/\*NEURAL NETWORK:

CODED BY JUNSATO SINCE 2019.8.14.\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <malloc.h>

#include <math.h>

#include <dos.h>

#include <limits.h>

#include <windows.h>

#define PI 3.1415926535897932384

#define SIGMOID 101

#define SOFTMAX 102

#define RESIDUALSUMSQUARE 201

#define CROSSENTROPY 202

double \*mallocdoublevector(int vsize);

double \*\*mallocdoublematrix(int nline,

int nrow);

void freematrix(double \*\*mtx,int nline);

main(void)

{

HANDLE hndc;

HWND hwndc;

HDC hdcc;

CONSOLE\_SCREEN\_BUFFER\_INFO cb;

FILE \*fin, \*fout;

char non[10],str[256],wname[256];

int i,j,k,ii,jj,kk,m,n,ni,no,iflag,iactivate,iloss;

int ncategory,nrefer,nlayer,nstep,mlabel;

int \*ic,\*nnode;

double dv,df,f1,f2,fa,c1,c2,c3,alpha,beta,

gamma,vsize,eps;

/\*UPDATING RATE = LEARNING RATE, LOSS FUNCTION\*/

double urate,floss,ploss,mloss,vsum;

/\*VALUE, WEIGHT, BIAS\*/

double \*\*v,\*\*\*w,\*\*b;

double \*\*\*wi,\*\*bi,\*\*\*wf,\*\*bf;

/\*REFERENCES, SPECIMEN,

CATEGORY\*/

double \*\*r,\*s,\*\*tc;

/\*PARTIAL DERIVATIVE\*/

double \*\*dldb,\*\*\*dldw,\*\*dldv;

fout=fopen("cnntest.txt","w");

/\*CONDITIONS\*/

eps=0.001;

urate=0.1;

iactivate=SOFTMAX;

iloss=CROSSENTROPY;

ncategory=4;

nrefer=4;

nlayer=3;

nnode=(int \*)malloc(nlayer\*sizeof(int));

\*(nnode+0)=3; /\*=INPUT\*/

\*(nnode+1)=9;

\*(nnode+2)=4; /\*=OUTPUT\*/

ni=\*(nnode+0);

no=\*(nnode+nlayer-1);

/\*CATEGORIES\*/

tc=(double \*\*)malloc(ncategory

\*sizeof(double \*));

for(ii=0; ii<ncategory; ii++)

\*(tc+ii)=mallocdoublevector(no);

/\*CASE 003 SUCCESSFUL : 4 COLORS\*/

/\*CYAN\*/

\*(\*(tc+0)+0)=1.0; \*(\*(tc+0)+1)=0.0;

\*(\*(tc+0)+2)=0.0; \*(\*(tc+0)+3)=0.0;

/\*YELLOW\*/

\*(\*(tc+1)+0)=0.0; \*(\*(tc+1)+1)=1.0;

\*(\*(tc+1)+2)=0.0; \*(\*(tc+1)+3)=0.0;

/\*MAGENTA\*/

\*(\*(tc+2)+0)=0.0; \*(\*(tc+2)+1)=0.0;

\*(\*(tc+2)+2)=1.0; \*(\*(tc+2)+3)=0.0;

/\*ORANGE\*/

\*(\*(tc+3)+0)=0.0; \*(\*(tc+3)+1)=0.0;

\*(\*(tc+3)+2)=0.0; \*(\*(tc+3)+3)=1.0;

/\*REFERENCES\*/

ic=(int \*)malloc(nrefer\*sizeof(int));

r=(double \*\*)malloc(nrefer\*sizeof(double \*));

for(ii=0; ii<nrefer; ii++)

\*(r+ii)=mallocdoublevector(ni);

/\*REFERENCE LIST RGB\*/

\*(ic+ 0)=0; \*(\*(r+ 0)+0)= 0;

\*(\*(r+ 0)+1)=255;

\*(\*(r+ 0)+2)=255;

\*(ic+ 1)=1; \*(\*(r+ 1)+0)=255;

\*(\*(r+ 1)+1)=255;

\*(\*(r+ 1)+2)= 0;

\*(ic+ 2)=2; \*(\*(r+ 2)+0)=255;

