#include "dropout\_layer.h"

#include "utils.h"

#include "dark\_cuda.h"

#include <stdlib.h>

#include <stdio.h>

dropout\_layer make\_dropout\_layer(int batch, int inputs, float probability, int dropblock, float dropblock\_size\_rel, int dropblock\_size\_abs, int w, int h, int c)

{

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

l.type = DROPOUT;

l.probability = probability;

l.dropblock = dropblock;

l.dropblock\_size\_rel = dropblock\_size\_rel;

l.dropblock\_size\_abs = dropblock\_size\_abs;

if (l.dropblock) {

l.out\_w = l.w = w;

l.out\_h = l.h = h;

l.out\_c = l.c = c;

if (l.w <= 0 || l.h <= 0 || l.c <= 0) {

printf(" Error: DropBlock - there must be positive values for: l.w=%d, l.h=%d, l.c=%d \n", l.w, l.h, l.c);

exit(0);

}

}

l.inputs = inputs;

l.outputs = inputs;

l.batch = batch;

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

l.scale = 1./(1.0 - probability);

l.forward = forward\_dropout\_layer;

l.backward = backward\_dropout\_layer;

#ifdef GPU

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

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

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

if (l.dropblock) {

l.drop\_blocks\_scale = cuda\_make\_array\_pinned(l.rand, l.batch);

l.drop\_blocks\_scale\_gpu = cuda\_make\_array(l.rand, l.batch);

}

#endif

if (l.dropblock) {

if(l.dropblock\_size\_abs) fprintf(stderr, "dropblock p = %.3f l.dropblock\_size\_abs = %d %4d -> %4d\n", probability, l.dropblock\_size\_abs, inputs, inputs);

else fprintf(stderr, "dropblock p = %.3f l.dropblock\_size\_rel = %.2f %4d -> %4d\n", probability, l.dropblock\_size\_rel, inputs, inputs);

}

else fprintf(stderr, "dropout p = %.3f %4d -> %4d\n", probability, inputs, inputs);

return l;

}

void resize\_dropout\_layer(dropout\_layer \*l, int inputs)

{

l->inputs = l->outputs = inputs;

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

#ifdef GPU

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

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

if (l->dropblock) {

cudaFreeHost(l->drop\_blocks\_scale);

l->drop\_blocks\_scale = cuda\_make\_array\_pinned(l->rand, l->batch);

cuda\_free(l->drop\_blocks\_scale\_gpu);

l->drop\_blocks\_scale\_gpu = cuda\_make\_array(l->rand, l->batch);

}

#endif

}

void forward\_dropout\_layer(dropout\_layer l, network\_state state)

{

int i;

if (!state.train) return;

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

float r = rand\_uniform(0, 1);

l.rand[i] = r;

if(r < l.probability) state.input[i] = 0;

else state.input[i] \*= l.scale;

}

}

void backward\_dropout\_layer(dropout\_layer l, network\_state state)

{

int i;

if(!state.delta) return;

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

float r = l.rand[i];

if(r < l.probability) state.delta[i] = 0;

else state.delta[i] \*= l.scale;

}

}