#include "local\_layer.h"

#include "utils.h"

#include "im2col.h"

#include "col2im.h"

#include "blas.h"

#include "gemm.h"

#include <stdio.h>

#include <time.h>

int local\_out\_height(local\_layer l)

{

int h = l.h;

if (!l.pad) h -= l.size;

else h -= 1;

return h/l.stride + 1;

}

int local\_out\_width(local\_layer l)

{

int w = l.w;

if (!l.pad) w -= l.size;

else w -= 1;

return w/l.stride + 1;

}

local\_layer make\_local\_layer(int batch, int h, int w, int c, int n, int size, int stride, int pad, ACTIVATION activation)

{

int i;

local\_layer l = { (LAYER\_TYPE)0 };

l.type = LOCAL;

l.h = h;

l.w = w;

l.c = c;

l.n = n;

l.batch = batch;

l.stride = stride;

l.size = size;

l.pad = pad;

int out\_h = local\_out\_height(l);

int out\_w = local\_out\_width(l);

int locations = out\_h\*out\_w;

l.out\_h = out\_h;

l.out\_w = out\_w;

l.out\_c = n;

l.outputs = l.out\_h \* l.out\_w \* l.out\_c;

l.inputs = l.w \* l.h \* l.c;

l.weights = (float\*)xcalloc(c \* n \* size \* size \* locations, sizeof(float));

l.weight\_updates = (float\*)xcalloc(c \* n \* size \* size \* locations, sizeof(float));

l.biases = (float\*)xcalloc(l.outputs, sizeof(float));

l.bias\_updates = (float\*)xcalloc(l.outputs, sizeof(float));

// float scale = 1./sqrt(size\*size\*c);

float scale = sqrt(2./(size\*size\*c));

for(i = 0; i < c\*n\*size\*size; ++i) l.weights[i] = scale\*rand\_uniform(-1,1);

l.col\_image = (float\*)xcalloc(out\_h \* out\_w \* size \* size \* c, sizeof(float));

l.output = (float\*)xcalloc(l.batch \* out\_h \* out\_w \* n, sizeof(float));

l.delta = (float\*)xcalloc(l.batch \* out\_h \* out\_w \* n, sizeof(float));

l.forward = forward\_local\_layer;

l.backward = backward\_local\_layer;

l.update = update\_local\_layer;

#ifdef GPU

l.forward\_gpu = forward\_local\_layer\_gpu;

l.backward\_gpu = backward\_local\_layer\_gpu;

l.update\_gpu = update\_local\_layer\_gpu;

l.weights\_gpu = cuda\_make\_array(l.weights, c\*n\*size\*size\*locations);

l.weight\_updates\_gpu = cuda\_make\_array(l.weight\_updates, c\*n\*size\*size\*locations);

l.biases\_gpu = cuda\_make\_array(l.biases, l.outputs);

l.bias\_updates\_gpu = cuda\_make\_array(l.bias\_updates, l.outputs);

l.col\_image\_gpu = cuda\_make\_array(l.col\_image, out\_h\*out\_w\*size\*size\*c);

l.delta\_gpu = cuda\_make\_array(l.delta, l.batch\*out\_h\*out\_w\*n);

l.output\_gpu = cuda\_make\_array(l.output, l.batch\*out\_h\*out\_w\*n);

#endif

l.activation = activation;

fprintf(stderr, "Local Layer: %d x %d x %d image, %d filters -> %d x %d x %d image\n", h,w,c,n, out\_h, out\_w, n);

return l;

}

void forward\_local\_layer(const local\_layer l, network\_state state)

{

int out\_h = local\_out\_height(l);

int out\_w = local\_out\_width(l);

int i, j;

int locations = out\_h \* out\_w;

for(i = 0; i < l.batch; ++i){

copy\_cpu(l.outputs, l.biases, 1, l.output + i\*l.outputs, 1);

}

for(i = 0; i < l.batch; ++i){

float \*input = state.input + i\*l.w\*l.h\*l.c;

im2col\_cpu(input, l.c, l.h, l.w,

l.size, l.stride, l.pad, l.col\_image);

float \*output = l.output + i\*l.outputs;

for(j = 0; j < locations; ++j){

float \*a = l.weights + j\*l.size\*l.size\*l.c\*l.n;

float \*b = l.col\_image + j;

float \*c = output + j;

int m = l.n;

int n = 1;

int k = l.size\*l.size\*l.c;

gemm(0,0,m,n,k,1,a,k,b,locations,1,c,locations);

}

}

activate\_array(l.output, l.outputs\*l.batch, l.activation);

}

void backward\_local\_layer(local\_layer l, network\_state state)

{

int i, j;

int locations = l.out\_w\*l.out\_h;

gradient\_array(l.output, l.outputs\*l.batch, l.activation, l.delta);

for(i = 0; i < l.batch; ++i){

axpy\_cpu(l.outputs, 1, l.delta + i\*l.outputs, 1, l.bias\_updates, 1);

}

for(i = 0; i < l.batch; ++i){

float \*input = state.input + i\*l.w\*l.h\*l.c;

im2col\_cpu(input, l.c, l.h, l.w,

l.size, l.stride, l.pad, l.col\_image);

for(j = 0; j < locations; ++j){

float \*a = l.delta + i\*l.outputs + j;

float \*b = l.col\_image + j;

float \*c = l.weight\_updates + j\*l.size\*l.size\*l.c\*l.n;

int m = l.n;

int n = l.size\*l.size\*l.c;

int k = 1;

gemm(0,1,m,n,k,1,a,locations,b,locations,1,c,n);