\*(\*(r+ 2)+1)= 0;

\*(\*(r+ 2)+2)=255;

\*(ic+ 3)=3; \*(\*(r+ 3)+0)=255;

\*(\*(r+ 3)+1)=100;

\*(\*(r+ 3)+2)= 0;

hndc=GetStdHandle(STD\_ERROR\_HANDLE);

GetConsoleScreenBufferInfo(hndc,&cb);

SetConsoleTextAttribute(hndc,FOREGROUND\_INTENSITY | FOREGROUND\_RED | FOREGROUND\_BLUE);

fprintf(stderr,"BACK PROPAGATION TEST\n");

SetConsoleTextAttribute(hndc,cb.wAttributes);

for(ii=0; ii<nrefer; ii++)

{

fprintf(stderr,"REFERENCE : V%d =",ii+1);

for(jj=0; jj<ni; jj++) fprintf(stderr," %8.3f",\*(\*(r+ii)+jj));

fprintf(stderr,"\n");

fprintf(stderr,"CATEGORY : T%d =",\*(ic+ii));

for(jj=0; jj<no; jj++) fprintf(stderr," %8.3f",\*(\*(tc+(\*(ic+ii)))+jj));

fprintf(stderr,"\n");

}

gets(non);

/\*SPECIMEN\*/

s=(double \*)malloc(ni\*sizeof(double));

\*(s+0)=140; \*(s+1)=180; \*(s+2)=40; /\*RGB : CASE 003\*/

/\*PARAMETERS\*/

v=(double \*\*)malloc(nlayer\*sizeof(double \*));

dldv=(double \*\*)malloc(nlayer\*sizeof(double \*));

for(ii=0; ii<nlayer; ii++)

{

\*(v+ii)=mallocdoublevector(\*(nnode+ii));

\*(dldv+ii)=mallocdoublevector(\*(nnode+ii));

}

b=(double \*\*)malloc(nlayer\*sizeof(double \*));

bi=(double \*\*)malloc(nlayer\*sizeof(double \*));

bf=(double \*\*)malloc(nlayer\*sizeof(double \*));

dldb=(double \*\*)malloc(nlayer\*sizeof(double \*));

for(ii=0; ii<(nlayer-1); ii++)

{

\*(b+ii)=mallocdoublevector(\*(nnode+ii+1));

\*(bi+ii)=mallocdoublevector(\*(nnode+ii+1));

\*(bf+ii)=mallocdoublevector(\*(nnode+ii+1));

\*(dldb+ii)=mallocdoublevector(\*(nnode+ii+1));

}

w=(double \*\*\*)malloc((nlayer-1)\*sizeof(double \*\*));

wi=(double \*\*\*)malloc((nlayer-1)\*sizeof(double \*\*));

wf=(double \*\*\*)malloc((nlayer-1)\*sizeof(double \*\*));

dldw=(double \*\*\*)malloc((nlayer-1)\*sizeof(double \*\*));

for(ii=0; ii<(nlayer-1); ii++)

{

\*(w+ii)=(double \*\*)malloc((\*(nnode+ii))\*sizeof(double \*));

\*(wi+ii)=(double \*\*)malloc((\*(nnode+ii))\*sizeof(double \*));

\*(wf+ii)=(double \*\*)malloc((\*(nnode+ii))\*sizeof(double \*));

\*(dldw+ii)=(double \*\*)malloc((\*(nnode+ii))\*sizeof(double \*));

for(jj=0; jj<(\*(nnode+ii)); jj++)

{

\*(\*(w+ii)+jj)=mallocdoublevector(\*(nnode+ii+1));

\*(\*(wi+ii)+jj)=mallocdoublevector(\*(nnode+ii+1));

\*(\*(wf+ii)+jj)=mallocdoublevector(\*(nnode+ii+1));

\*(\*(dldw+ii)+jj)=mallocdoublevector(\*(nnode+ii+1));

}

}

/\*INITIAL WEIGHT, BIAS : RANDOM\*/

iflag=0;

for(ii=0; ii<(nlayer-1); ii++)

{

for(jj=0; jj<(\*(nnode+ii+1)); jj++)

{

\*(\*(bf+ii)+jj)=1.0

\*(double)(rand()%100+1)/1000.0;

iflag++;

