http://wenku.baidu.com/link?url=lhTDjLUeibwAEC3c06hb5dEOk7qstXz0o8MPZjDaAda2n0bN3cP2AyXKGMtzSXHFm-vb5ljZWg7YdZY5X8srP7XDDhQ-twPILFWwp-OBUHu

BP神经网络C程序

在该题的程序设计中采用了文件相关的操作，记录了相关学习和测试信息数据。权值用伪随机数函数随机产生（范围是（0,0.5）） 采用结构体及链表来实现神经网络的结构 分为实例结构体、层结构体和网络结构体 数据结构的设计参照了《人工神经网络原理》（马锐编著，北京：机械工业出版社，2010,7）一书

学习算法的优化也参照该书

采用学习效率自适应调整算法优化源程序的学习算法，以减少学习次数

由于能力和知识有限，该程序存在较大漏洞误差，在调整学习率时，不好掌握调节系数 初始权值的限定范围适中，则程序的学习次数将明显减少

在随机赋初始权值（0,0.5）时，学习次数可调节至135，但对测试数据的判别效果不理想，没有采用

#include<stdio.h> #include<stdlib.h> #include<math.h> #include<malloc.h>

#define TRUE 1 #define FALSE 0

#define NUM\_LAYERS 4

#define NUM 20 //训练实例个数 #define N 2 //输入层单元数 #define M 2 //输出层单元数

int Units[NUM\_LAYERS] = {N,3,3,M}; //每层单元数 FILE \*fp,\*fb;

typedef struct //训练实例 { float x[N]; float y[M]; }TRAIN;

typedef struct //网络层结构 { int Units; //该层中单元的个数 float \*Output; //第 i 个单元的输出 float \*Error; //第 i 个单元的校正误差 float \*\*Weight; //第 i 个单元的连接权值

}LAYER;

typedef struct //网络 { LAYER \*\*Layer; //隐层定义 LAYER \*Inputlayer; //输入层 LAYER \*Outputlayer; //输出层 float Error; //允许误差 float Eta; //学习率 }NET;

//初始化伪随机数发生器 void InitializeRandoms() { srand(4711); return; }

//产生随机实数并规范化

float RandomReal() //产生(-0.5,0.5)之间的随机数 { return (float)((rand()%100)/200.0); }

//初始化训练数据

void InitializeTrainingData(TRAIN \*training) { int i,j; char filename[20]; printf("\n请输入训练实例的数据文件名: \n"); gets(filename); fb = fopen(filename,"r"); fprintf(fp,"\n\n--Saving initialization training datas ...\n"); for(i=0;i<NUM;i++) { for(j=0;j<N;j++) { fscanf(fb,"%f",&(training+i)->x[j]); fprintf(fp,"%10.4f",(training+i)->x[j]); } for(j=0;j<M;j++) { fscanf(fb,"%f",&(training+i)->y[j]); fprintf(fp,"%10.4f",(training+i)->y[j]);

}LAYER;

typedef struct //网络 { LAYER \*\*Layer; //隐层定义 LAYER \*Inputlayer; //输入层 LAYER \*Outputlayer; //输出层 float Error; //允许误差 float Eta; //学习率 }NET;

//初始化伪随机数发生器 void InitializeRandoms() { srand(4711); return; }

//产生随机实数并规范化

float RandomReal() //产生(-0.5,0.5)之间的随机数 { return (float)((rand()%100)/200.0); }

//初始化训练数据

void InitializeTrainingData(TRAIN \*training) { int i,j; char filename[20]; printf("\n请输入训练实例的数据文件名: \n"); gets(filename); fb = fopen(filename,"r"); fprintf(fp,"\n\n--Saving initialization training datas ...\n"); for(i=0;i<NUM;i++) { for(j=0;j<N;j++) { fscanf(fb,"%f",&(training+i)->x[j]); fprintf(fp,"%10.4f",(training+i)->x[j]); } for(j=0;j<M;j++) { fscanf(fb,"%f",&(training+i)->y[j]); fprintf(fp,"%10.4f",(training+i)->y[j]);

} fprintf(fp,"\n"); } fclose(fb); return; }

//应用程序初始化

void InitializeApplication(NET \*Net) { Net->Eta = (float)0.3; Net->Error = (float)0.0001; fp = fopen("BPResultData.txt","w+"); return; }

//应用程序关闭时终止打开的文件 void FinalizeApplication(NET \*Net) { fclose(fp); return; }

//分配内存,建立网络

void GenerateNetwork(NET \*Net) { int l,i; Net->Layer = (LAYER \*\*)calloc(NUM\_LAYERS,sizeof(LAYER \*)); for(l=0;l<NUM\_LAYERS;l++) { Net->Layer[l] = (LAYER \*)malloc(sizeof(LAYER)); Net->Layer[l]->Units = Units[l]; Net->Layer[l]->Output = (float \*) calloc(Units[l]+1,sizeof(float)); Net->Layer[l]->Error = (float \*) calloc(Units[l]+1,sizeof(float)); Net->Layer[l]->Weight = (float \*\*)calloc(Units[l]+1,sizeof(float \*)); Net->Layer[l]->Output[0] = 1; if(l != 0) for(i=1;i <= Units[l];i++) //下标从"1"开始 Net->Layer[l]->Weight[i] = (float \*)calloc(Units[l-1]+1,sizeof(float)); } Net->Inputlayer = Net->Layer[0]; Net->Outputlayer = Net->Layer[NUM\_LAYERS - 1]; return; }

