (n=0;n<synapses[0];n++) {
289         synapsebp(n,0);
290     }
291
292     /* output data is now available in bpinput[0] */
293
294     /* weight updates may now be computed */
295
296     /* teach layer0 synapses (input) */
297     for (n=0;n<synapses[0];n++) {
298         learn(n,0);
299     }
```

```
300
301 /* teach layer1 synapses (hidden) */
302 for (n=0;n<synapses[1];n++) {
303     learn(n,1);
304 }
305
306 }
307
308 void orderdata(unsigned int datalength) {
309     short used[50];
310     unsigned int n, r;
311
312     for (n=0;n<datalength;n++) {
313         used[n] = 0;
314     }
315
316     for (n=0;n<datalength;n++) {
317         do {
318             r = ((unsigned int) rand()) % datalength;
319             } while (used[r] == 1);
320         used[r] = 1;
321         dataorder[n] = r;
322     }
323 }
324
325 void selectsynapses() {
326     //monitor 100 of the synapses in layer 0 and 10 in layer 1
327     unsigned int n, m;
328     for (n=0;n<100;n++) {
329         monitor_synapses[n][0] = rand() % synapses[0];
330         for (m=0;m<n;m++) {
331             while (monitor_synapses[n][0] == monitor_synapses[m][0]) {
332                 monitor_synapses[n][0] = rand() % synapses[0];
333                 m=0;
334             }
335         }
336     }
337     for (n=0;n<10;n++) {
338         monitor_synapses[n][1] = rand() % synapses[1];
339         for (m=0;m<n;m++) {
340             while (monitor_synapses[n][1] == monitor_synapses[m][1]) {
341                 monitor_synapses[n][1] = rand() % synapses[1];
342                 m=0;
343             }
344         }
345     }
346 }
347
348 void monitorsynapses() {
349     unsigned int n;
350     double w;
351     const double threshold = 0.001;
352     for (n=0;n<100;n++) {
353         w = get_weight(monitor_synapses[n][0],0);
354         if (w<threshold && w>-threshold) {
355             //w contributes very little, it can probably be removed
356             remove_synapse(n, 0);
357         }
358     }
359     for (n=0;n<10;n++) {
360         w = get_weight(monitor_synapses[n][1],0);
361         if (w<threshold && w>-threshold) {
```

```
362         //w contributes very little, it can probably be removed
363         remove_synapse(n, 1);
364     }
365 }
366 }
367
368 void main() {
369     long n, m;
370     long l1, l2;
371
372     long cycle, dataset;
373
374     unsigned int opt, trial;
375
376     double mse; //mean square error
377
378     short fullyconnected = 1;
379     short speciallearning = 1;
380
381     unsigned int connect_to[50];
382
383     FILE * pFile;
384
385     char fileout[100];
386
387     clrscr();
388     printf("Neural Network Simulation\n");
389     printf("By Malcolm Stagg\n");
390
391     /* create random seed */
392     time(&seconds);
393
394     srand((unsigned int) seconds);
395
396     pFile = fopen ("c:\\nn_out0.txt","wt");
397     fputs
398     ("CYCLE,MSE,HIDDEN_SYNAPSES,OUTPUT_SYNAPSES,TRIAL,METHOD\n",pFile);
399
400
401     opt=0;
402
403     //for (opt=0;opt<3;opt++) {
404     for (trial=0;trial<10;trial++) {
405         printf("\n\n Learning Method %u, Trial %u \n\n", opt, trial);
406         if (opt == 0) {
407             fullyconnected=1;
408             speciallearning=0;
409         } else if (opt == 1) {
410             fullyconnected=0;
411             speciallearning=0;
412
413         } else {
414             fullyconnected=1;
415             speciallearning=1;
416
417         }
418
419
420     printf("Defining Network...\n");
421
422     /* define network layout */
```

```
423  /* OCR: 100 inputs, 50 hidden, 10 outputs */
424  neurons[0] = 100;
425  neurons[1] = 50;
426  neurons[2] = 10;
427
428  /* start off with a fully connected network */
429  if (fullyconnected) {
430      synapses[0]=5000;
431      synapses[1]=500;
432  } else {
433      //lets make a 20% partially connected, randomly
434      synapses[0]=1000;
435      synapses[1]=100;
436  }
437
438  /* allocate memory */
439
440  if ((fpinput = (double *)malloc(300L * sizeof(double))) == NULL)
441      exit(EXIT_FAILURE);
442  if ((fpoutput = (double *)malloc(300L * sizeof(double))) == NULL)
443      exit(EXIT_FAILURE);
444  if ((bpinput = (double *)malloc(300L * sizeof(double))) == NULL)
445      exit(EXIT_FAILURE);
446  if ((bpoutput = (double *)malloc(300L * sizeof(double))) == NULL)
447      exit(EXIT_FAILURE);
448  if ((fpmean = (double *)malloc(300L * sizeof(double))) == NULL)
449      exit(EXIT_FAILURE);
450  if ((fpsd = (double *)malloc(300L * sizeof(double))) == NULL)
451      exit(EXIT_FAILURE);
452
453  if ((weight = (double *)malloc(10000L * sizeof(double))) == NULL)
454      exit(EXIT_FAILURE);
455  if ((learning_rate = (double *)malloc(10000L * sizeof(double))) ==
456      NULL) exit(EXIT_FAILURE);
457  if ((delta_w = (double *)malloc(10000L * sizeof(double))) == NULL)
458      exit(EXIT_FAILURE);
459  if ((weightconnection = (unsigned int *)malloc(20000L *
460      sizeof(unsigned int))) == NULL) exit(EXIT_FAILURE);
461
462  printf("Connecting Synapses...");
463
464  /* connect synapses */
465
466  n=0;
467
468  if (fullyconnected) {
469
470      for (l1=0;l1<neurons[0];l1++) {
471          for (l2=0;l2<neurons[1];l2++) {
472              set_weightconnection(n,0,0, l1);
473              set_weightconnection(n,1,0, l2);
474              /* randomize weight in range of +/- .25*/
475              set_weight(n,0, .5*(((double)rand()) / (RAND_MAX))-.25);
476              set_learning_rate(n,0, 0.01);
477              n++;
478          }
479      }
480
481  }
482
483  n=0;
484
485  for (l1=0;l1<neurons[1];l1++) {
486      for (l2=0;l2<neurons[2];l2++) {
```