}

}

iflag=0;

for(ii=0; ii<(nlayer-1); ii++)

{

for(jj=0; jj<(\*(nnode+ii)); jj++)

{

for(kk=0; kk<(\*(nnode+ii+1)); kk++)

{

\*(\*(\*(wf+ii)+jj)+kk)=1.0

\*(double)(rand()%100+1)/10000.0;

iflag++;

}

}

}

/\*GRADIENT DESCENT\*/

nstep=0;

iflag=0;

while(1)

{

nstep++;

/\*STOCK INITIAL WEIGHT, BIAS\*/

for(ii=0; ii<(nlayer-1); ii++)

{

for(jj=0; jj<(\*(nnode+ii+1)); jj++)

\*(\*(bi+ii)+jj)=\*(\*(bf+ii)+jj);

}

for(ii=0; ii<(nlayer-1); ii++)

{

for(jj=0; jj<(\*(nnode+ii)); jj++)

{

for(kk=0; kk<(\*(nnode+ii+1)); kk++)

\*(\*(\*(wi+ii)+jj)+kk)

=\*(\*(\*(wf+ii)+jj)+kk);

}

}

floss=0.0;

if(nstep==1) ploss=0.0;

for(m=0; m<nrefer; m++)

{

for(i=0; i<ni; i++)

\*(\*(v+0)+i)=\*(\*(r+m)+i);

/\*INITIAL WEIGHT, BIAS\*/

for(ii=0; ii<(nlayer-1); ii++)

{

for(jj=0; jj<(\*(nnode+ii+1)); jj++)

\*(\*(b+ii)+jj)=\*(\*(bi+ii)+jj);

}

for(ii=0; ii<(nlayer-1); ii++)

{

for(jj=0; jj<(\*(nnode+ii)); jj++)

{

for(kk=0; kk<(\*(nnode+ii+1));

kk++)

\*(\*(\*(w+ii)+jj)+kk)

=\*(\*(\*(wi+ii)+jj)+kk);

}

}

/\*FORWARD PROPAGATION\*/

for(i=1; i<nlayer; i++)

{

if(i==nlayer-1) vsum=0.0;

for(k=0; k<(\*(nnode+i)); k++)

{

\*(\*(v+i)+k)=0.0;

for(j=0; j<(\*(nnode+i-1)); j++)

\*(\*(v+i)+k)+=(\*(\*(\*(w+i-1)+j)+k))

\*(\*(\*(v+i-1)+j));

\*(\*(v+i)+k)+=(\*(\*(b+i-1)+k));

if(i<(nlayer-1))

\*(\*(v+i)+k)

=1.0/(1.0+exp(-(\*(\*(v+i)+k))));

else if(iactivate==SIGMOID)

\*(\*(v+i)+k)

=1.0/(1.0+exp(-(\*(\*(v+i)+k))));

else if(iactivate==SOFTMAX)

vsum+=exp(\*(\*(v+i)+k));

}

/\*SOFTMAX FOR FINAL LAYER\*/

if(iactivate==SOFTMAX

&& i==nlayer-1)

{

for(k=0; k<(\*(nnode+i)); k++)

\*(\*(v+i)+k)=exp(\*(\*(v+i)+k))/vsum;

}

}

for(k=0; k<no; k++)

{

/\*RESIDUAL SUM OF SQUARES\*/

if(iloss==RESIDUALSUMSQUARE)

{

dv=(\*(\*(tc+(\*(ic+m)))+k))

-(\*(\*(v+nlayer-1)+k));

floss+=0.5\*dv\*dv;

}

/\*CROSS ENTROPY\*/

if(iloss==CROSSENTROPY)

{

dv=-(\*(\*(tc+(\*(ic+m)))+k))\*log(\*(\*(v+nlayer-1)+k))/(double)(nrefer\*no);

floss+=dv;

}

}

for(ii=0; ii<nlayer; ii++)

