#include "route\_layer.h"

#include "utils.h"

#include "dark\_cuda.h"

#include "blas.h"

#include <stdio.h>

route\_layer make\_route\_layer(int batch, int n, int \*input\_layers, int \*input\_sizes, int groups, int group\_id)

{

fprintf(stderr,"route ");

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

l.type = ROUTE;

l.batch = batch;

l.n = n;

l.input\_layers = input\_layers;

l.input\_sizes = input\_sizes;

l.groups = groups;

l.group\_id = group\_id;

int i;

int outputs = 0;

for(i = 0; i < n; ++i){

fprintf(stderr," %d", input\_layers[i]);

outputs += input\_sizes[i];

}

outputs = outputs / groups;

l.outputs = outputs;

l.inputs = outputs;

//fprintf(stderr, " inputs = %d \t outputs = %d, groups = %d, group\_id = %d \n", l.inputs, l.outputs, l.groups, l.group\_id);

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

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

l.forward = forward\_route\_layer;

l.backward = backward\_route\_layer;

#ifdef GPU

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

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

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

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

#endif

return l;

}

void resize\_route\_layer(route\_layer \*l, network \*net)

{

int i;

layer first = net->layers[l->input\_layers[0]];

l->out\_w = first.out\_w;

l->out\_h = first.out\_h;

l->out\_c = first.out\_c;

l->outputs = first.outputs;

l->input\_sizes[0] = first.outputs;

for(i = 1; i < l->n; ++i){

int index = l->input\_layers[i];

layer next = net->layers[index];

l->outputs += next.outputs;

l->input\_sizes[i] = next.outputs;

if(next.out\_w == first.out\_w && next.out\_h == first.out\_h){

l->out\_c += next.out\_c;

}else{

printf("Error: Different size of input layers: %d x %d, %d x %d\n", next.out\_w, next.out\_h, first.out\_w, first.out\_h);

l->out\_h = l->out\_w = l->out\_c = 0;

exit(EXIT\_FAILURE);

}

}

l->out\_c = l->out\_c / l->groups;

l->outputs = l->outputs / l->groups;

l->inputs = l->outputs;

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

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

#ifdef GPU

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

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

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

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

#endif

}

void forward\_route\_layer(const route\_layer l, network\_state state)

{

int i, j;

int offset = 0;

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

int index = l.input\_layers[i];

float \*input = state.net.layers[index].output;

int input\_size = l.input\_sizes[i];

int part\_input\_size = input\_size / l.groups;

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

//copy\_cpu(input\_size, input + j\*input\_size, 1, l.output + offset + j\*l.outputs, 1);

copy\_cpu(part\_input\_size, input + j\*input\_size + part\_input\_size\*l.group\_id, 1, l.output + offset + j\*l.outputs, 1);

}

//offset += input\_size;

offset += part\_input\_size;

}

}

void backward\_route\_layer(const route\_layer l, network\_state state)

{

int i, j;

int offset = 0;

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

int index = l.input\_layers[i];

float \*delta = state.net.layers[index].delta;

int input\_size = l.input\_sizes[i];

int part\_input\_size = input\_size / l.groups;

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

//axpy\_cpu(input\_size, 1, l.delta + offset + j\*l.outputs, 1, delta + j\*input\_size, 1);

axpy\_cpu(part\_input\_size, 1, l.delta + offset + j\*l.outputs, 1, delta + j\*input\_size + part\_input\_size\*l.group\_id, 1);

}

//offset += input\_size;

offset += part\_input\_size;

}

}

#ifdef GPU

void forward\_route\_layer\_gpu(const route\_layer l, network\_state state)

{

int i, j;

int offset = 0;

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

int index = l.input\_layers[i];

float \*input = state.net.layers[index].output\_gpu;

int input\_size = l.input\_sizes[i];

int part\_input\_size = input\_size / l.groups;

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

//copy\_ongpu(input\_size, input + j\*input\_size, 1, l.output\_gpu + offset + j\*l.outputs, 1);

//simple\_copy\_ongpu(input\_size, input + j\*input\_size, l.output\_gpu + offset + j\*l.outputs);

simple\_copy\_ongpu(part\_input\_size, input + j\*input\_size + part\_input\_size\*l.group\_id, l.output\_gpu + offset + j\*l.outputs);

}

//offset += input\_size;

offset += part\_input\_size;

}

}

void backward\_route\_layer\_gpu(const route\_layer l, network\_state state)

{

int i, j;

int offset = 0;

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

int index = l.input\_layers[i];

float \*delta = state.net.layers[index].delta\_gpu;

int input\_size = l.input\_sizes[i];

int part\_input\_size = input\_size / l.groups;

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

//axpy\_ongpu(input\_size, 1, l.delta\_gpu + offset + j\*l.outputs, 1, delta + j\*input\_size, 1);

axpy\_ongpu(part\_input\_size, 1, l.delta\_gpu + offset + j\*l.outputs, 1, delta + j\*input\_size + part\_input\_size\*l.group\_id, 1);

}

//offset += input\_size;

offset += part\_input\_size;

}

}

#endif