```
475     set_weightconnection(n,0,1, l1);
476     set_weightconnection(n,1,1, l2);
477     /* randomize weight in range of +/- .25*/
478     set_weight(n,1, .5*(((double)rand()) / (RAND_MAX))-.25);
479     set_learning_rate(n,1, 0.01);
480     n++;
481 }
482 }
483
484 } else {
485
486     for (l1=0;l1<neurons[0];l1++) {
487         //pick 10 neurons to connect to, must not be the same
488         for (n=0;n<10;n++) {
489             l2 = rand() % neurons[1];
490             for (m=0;m<n;m++) {
491                 while (connect_to[m] == l2) {
492                     l2 = rand() % neurons[1];
493                     m=0;
494                 }
495             }
496             set_weightconnection(l1*10+n,0,0, l1);
497             set_weightconnection(l1*10+n,1,0, l2);
498             /* randomize weight in range of +/- .25*/
499             set_weight(l1*10+n,0, .5*(((double)rand()) / (RAND_MAX))-.25);
500             set_learning_rate(l1*10+n,0, 0.01);
501         }
502     }
503
504     for (l1=0;l1<neurons[1];l1++) {
505         //pick 2 neurons to connect to, must not be the same
506         for (n=0;n<2;n++) {
507             l2 = rand() % neurons[2];
508             for (m=0;m<n;m++) {
509                 while (connect_to[m] == l2) {
510                     l2 = rand() % neurons[2];
511                     m=0;
512                 }
513             }
514             set_weightconnection(l1*2+n,0,1, l1);
515             set_weightconnection(l1*2+n,1,1, l2);
516             /* randomize weight in range of +/- .25*/
517             set_weight(l1*2+n,1, .5*(((double)rand()) / (RAND_MAX))-.25);
518             set_learning_rate(l1*2+n,1, 0.01);
519         }
520     }
521 }
522
523 for (cycle=0;cycle<1000;cycle++) {
524
525     /* randomly order the training data */
526
527     orderdata(50);
528
529     mse = 0;
530
531     for (dataset=0;dataset<50;dataset++) {
532
533         //printf("Presenting Training Set %u...\n", dataorder[dataset]);
534
535         /* now try it out - present input data*/
536         for (n=0;n<100;n++) {
```

```
537     set_fpoutput(n,0, (double) in_test_set[dataorder[dataset]][n]);
538     }
539
540     forwards_propagate();
541
542     /* and present feedback */
543
544     for (n=0;n<10;n++) {
545         set_bpinput(n,2, (double) out_test_set[dataorder[dataset]][n] -
546         get_fpoutput(n,2));
547         //printf("%1.3f ", (double) out_test_set[dataorder[dataset]][n] -
548         get_fpoutput(n,2));
549         mse += ((double) out_test_set[dataorder[dataset]][n] -
550         get_fpoutput(n,2))*((double) out_test_set[dataorder[dataset]][n] -
551         get_fpoutput(n,2));
552     }
553
554     /* printf("\nTraining...\n"); */
555
556     backwards_propagate();
557
558     //printf("Mean Square Error: %1.3f Synapses %u %u \n", mse,
559     synapses[0], synapses[1]);
560     //printf(".");
561
562     sprintf(fileout,
563     "%u,%f,%u,%u,%u,%u\n", cycle,mse, synapses[0], synapses[1], trial, opt);
564
565     fputs (fileout, pFile);
566
567     //if ((cycle%20 == 19) && speciallearning) {
568     if (speciallearning) {
569         selectsynapses();
570         monitorsynapses();
571     }
572     //}
573
574     }
575 }//}
576
577 fclose(pFile);
578
579 /* de-allocate memory */
580
581 free(fpinput);
582 free(fpoutput);
583 free(bpinput);
584 free(bpoutput);
585 free(fpmean);
586 free(fpsd);
587
588 free(weight);
589 free(learning_rate);
590 free(delta_w);
591
592 free(weightconnection);
593 }
```