{

if(ii==0 || ii==(nlayer-1)) SetConsoleTextAttribute(hndc,FOREGROUND\_INTENSITY | FOREGROUND\_GREEN | FOREGROUND\_BLUE);

if(ii==0) fprintf(stderr,"STEP %2d REFERENCE %d : v =",nstep,m+1);

else fprintf(stderr," ");

for(jj=0; jj<(\*(nnode+ii)); jj++) fprintf(stderr," %9.5f",\*(\*(v+ii)+jj));

fprintf(stderr,"\n");

if(ii==0 || ii==(nlayer-1)) SetConsoleTextAttribute(hndc,cb.wAttributes);

}

/\*gets(non);\*/

if(m==(nrefer-1))

{

SetConsoleTextAttribute(hndc,FOREGROUND\_INTENSITY | FOREGROUND\_RED | FOREGROUND\_BLUE);

fprintf(stderr,"STEP %d : LOSS = %9.5f DLOSS = %9.5f\n",nstep,floss,(ploss-floss));

SetConsoleTextAttribute(hndc,cb.wAttributes);

}

if(m==(nrefer-1) && floss<eps)

{

fprintf(stderr,"COMPLETED : LOSS = %9.5f < eps = %9.5f\n",floss,eps);

gets(non);

iflag=1;

break;

}

if(nstep>=2 && m==(nrefer-1) && (ploss-floss)<(0.1\*eps))

{

fprintf(stderr,"COMPLETED : DLOSS = %9.5f - %9.5f = %9.5f < eps = %9.5f\n",

ploss,floss,(ploss-floss),0.1\*eps);

gets(non);

iflag=1;

break;

}

/\*BACK PROPAGATION\*/

fprintf(stderr,"BACK PROPAGATION\n");

for(k=0; k<(\*(nnode+nlayer-1)); k++)

{

/\*SIGMOID & RESIDUAL SUM OF SQUARES\*/

if(iactivate==SIGMOID && iloss==RESIDUALSUMSQUARE)

{

\*(\*(dldb+(nlayer-2))+k)=(\*(\*(v+nlayer-1)+k)-\*(\*(tc+(\*(ic+m)))+k))

\*(1.0-\*(\*(v+nlayer-1)+k))

\*(\*(\*(v+nlayer-1)+k));

}

/\*SOFTMAX & CROSS ENTROPY\*/

if(iactivate==SOFTMAX && iloss==CROSSENTROPY)

{

\*(\*(dldb+(nlayer-2))+k)=\*(\*(v+nlayer-1)+k)-\*(\*(tc+(\*(ic+m)))+k);

}

for(j=0; j<(\*(nnode+nlayer-2)); j++)

{

\*(\*(\*(dldw+(nlayer-2))+j)+k)=(\*(\*(dldb+(nlayer-2))+k))

\*(\*(\*(v+nlayer-2)+j));

}

}

for(k=(nlayer-3); k>=0; k--)

{

for(j=0; j<(\*(nnode+k+1)); j++)

{

\*(\*(dldv+k+1)+j)=0.0;

for(i=0; i<(\*(nnode+k+2)); i++)

{

\*(\*(dldv+k+1)+j)+=(\*(\*(dldb+k+1)+i))

\*(\*(\*(\*(w+k+1)+j)+i));

}

}

for(j=0; j<(\*(nnode+k+1)); j++)

{

/\*SIGMOID & RESIDUAL SUM OF SQUARES\*/

/\*SIGMOID & CROSS ENTROPY\*/

\*(\*(dldb+k)+j)=(\*(\*(dldv+k+1)+j))

\*(1.0-\*(\*(v+k+1)+j))

\*(\*(\*(v+k+1)+j));

for(i=0; i<(\*(nnode+k)); i++)

{

\*(\*(\*(dldw+k)+i)+j)=(\*(\*(dldb+k)+j))

\*(\*(\*(v+k)+i));

}

}

}

/\*UPDATE\*/

for(k=0; k<(nlayer-1); k++)

{

for(j=0; j<(\*(nnode+k+1)); j++)

{

\*(\*(bf+k)+j)-=urate\*(\*(\*(dldb+k)+j))/(double)nrefer;

for(i=0; i<(\*(nnode+k)); i++) \*(\*(\*(wf+k)+i)+j)-=urate\*(\*(\*(\*(dldw+k)+i)+j))/(double)nrefer;

}

}

for(ii=0; ii<(nlayer-1); ii++)

{

for(jj=0; jj<(\*(nnode+ii)); jj++)

