#include "gru\_layer.h"

#include "connected\_layer.h"

#include "utils.h"

#include "dark\_cuda.h"

#include "blas.h"

#include "gemm.h"

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

static void increment\_layer(layer \*l, int steps)

{

int num = l->outputs\*l->batch\*steps;

l->output += num;

l->delta += num;

l->x += num;

l->x\_norm += num;

#ifdef GPU

l->output\_gpu += num;

l->delta\_gpu += num;

l->x\_gpu += num;

l->x\_norm\_gpu += num;

#endif

}

layer make\_gru\_layer(int batch, int inputs, int outputs, int steps, int batch\_normalize)

{

fprintf(stderr, "GRU Layer: %d inputs, %d outputs\n", inputs, outputs);

batch = batch / steps;

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

l.batch = batch;

l.type = GRU;

l.steps = steps;

l.inputs = inputs;

l.input\_z\_layer = (layer\*)xcalloc(1,sizeof(layer));

fprintf(stderr, "\t\t");

\*(l.input\_z\_layer) = make\_connected\_layer(batch, steps, inputs, outputs, LINEAR, batch\_normalize);

l.input\_z\_layer->batch = batch;

l.state\_z\_layer = (layer\*)xcalloc(1,sizeof(layer));

fprintf(stderr, "\t\t");

\*(l.state\_z\_layer) = make\_connected\_layer(batch, steps, outputs, outputs, LINEAR, batch\_normalize);

l.state\_z\_layer->batch = batch;

l.input\_r\_layer = (layer\*)xcalloc(1,sizeof(layer));

fprintf(stderr, "\t\t");

\*(l.input\_r\_layer) = make\_connected\_layer(batch, steps, inputs, outputs, LINEAR, batch\_normalize);

l.input\_r\_layer->batch = batch;

l.state\_r\_layer = (layer\*)xcalloc(1,sizeof(layer));

fprintf(stderr, "\t\t");

\*(l.state\_r\_layer) = make\_connected\_layer(batch, steps, outputs, outputs, LINEAR, batch\_normalize);

l.state\_r\_layer->batch = batch;

l.input\_h\_layer = (layer\*)xcalloc(1,sizeof(layer));

fprintf(stderr, "\t\t");

\*(l.input\_h\_layer) = make\_connected\_layer(batch, steps, inputs, outputs, LINEAR, batch\_normalize);

l.input\_h\_layer->batch = batch;

l.state\_h\_layer = (layer\*)xcalloc(1,sizeof(layer));

fprintf(stderr, "\t\t");

\*(l.state\_h\_layer) = make\_connected\_layer(batch, steps, outputs, outputs, LINEAR, batch\_normalize);

l.state\_h\_layer->batch = batch;

l.batch\_normalize = batch\_normalize;

l.outputs = outputs;

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

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

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

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

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

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

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

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

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

l.forward = forward\_gru\_layer;

l.backward = backward\_gru\_layer;

l.update = update\_gru\_layer;

#ifdef GPU

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

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

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

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

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

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

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

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

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

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

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

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

#endif

return l;

}

void update\_gru\_layer(layer l, int batch, float learning\_rate, float momentum, float decay)

{

update\_connected\_layer(\*(l.input\_layer), batch, learning\_rate, momentum, decay);

update\_connected\_layer(\*(l.self\_layer), batch, learning\_rate, momentum, decay);

update\_connected\_layer(\*(l.output\_layer), batch, learning\_rate, momentum, decay);

}

void forward\_gru\_layer(layer l, network\_state state)