}

if(state.delta){

for(j = 0; j < locations; ++j){

float \*a = l.weights + j\*l.size\*l.size\*l.c\*l.n;

float \*b = l.delta + i\*l.outputs + j;

float \*c = l.col\_image + j;

int m = l.size\*l.size\*l.c;

int n = 1;

int k = l.n;

gemm(1,0,m,n,k,1,a,m,b,locations,0,c,locations);

}

col2im\_cpu(l.col\_image, l.c, l.h, l.w, l.size, l.stride, l.pad, state.delta+i\*l.c\*l.h\*l.w);

}

}

}

void update\_local\_layer(local\_layer l, int batch, float learning\_rate, float momentum, float decay)

{

int locations = l.out\_w\*l.out\_h;

int size = l.size\*l.size\*l.c\*l.n\*locations;

axpy\_cpu(l.outputs, learning\_rate/batch, l.bias\_updates, 1, l.biases, 1);

scal\_cpu(l.outputs, momentum, l.bias\_updates, 1);

axpy\_cpu(size, -decay\*batch, l.weights, 1, l.weight\_updates, 1);

axpy\_cpu(size, learning\_rate/batch, l.weight\_updates, 1, l.weights, 1);

scal\_cpu(size, momentum, l.weight\_updates, 1);

}

#ifdef GPU

void forward\_local\_layer\_gpu(const local\_layer l, network\_state state)

{

int out\_h = local\_out\_height(l);

int out\_w = local\_out\_width(l);

int i, j;

int locations = out\_h \* out\_w;

for(i = 0; i < l.batch; ++i){

copy\_ongpu(l.outputs, l.biases\_gpu, 1, l.output\_gpu + i\*l.outputs, 1);

}

for(i = 0; i < l.batch; ++i){

float \*input = state.input + i\*l.w\*l.h\*l.c;

im2col\_ongpu(input, l.c, l.h, l.w,

l.size, l.stride, l.pad, l.col\_image\_gpu);

float \*output = l.output\_gpu + i\*l.outputs;

for(j = 0; j < locations; ++j){

float \*a = l.weights\_gpu + j\*l.size\*l.size\*l.c\*l.n;

float \*b = l.col\_image\_gpu + j;

float \*c = output + j;

int m = l.n;

int n = 1;

int k = l.size\*l.size\*l.c;

gemm\_ongpu(0,0,m,n,k,1,a,k,b,locations,1,c,locations);

}

}

activate\_array\_ongpu(l.output\_gpu, l.outputs\*l.batch, l.activation);

}

void backward\_local\_layer\_gpu(local\_layer l, network\_state state)

{

int i, j;

int locations = l.out\_w\*l.out\_h;

gradient\_array\_ongpu(l.output\_gpu, l.outputs\*l.batch, l.activation, l.delta\_gpu);

for(i = 0; i < l.batch; ++i){

axpy\_ongpu(l.outputs, 1, l.delta\_gpu + i\*l.outputs, 1, l.bias\_updates\_gpu, 1);

}

for(i = 0; i < l.batch; ++i){

float \*input = state.input + i\*l.w\*l.h\*l.c;

im2col\_ongpu(input, l.c, l.h, l.w,

l.size, l.stride, l.pad, l.col\_image\_gpu);

for(j = 0; j < locations; ++j){

float \*a = l.delta\_gpu + i\*l.outputs + j;

float \*b = l.col\_image\_gpu + j;

float \*c = l.weight\_updates\_gpu + j\*l.size\*l.size\*l.c\*l.n;

int m = l.n;

int n = l.size\*l.size\*l.c;

int k = 1;

gemm\_ongpu(0,1,m,n,k,1,a,locations,b,locations,1,c,n);

}

if(state.delta){

for(j = 0; j < locations; ++j){

float \*a = l.weights\_gpu + j\*l.size\*l.size\*l.c\*l.n;

float \*b = l.delta\_gpu + i\*l.outputs + j;

float \*c = l.col\_image\_gpu + j;

int m = l.size\*l.size\*l.c;

int n = 1;

int k = l.n;

gemm\_ongpu(1,0,m,n,k,1,a,m,b,locations,0,c,locations);

}

col2im\_ongpu(l.col\_image\_gpu, l.c, l.h, l.w, l.size, l.stride, l.pad, state.delta+i\*l.c\*l.h\*l.w);

}

}

}

void update\_local\_layer\_gpu(local\_layer l, int batch, float learning\_rate, float momentum, float decay, float loss\_scale)

{

int locations = l.out\_w\*l.out\_h;

int size = l.size\*l.size\*l.c\*l.n\*locations;

axpy\_ongpu(l.outputs, learning\_rate/batch, l.bias\_updates\_gpu, 1, l.biases\_gpu, 1);

scal\_ongpu(l.outputs, momentum, l.bias\_updates\_gpu, 1);

axpy\_ongpu(size, -decay\*batch, l.weights\_gpu, 1, l.weight\_updates\_gpu, 1);

axpy\_ongpu(size, learning\_rate/batch, l.weight\_updates\_gpu, 1, l.weights\_gpu, 1);

scal\_ongpu(size, momentum, l.weight\_updates\_gpu, 1);

}

void pull\_local\_layer(local\_layer l)

{

int locations = l.out\_w\*l.out\_h;

int size = l.size\*l.size\*l.c\*l.n\*locations;

cuda\_pull\_array(l.weights\_gpu, l.weights, size);

cuda\_pull\_array(l.biases\_gpu, l.biases, l.outputs);

}

void push\_local\_layer(local\_layer l)

{

int locations = l.out\_w\*l.out\_h;

int size = l.size\*l.size\*l.c\*l.n\*locations;

cuda\_push\_array(l.weights\_gpu, l.weights, size);

cuda\_push\_array(l.biases\_gpu, l.biases, l.outputs);

}

#endif