{

if(ii==0 && jj==0) fprintf(stderr,"DERIVATIVE WEIGHT : dldw%d%d =",ii+1,jj+1);

else fprintf(stderr," dldw%d%d =",ii+1,jj+1);

for(kk=0; kk<(\*(nnode+ii+1)); kk++) fprintf(stderr," %9.5f",\*(\*(\*(dldw+ii)+jj)+kk));

fprintf(stderr,"\n");

}

}

for(ii=0; ii<(nlayer-1); ii++)

{

if(ii==0) fprintf(stderr,"DERIVATIVE BIAS : b%d =",ii+1);

else fprintf(stderr," b%d =",ii+1);

for(jj=0; jj<(\*(nnode+ii+1)); jj++) fprintf(stderr," %9.5f",\*(\*(dldb+ii)+jj));

fprintf(stderr,"\n");

}

for(ii=0; ii<(nlayer-1); ii++)

{

for(jj=0; jj<(\*(nnode+ii)); jj++)

{

if(ii==0 && jj==0) fprintf(stderr,"UPDATED WEIGHT : w%d%d =",ii+1,jj+1);

else fprintf(stderr," w%d%d =",ii+1,jj+1);

for(kk=0; kk<(\*(nnode+ii+1)); kk++) fprintf(stderr," %9.5f",\*(\*(\*(wf+ii)+jj)+kk));

fprintf(stderr,"\n");

}

}

for(ii=0; ii<(nlayer-1); ii++)

{

if(ii==0) fprintf(stderr,"UPDATED BIAS : b%d =",ii+1);

else fprintf(stderr," b%d =",ii+1);

for(jj=0; jj<(\*(nnode+ii+1)); jj++) fprintf(stderr," %9.5f",\*(\*(bf+ii)+jj));

fprintf(stderr,"\n");

}

}

if(iflag==1) break;

ploss=floss;

SetConsoleTextAttribute(hndc,FOREGROUND\_INTENSITY | FOREGROUND\_RED | FOREGROUND\_BLUE);

fprintf(stderr,"STEP %2d : LOSS = %9.5f\n",nstep,floss);

SetConsoleTextAttribute(hndc,cb.wAttributes);

}

/\*INPUT SPECIMEN\*/

for(i=0; i<ni; i++) \*(\*(v+0)+i)=\*(s+i);

SetConsoleTextAttribute(hndc,FOREGROUND\_INTENSITY | FOREGROUND\_GREEN | FOREGROUND\_BLUE);

fprintf(stderr,"SPECIMEN INPUT : v =");

for(jj=0; jj<ni; jj++) fprintf(stderr," %8.3f",\*(\*(v+0)+jj));

fprintf(stderr,"\n");

SetConsoleTextAttribute(hndc,cb.wAttributes);

for(ii=0; ii<(nlayer-1); ii++)

{

for(jj=0; jj<(\*(nnode+ii)); jj++)

{

if(ii==0 && jj==0) fprintf(stderr,"FINAL WEIGHT : w%d%d =",ii+1,jj+1);

else fprintf(stderr," w%d%d =",ii+1,jj+1);

for(kk=0; kk<(\*(nnode+ii+1)); kk++) fprintf(stderr," %9.5f",\*(\*(\*(wf+ii)+jj)+kk));

fprintf(stderr,"\n");

}

}

for(ii=0; ii<(nlayer-1); ii++)

{

if(ii==0) fprintf(stderr,"FINAL BIAS : b%d =",ii+1);

else fprintf(stderr," b%d =",ii+1);

for(jj=0; jj<(\*(nnode+ii+1)); jj++) fprintf(stderr," %9.5f",\*(\*(bf+ii)+jj));

fprintf(stderr,"\n");

}

/\*FORWARD PROPAGATION\*/

for(i=1; i<nlayer; i++)

{

if(i==nlayer-1) vsum=0.0;

for(k=0; k<(\*(nnode+i)); k++)

{

\*(\*(v+i)+k)=0.0;

for(j=0; j<(\*(nnode+i-1)); j++) \*(\*(v+i)+k)+=(\*(\*(\*(wf+i-1)+j)+k))\*(\*(\*(v+i-1)+j));