{

network\_state s = {0};

s.train = state.train;

s.workspace = state.workspace;

int i;

layer input\_z\_layer = \*(l.input\_z\_layer);

layer input\_r\_layer = \*(l.input\_r\_layer);

layer input\_h\_layer = \*(l.input\_h\_layer);

layer state\_z\_layer = \*(l.state\_z\_layer);

layer state\_r\_layer = \*(l.state\_r\_layer);

layer state\_h\_layer = \*(l.state\_h\_layer);

fill\_cpu(l.outputs \* l.batch \* l.steps, 0, input\_z\_layer.delta, 1);

fill\_cpu(l.outputs \* l.batch \* l.steps, 0, input\_r\_layer.delta, 1);

fill\_cpu(l.outputs \* l.batch \* l.steps, 0, input\_h\_layer.delta, 1);

fill\_cpu(l.outputs \* l.batch \* l.steps, 0, state\_z\_layer.delta, 1);

fill\_cpu(l.outputs \* l.batch \* l.steps, 0, state\_r\_layer.delta, 1);

fill\_cpu(l.outputs \* l.batch \* l.steps, 0, state\_h\_layer.delta, 1);

if(state.train) {

fill\_cpu(l.outputs \* l.batch \* l.steps, 0, l.delta, 1);

copy\_cpu(l.outputs\*l.batch, l.state, 1, l.prev\_state, 1);

}

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

s.input = l.state;

forward\_connected\_layer(state\_z\_layer, s);

forward\_connected\_layer(state\_r\_layer, s);

s.input = state.input;

forward\_connected\_layer(input\_z\_layer, s);

forward\_connected\_layer(input\_r\_layer, s);

forward\_connected\_layer(input\_h\_layer, s);

copy\_cpu(l.outputs\*l.batch, input\_z\_layer.output, 1, l.z\_cpu, 1);

axpy\_cpu(l.outputs\*l.batch, 1, state\_z\_layer.output, 1, l.z\_cpu, 1);

copy\_cpu(l.outputs\*l.batch, input\_r\_layer.output, 1, l.r\_cpu, 1);

axpy\_cpu(l.outputs\*l.batch, 1, state\_r\_layer.output, 1, l.r\_cpu, 1);

activate\_array(l.z\_cpu, l.outputs\*l.batch, LOGISTIC);

activate\_array(l.r\_cpu, l.outputs\*l.batch, LOGISTIC);

copy\_cpu(l.outputs\*l.batch, l.state, 1, l.forgot\_state, 1);

mul\_cpu(l.outputs\*l.batch, l.r\_cpu, 1, l.forgot\_state, 1);

s.input = l.forgot\_state;

forward\_connected\_layer(state\_h\_layer, s);

copy\_cpu(l.outputs\*l.batch, input\_h\_layer.output, 1, l.h\_cpu, 1);

axpy\_cpu(l.outputs\*l.batch, 1, state\_h\_layer.output, 1, l.h\_cpu, 1);

#ifdef USET

activate\_array(l.h\_cpu, l.outputs\*l.batch, TANH);

#else

activate\_array(l.h\_cpu, l.outputs\*l.batch, LOGISTIC);

#endif

weighted\_sum\_cpu(l.state, l.h\_cpu, l.z\_cpu, l.outputs\*l.batch, l.output);

copy\_cpu(l.outputs\*l.batch, l.output, 1, l.state, 1);

state.input += l.inputs\*l.batch;

l.output += l.outputs\*l.batch;

increment\_layer(&input\_z\_layer, 1);

increment\_layer(&input\_r\_layer, 1);

increment\_layer(&input\_h\_layer, 1);

increment\_layer(&state\_z\_layer, 1);

increment\_layer(&state\_r\_layer, 1);

increment\_layer(&state\_h\_layer, 1);

}

}

void backward\_gru\_layer(layer l, network\_state state)

{

}

#ifdef GPU

void pull\_gru\_layer(layer l)

{

}

void push\_gru\_layer(layer l)

{

}

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

{

update\_connected\_layer\_gpu(\*(l.input\_r\_layer), batch, learning\_rate, momentum, decay, loss\_scale);

update\_connected\_layer\_gpu(\*(l.input\_z\_layer), batch, learning\_rate, momentum, decay, loss\_scale);

update\_connected\_layer\_gpu(\*(l.input\_h\_layer), batch, learning\_rate, momentum, decay, loss\_scale);

update\_connected\_layer\_gpu(\*(l.state\_r\_layer), batch, learning\_rate, momentum, decay, loss\_scale);

update\_connected\_layer\_gpu(\*(l.state\_z\_layer), batch, learning\_rate, momentum, decay, loss\_scale);

update\_connected\_layer\_gpu(\*(l.state\_h\_layer), batch, learning\_rate, momentum, decay, loss\_scale);