//产生随机实数作为初始连接权值 void RandomWeights(NET \*Net) { int l,i,j; for(l=1;l<NUM\_LAYERS;l++) for(i=1;i <= Net->Layer[l]->Units;i++) for(j=0;j <= Net->Layer[l-1]->Units;j++) Net->Layer[l]->Weight[i][j] = RandomReal(); return; }

//设置输入层的输出值

void SetInput(NET \*Net,float \*Input) { int i; for(i=1;i <= Net->Inputlayer->Units;i++) Net->Inputlayer->Output[i] = Input[i-1]; //输入层采用 u(x) = x return; }

//设置输出层的输出值

void GetOutput(NET \*Net,float \*Output) { int i; for(i=1;i <= Net->Outputlayer->Units;i++) Output[i-1] = (float)(1/(1 + exp(-Net->Outputlayer->Output[i]))); //输出层采用 f(x)=1/(1+e^(-x)) return; }

//层间顺传播

void PropagateLayer(NET \*Net,LAYER \*Lower,LAYER \*Upper) { int i,j; float sum; for(i=1;i <= Upper->Units;i++) { sum = 0; for(j=1;j <= Lower->Units;j++) sum += (Upper->Weight[i][j] \* Lower->Output[j]); Upper->Output[i] = (float)(1/(1 + exp(-sum))); }

return; }

//整个网络所有层间的顺传播 void PropagateNet(NET \*Net) { int l; for(l=0;l < NUM\_LAYERS-1;l++) PropagateLayer(Net,Net->Layer[l],Net->Layer[l+1]); return; }

//计算输出层误差

void ComputeOutputError(NET \*Net,float \*target) { int i; float Out,Err; for(i=1;i <= Net->Outputlayer->Units;i++) { Out = Net->Outputlayer->Output[i]; Err = target[i-1] - Out; Net->Outputlayer->Error[i] = Out\*(1-Out)\*Err; } return; }

//层间逆传播

void BackpropagateLayer(NET \*Net,LAYER \*Upper,LAYER \*Lower) { int i,j; float Out,Err; for(i=1;i <= Lower->Units;i++) { Out = Lower->Output[i]; Err = 0; for(j=1;j <= Upper->Units;j++) Err += (Upper->Weight[j][i] \* Upper->Error[j]); Lower->Error[i] = Out\*(1-Out)\*Err; } return; }

//整个网络所有层间的逆传播 void BackpropagateNet(NET \*Net)

{ int l; for(l=NUM\_LAYERS-1;l>1;l--) BackpropagateLayer(Net,Net->Layer[l],Net->Layer[l-1]); return; }

//权值调整

void AdjustWeights(NET \*Net) { int l,i,j; float Out,Err; for(l=1;l<NUM\_LAYERS;l++) for(i=1;i <= Net->Layer[l]->Units;i++) for(j=0;j <= Net->Layer[l-1]->Units;j++) { Out = Net->Layer[l-1]->Output[j]; Err = Net->Layer[l]->Error[i]; Net->Layer[l]->Weight[i][j] += (Net->Eta\*Err\*Out); } return; }

//网络处理过程

void SimulateNet(NET \*Net,float \*Input,float \*Output,float \*target,int TrainOrNot) { SetInput(Net,Input); //输入数据 PropagateNet(Net); //模式顺传播 GetOutput(Net,Output); //形成输出 ComputeOutputError(Net,target); //计算输出误差 if(TrainOrNot) { BackpropagateNet(Net); //误差逆传播 AdjustWeights(Net); //调整权值 } return; }

//训练过程

void TrainNet(NET \*Net,TRAIN \*training) { int l,i,j,k; int count=0,flag=0; float Output[M],outputfront[M],ERR,err,sum;

do { flag = 0; sum = 0; ERR = 0; if(count >= 1) for(j=0;j<M;j++)

outputfront[j]=Output[j];

SimulateNet(Net,(training+(count%NUM))->x,Output,(training+(count%NUM))->y,TRUE); if(count >= 1) { k = count%NUM; for(i=1;i <= Net->Outputlayer->Units;i++) { sum += Net->Outputlayer->Error[i]; err = (training+k-1)->y[i-1] - outputfront[i-1]; ERR += (outputfront[i-1] \* (1 - outputfront[i-1]) \* err); } if(sum <= ERR) Net->Eta = (float)(0.9999 \* Net->Eta); else Net->Eta = (float)(1.0015 \* Net->Eta); } if(count >= NUM) { for(k=1;k <= M;k++) if(Net->Outputlayer->Error[k] > Net->Error) { flag=1; break; } if(k>M) flag=0; } count++;

