#include "cost\_layer.h"

#include "utils.h"

#include "dark\_cuda.h"

#include "blas.h"

#include <math.h>

#include <string.h>

#include <stdlib.h>

#include <stdio.h>

COST\_TYPE get\_cost\_type(char \*s)

{

if (strcmp(s, "sse")==0) return SSE;

if (strcmp(s, "masked")==0) return MASKED;

if (strcmp(s, "smooth")==0) return SMOOTH;

fprintf(stderr, "Couldn't find cost type %s, going with SSE\n", s);

return SSE;

}

char \*get\_cost\_string(COST\_TYPE a)

{

switch(a){

case SSE:

return "sse";

case MASKED:

return "masked";

case SMOOTH:

return "smooth";

default:

return "sse";

}

}

cost\_layer make\_cost\_layer(int batch, int inputs, COST\_TYPE cost\_type, float scale)

{

fprintf(stderr, "cost %4d\n", inputs);

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

l.type = COST;

l.scale = scale;

l.batch = batch;

l.inputs = inputs;

l.outputs = inputs;

l.cost\_type = cost\_type;

l.delta = (float\*)xcalloc(inputs \* batch, sizeof(float));

l.output = (float\*)xcalloc(inputs \* batch, sizeof(float));

l.cost = (float\*)xcalloc(1, sizeof(float));

l.forward = forward\_cost\_layer;

l.backward = backward\_cost\_layer;

#ifdef GPU

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

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

l.delta\_gpu = cuda\_make\_array(l.delta, inputs\*batch);

l.output\_gpu = cuda\_make\_array(l.output, inputs\*batch);

#endif

return l;

}

void resize\_cost\_layer(cost\_layer \*l, int inputs)

{

l->inputs = inputs;

l->outputs = inputs;

l->delta = (float\*)xrealloc(l->delta, inputs \* l->batch \* sizeof(float));

l->output = (float\*)xrealloc(l->output, inputs \* l->batch \* sizeof(float));

#ifdef GPU

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

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

l->delta\_gpu = cuda\_make\_array(l->delta, inputs\*l->batch);

l->output\_gpu = cuda\_make\_array(l->output, inputs\*l->batch);

#endif

}

void forward\_cost\_layer(cost\_layer l, network\_state state)

{

if (!state.truth) return;

if(l.cost\_type == MASKED){

int i;

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

if(state.truth[i] == SECRET\_NUM) state.input[i] = SECRET\_NUM;

}

}

if(l.cost\_type == SMOOTH){

smooth\_l1\_cpu(l.batch\*l.inputs, state.input, state.truth, l.delta, l.output);

} else {

l2\_cpu(l.batch\*l.inputs, state.input, state.truth, l.delta, l.output);

}

l.cost[0] = sum\_array(l.output, l.batch\*l.inputs);

}

void backward\_cost\_layer(const cost\_layer l, network\_state state)

{

axpy\_cpu(l.batch\*l.inputs, l.scale, l.delta, 1, state.delta, 1);

}

#ifdef GPU

void pull\_cost\_layer(cost\_layer l)

{

cuda\_pull\_array(l.delta\_gpu, l.delta, l.batch\*l.inputs);

}

void push\_cost\_layer(cost\_layer l)

{

cuda\_push\_array(l.delta\_gpu, l.delta, l.batch\*l.inputs);

}

int float\_abs\_compare (const void \* a, const void \* b)

{

float fa = \*(const float\*) a;

if(fa < 0) fa = -fa;

float fb = \*(const float\*) b;

if(fb < 0) fb = -fb;

return (fa > fb) - (fa < fb);

}

void forward\_cost\_layer\_gpu(cost\_layer l, network\_state state)

{

if (!state.truth) return;

if (l.cost\_type == MASKED) {

mask\_ongpu(l.batch\*l.inputs, state.input, SECRET\_NUM, state.truth);

}

if(l.cost\_type == SMOOTH){

smooth\_l1\_gpu(l.batch\*l.inputs, state.input, state.truth, l.delta\_gpu, l.output\_gpu);

} else {

l2\_gpu(l.batch\*l.inputs, state.input, state.truth, l.delta\_gpu, l.output\_gpu);

}

if(l.ratio){

cuda\_pull\_array(l.delta\_gpu, l.delta, l.batch\*l.inputs);

qsort(l.delta, l.batch\*l.inputs, sizeof(float), float\_abs\_compare);

int n = (1-l.ratio) \* l.batch\*l.inputs;

float thresh = l.delta[n];

thresh = 0;

printf("%f\n", thresh);

supp\_ongpu(l.batch\*l.inputs, thresh, l.delta\_gpu, 1);

}

cuda\_pull\_array(l.output\_gpu, l.output, l.batch\*l.inputs);

l.cost[0] = sum\_array(l.output, l.batch\*l.inputs);

}

void backward\_cost\_layer\_gpu(const cost\_layer l, network\_state state)

{

axpy\_ongpu(l.batch\*l.inputs, l.scale, l.delta\_gpu, 1, state.delta, 1);

}

#endif