#include "normalization\_layer.h"

#include "blas.h"

#include "utils.h"

#include <stdio.h>

layer make\_normalization\_layer(int batch, int w, int h, int c, int size, float alpha, float beta, float kappa)

{

fprintf(stderr, "Local Response Normalization Layer: %d x %d x %d image, %d size\n", w,h,c,size);

layer layer = { (LAYER\_TYPE)0 };

layer.type = NORMALIZATION;

layer.batch = batch;

layer.h = layer.out\_h = h;

layer.w = layer.out\_w = w;

layer.c = layer.out\_c = c;

layer.kappa = kappa;

layer.size = size;

layer.alpha = alpha;

layer.beta = beta;

layer.output = (float\*)xcalloc(h \* w \* c \* batch, sizeof(float));

layer.delta = (float\*)xcalloc(h \* w \* c \* batch, sizeof(float));

layer.squared = (float\*)xcalloc(h \* w \* c \* batch, sizeof(float));

layer.norms = (float\*)xcalloc(h \* w \* c \* batch, sizeof(float));

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

layer.outputs = layer.inputs;

layer.forward = forward\_normalization\_layer;

layer.backward = backward\_normalization\_layer;

#ifdef GPU

layer.forward\_gpu = forward\_normalization\_layer\_gpu;

layer.backward\_gpu = backward\_normalization\_layer\_gpu;

layer.output\_gpu = cuda\_make\_array(layer.output, h \* w \* c \* batch);

layer.delta\_gpu = cuda\_make\_array(layer.delta, h \* w \* c \* batch);

layer.squared\_gpu = cuda\_make\_array(layer.squared, h \* w \* c \* batch);

layer.norms\_gpu = cuda\_make\_array(layer.norms, h \* w \* c \* batch);

#endif

return layer;

}

void resize\_normalization\_layer(layer \*layer, int w, int h)

{

int c = layer->c;

int batch = layer->batch;

layer->h = h;

layer->w = w;

layer->out\_h = h;

layer->out\_w = w;

layer->inputs = w\*h\*c;

layer->outputs = layer->inputs;

layer->output = (float\*)xrealloc(layer->output, h \* w \* c \* batch \* sizeof(float));

layer->delta = (float\*)xrealloc(layer->delta, h \* w \* c \* batch \* sizeof(float));

layer->squared = (float\*)xrealloc(layer->squared, h \* w \* c \* batch \* sizeof(float));

layer->norms = (float\*)xrealloc(layer->norms, h \* w \* c \* batch \* sizeof(float));

#ifdef GPU

cuda\_free(layer->output\_gpu);

cuda\_free(layer->delta\_gpu);

cuda\_free(layer->squared\_gpu);

cuda\_free(layer->norms\_gpu);

layer->output\_gpu = cuda\_make\_array(layer->output, h \* w \* c \* batch);

layer->delta\_gpu = cuda\_make\_array(layer->delta, h \* w \* c \* batch);

layer->squared\_gpu = cuda\_make\_array(layer->squared, h \* w \* c \* batch);

layer->norms\_gpu = cuda\_make\_array(layer->norms, h \* w \* c \* batch);

#endif

}

void forward\_normalization\_layer(const layer layer, network\_state state)

{

int k,b;

int w = layer.w;

int h = layer.h;

int c = layer.c;

scal\_cpu(w\*h\*c\*layer.batch, 0, layer.squared, 1);

for(b = 0; b < layer.batch; ++b){

float \*squared = layer.squared + w\*h\*c\*b;

float \*norms = layer.norms + w\*h\*c\*b;

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

pow\_cpu(w\*h\*c, 2, input, 1, squared, 1);

const\_cpu(w\*h, layer.kappa, norms, 1);

for(k = 0; k < layer.size/2; ++k){

axpy\_cpu(w\*h, layer.alpha, squared + w\*h\*k, 1, norms, 1);

}

for(k = 1; k < layer.c; ++k){

copy\_cpu(w\*h, norms + w\*h\*(k-1), 1, norms + w\*h\*k, 1);

int prev = k - ((layer.size-1)/2) - 1;

int next = k + (layer.size/2);

if(prev >= 0) axpy\_cpu(w\*h, -layer.alpha, squared + w\*h\*prev, 1, norms + w\*h\*k, 1);

if(next < layer.c) axpy\_cpu(w\*h, layer.alpha, squared + w\*h\*next, 1, norms + w\*h\*k, 1);

}

}

pow\_cpu(w\*h\*c\*layer.batch, -layer.beta, layer.norms, 1, layer.output, 1);

mul\_cpu(w\*h\*c\*layer.batch, state.input, 1, layer.output, 1);

}

void backward\_normalization\_layer(const layer layer, network\_state state)

{

// TODO This is approximate ;-)

// Also this should add in to delta instead of overwritting.

int w = layer.w;

int h = layer.h;

int c = layer.c;

pow\_cpu(w\*h\*c\*layer.batch, -layer.beta, layer.norms, 1, state.delta, 1);

mul\_cpu(w\*h\*c\*layer.batch, layer.delta, 1, state.delta, 1);

}

#ifdef GPU

void forward\_normalization\_layer\_gpu(const layer layer, network\_state state)

{

int k,b;

int w = layer.w;

int h = layer.h;

int c = layer.c;

scal\_ongpu(w\*h\*c\*layer.batch, 0, layer.squared\_gpu, 1);

for(b = 0; b < layer.batch; ++b){

float \*squared = layer.squared\_gpu + w\*h\*c\*b;

float \*norms = layer.norms\_gpu + w\*h\*c\*b;

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

pow\_ongpu(w\*h\*c, 2, input, 1, squared, 1);

const\_ongpu(w\*h, layer.kappa, norms, 1);

for(k = 0; k < layer.size/2; ++k){

axpy\_ongpu(w\*h, layer.alpha, squared + w\*h\*k, 1, norms, 1);

}

for(k = 1; k < layer.c; ++k){

copy\_ongpu(w\*h, norms + w\*h\*(k-1), 1, norms + w\*h\*k, 1);

int prev = k - ((layer.size-1)/2) - 1;

int next = k + (layer.size/2);

if(prev >= 0) axpy\_ongpu(w\*h, -layer.alpha, squared + w\*h\*prev, 1, norms + w\*h\*k, 1);

if(next < layer.c) axpy\_ongpu(w\*h, layer.alpha, squared + w\*h\*next, 1, norms + w\*h\*k, 1);

}

}

pow\_ongpu(w\*h\*c\*layer.batch, -layer.beta, layer.norms\_gpu, 1, layer.output\_gpu, 1);

mul\_ongpu(w\*h\*c\*layer.batch, state.input, 1, layer.output\_gpu, 1);

}

void backward\_normalization\_layer\_gpu(const layer layer, network\_state state)

{

// TODO This is approximate ;-)

int w = layer.w;

int h = layer.h;

int c = layer.c;

pow\_ongpu(w\*h\*c\*layer.batch, -layer.beta, layer.norms\_gpu, 1, state.delta, 1);

mul\_ongpu(w\*h\*c\*layer.batch, layer.delta\_gpu, 1, state.delta, 1);

}

#endif