}while(flag || count <= NUM); fprintf(fp,"\n\n\n");

fprintf(fp,"--training results ... \n");

fprintf(fp,"training times: %d\n",count); fprintf(fp,"\n\*\*\*\*\*the final weights\*\*\*\*\*\n"); for(l=1;l<NUM\_LAYERS;l++) { for(i=1;i <= Net->Layer[l]->Units;i++) {

for(j=1;j <= Net->Layer[l-1]->Units;j++)

fprintf(fp,"%15.6f",Net->Layer[l]->Weight[i][j]); fprintf(fp,"\n"); } fprintf(fp,"\n\n"); } }

//评估过程

void EvaluateNet(NET \*Net) { int i; printf("\n\n("); fprintf(fp,"\n\n("); for(i=1;i <= Net->Inputlayer->Units;i++) { printf(" %.4f",Net->Inputlayer->Output[i]); fprintf(fp,"%10.4f",Net->Inputlayer->Output[i]); } printf(")\t"); fprintf(fp,")\t"); for(i=1;i <= Net->Outputlayer->Units;i++) { if(fabs(Net->Outputlayer->Output[i] - 1.0) <= 0.0499) { printf("肯定是第 %d 类, ",i); fprintf(fp,"肯定是第 %d 类, ",i); } if(fabs(Net->Outputlayer->Output[i] - 0.9) <= 0.0499) { printf("几乎是第 %d 类, ",i); fprintf(fp,"几乎是第 %d 类, ",i); } if(fabs(Net->Outputlayer->Output[i] - 0.8) <= 0.0499) { printf("极是第 %d 类, ",i); fprintf(fp,"极是第 %d 类, ",i); } if(fabs(Net->Outputlayer->Output[i] - 0.7) <= 0.0499) { printf("很是第 %d 类, ",i); fprintf(fp,"很是第 %d 类, ",i); } if(fabs(Net->Outputlayer->Output[i] - 0.6) <= 0.0499) {

printf("相当是第 %d 类, ",i); fprintf(fp,"相当是第 %d 类, ",i); } if(fabs(Net->Outputlayer->Output[i] - 0.5) <= 0.0499) { printf("差不多是第 %d 类, ",i); fprintf(fp,"差不多是第 %d 类, ",i); } if(fabs(Net->Outputlayer->Output[i] - 0.4) <= 0.0499) { printf("比较像是第 %d 类, ",i); fprintf(fp,"比较像是第 %d 类, ",i); } if(fabs(Net->Outputlayer->Output[i] - 0.3) <= 0.0499) { printf("有些像是第 %d 类, ",i); fprintf(fp,"有些像是第 %d 类, ",i); } if(fabs(Net->Outputlayer->Output[i] - 0.2) <= 0.0499) { printf("有点像是第 %d 类, ",i); fprintf(fp,"有点像是第 %d 类, ",i); } if(fabs(Net->Outputlayer->Output[i] - 0.1) <= 0.0499) { printf("稍稍像是第 %d 类, ",i); fprintf(fp,"稍稍像是第 %d 类, ",i); } if(Net->Outputlayer->Output[i] <= 0.0499) { printf("肯定不是第 %d 类, ",i); fprintf(fp,"肯定不是第 %d 类, ",i); } } printf("\n\n"); fprintf(fp,"\n\n\n"); return; }

//测试过程

void TestNet(NET \*Net) { TRAIN Testdata; float Output[M];

int i,j,flag=0; char select; fprintf(fp,"\n\n--Saving test datas ...\n"); do { printf("\n请输入测试数据(x1 x2 ... y1 y2 ...): \n"); for(j=0;j<N;j++) { scanf("%f",&Testdata.x[j]); fprintf(fp,"%10.4f",Testdata.x[j]); } for(j=0;j<M;j++) { scanf("%f",&Testdata.y[j]); fprintf(fp,"%10.4f",Testdata.y[j]); } fprintf(fp,"\n"); SimulateNet(Net,Testdata.x,Output,Testdata.y,FALSE); fprintf(fp,"\n--NET Output and Error of the Test Data ....\n"); for(i=1;i <= Net->Outputlayer->Units;i++) fprintf(fp,"%10.6f %10.6f\n",Net->Outputlayer->Output[i],Net->Outputlayer->Error[i]); EvaluateNet(Net); printf("\n继续测试?(y/n):\n"); getchar(); scanf("%c",&select); printf("\n"); if((select == 'y')||(select == 'Y')) flag = 1; else flag=0; }while(flag); return; }

//主函数 void main() { TRAIN TrainingData[NUM]; NET Net; InitializeRandoms(); //初始化伪随机数发生器 GenerateNetwork(&Net); //建立网络 RandomWeights(&Net); //形成初始权值 InitializeApplication(&Net); //应用程序初始化,准备运行

InitializeTrainingData(TrainingData); //记录训练数据 TrainNet(&Net,TrainingData); //开始训练 TestNet(&Net);

FinalizeApplication(&Net); //程序关闭,完成善后工作 return;

}