\*(\*(v+i)+k)+=(\*(\*(bf+i-1)+k));

/\*SIGMOID\*/

if(i<(nlayer-1))

{

\*(\*(v+i)+k)=1.0/(1.0+exp(-(\*(\*(v+i)+k))));

fprintf(stderr,"SIGMOID VALUE : SIG{v%d%d} = %9.5f\n",i,k,\*(\*(v+i)+k));

}

else if(iactivate==SIGMOID) \*(\*(v+i)+k)=1.0/(1.0+exp(-(\*(\*(v+i)+k))));

else if(iactivate==SOFTMAX) vsum+=exp(\*(\*(v+i)+k));

}

/\*SOFTMAX FOR FINAL LAYER\*/

if(iactivate==SOFTMAX && i==nlayer-1)

{

for(k=0; k<(\*(nnode+i)); k++) \*(\*(v+i)+k)=exp(\*(\*(v+i)+k))/vsum;

}

}

SetConsoleTextAttribute(hndc,FOREGROUND\_INTENSITY | FOREGROUND\_GREEN | FOREGROUND\_BLUE);

fprintf(stderr,"SPECIMEN OUTPUT : v =");

for(jj=0; jj<no; jj++) fprintf(stderr," %9.5f",\*(\*(v+nlayer-1)+jj));

fprintf(stderr,"\n");

SetConsoleTextAttribute(hndc,cb.wAttributes);

mlabel=0;

for(ii=0; ii<ncategory; ii++)

{

floss=0.0;

for(jj=0; jj<no; jj++)

{

/\*RESIDUAL SUM OF SQUARE\*/

if(iloss==RESIDUALSUMSQUARE)

{

dv=(\*(\*(tc+ii)+jj))-(\*(\*(v+nlayer-1)+jj));

floss+=0.5\*dv\*dv;

}

/\*CROSS ENTROPY\*/

if(iloss==CROSSENTROPY)

{

dv=-(\*(\*(tc+ii)+jj))\*log(\*(\*(v+nlayer-1)+jj))/(double)no;

floss+=dv;

}

}

if(ii==0) mloss=floss;

else if(mloss>floss)

{

mloss=floss;

mlabel=ii;

}

fprintf(stderr,"CATEGORY %d LOSS = %9.5f\n",ii+1,floss);

}

SetConsoleTextAttribute(hndc,FOREGROUND\_INTENSITY | FOREGROUND\_RED | FOREGROUND\_BLUE);

fprintf(stderr,"CLOSEST CATEGORY = %d MINIMUM LOSS = %9.5f\n",mlabel+1,mloss);

SetConsoleTextAttribute(hndc,cb.wAttributes);

gets(non);

/\*FREE MEMORY\*/

freematrix(tc,ncategory);

freematrix(r,nrefer);

freematrix(v,nlayer);

freematrix(dldv,nlayer);

freematrix(b,nlayer);

freematrix(bi,nlayer);

freematrix(bf,nlayer);

freematrix(dldb,nlayer);

for(ii=0; ii<(nlayer-1); ii++)

{

freematrix(\*(w+ii),\*(nnode+ii));

freematrix(\*(wi+ii),\*(nnode+ii));

freematrix(\*(wf+ii),\*(nnode+ii));

freematrix(\*(dldw+ii),\*(nnode+ii));

}

free(nnode);

free(ic);

return NULL;

}/\*main\*/

double \*mallocdoublevector(int vsize)

/\*MALLOC DOUBLE VECTOR.\*/

{

double \*v;

v=(double \*)malloc(vsize\*sizeof(double));

return v;

}/\*mallocdoublevector\*/

double \*\*mallocdoublematrix(int nline,int nrow)

/\*MALLOC DOUBLE MATRIX.\*/

{

int i;

double \*\*mtx;

mtx=(double \*\*)malloc(nline\*sizeof(double \*));

for(i=0;i<nline;i++) \*(mtx+i)=(double \*)malloc(nrow\*sizeof(double));

return mtx;

}/\*mallocdoublematrix\*/

void freematrix(double \*\*mtx,int nline)

/\*FREE MATRIX.\*/

{

int i;

for(i=0;i<nline;i++) free(\*(mtx+i));

free(mtx);

return;

}/\*freematrix\*/