}

void forward\_gru\_layer\_gpu(layer l, network\_state state)

{

network\_state s = {0};

s.train = state.train;

s.workspace = state.workspace;

int i;

layer input\_z\_layer = \*(l.input\_z\_layer);

layer input\_r\_layer = \*(l.input\_r\_layer);

layer input\_h\_layer = \*(l.input\_h\_layer);

layer state\_z\_layer = \*(l.state\_z\_layer);

layer state\_r\_layer = \*(l.state\_r\_layer);

layer state\_h\_layer = \*(l.state\_h\_layer);

fill\_ongpu(l.outputs \* l.batch \* l.steps, 0, input\_z\_layer.delta\_gpu, 1);

fill\_ongpu(l.outputs \* l.batch \* l.steps, 0, input\_r\_layer.delta\_gpu, 1);

fill\_ongpu(l.outputs \* l.batch \* l.steps, 0, input\_h\_layer.delta\_gpu, 1);

fill\_ongpu(l.outputs \* l.batch \* l.steps, 0, state\_z\_layer.delta\_gpu, 1);

fill\_ongpu(l.outputs \* l.batch \* l.steps, 0, state\_r\_layer.delta\_gpu, 1);

fill\_ongpu(l.outputs \* l.batch \* l.steps, 0, state\_h\_layer.delta\_gpu, 1);

if(state.train) {

fill\_ongpu(l.outputs \* l.batch \* l.steps, 0, l.delta\_gpu, 1);

copy\_ongpu(l.outputs\*l.batch, l.state\_gpu, 1, l.prev\_state\_gpu, 1);

}

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

s.input = l.state\_gpu;

forward\_connected\_layer\_gpu(state\_z\_layer, s);

forward\_connected\_layer\_gpu(state\_r\_layer, s);

s.input = state.input;

forward\_connected\_layer\_gpu(input\_z\_layer, s);

forward\_connected\_layer\_gpu(input\_r\_layer, s);

forward\_connected\_layer\_gpu(input\_h\_layer, s);

copy\_ongpu(l.outputs\*l.batch, input\_z\_layer.output\_gpu, 1, l.z\_gpu, 1);

axpy\_ongpu(l.outputs\*l.batch, 1, state\_z\_layer.output\_gpu, 1, l.z\_gpu, 1);

copy\_ongpu(l.outputs\*l.batch, input\_r\_layer.output\_gpu, 1, l.r\_gpu, 1);

axpy\_ongpu(l.outputs\*l.batch, 1, state\_r\_layer.output\_gpu, 1, l.r\_gpu, 1);

activate\_array\_ongpu(l.z\_gpu, l.outputs\*l.batch, LOGISTIC);

activate\_array\_ongpu(l.r\_gpu, l.outputs\*l.batch, LOGISTIC);

copy\_ongpu(l.outputs\*l.batch, l.state\_gpu, 1, l.forgot\_state\_gpu, 1);

mul\_ongpu(l.outputs\*l.batch, l.r\_gpu, 1, l.forgot\_state\_gpu, 1);

s.input = l.forgot\_state\_gpu;

forward\_connected\_layer\_gpu(state\_h\_layer, s);

copy\_ongpu(l.outputs\*l.batch, input\_h\_layer.output\_gpu, 1, l.h\_gpu, 1);

axpy\_ongpu(l.outputs\*l.batch, 1, state\_h\_layer.output\_gpu, 1, l.h\_gpu, 1);

#ifdef USET

activate\_array\_ongpu(l.h\_gpu, l.outputs\*l.batch, TANH);

#else

activate\_array\_ongpu(l.h\_gpu, l.outputs\*l.batch, LOGISTIC);

#endif

weighted\_sum\_gpu(l.state\_gpu, l.h\_gpu, l.z\_gpu, l.outputs\*l.batch, l.output\_gpu);

copy\_ongpu(l.outputs\*l.batch, l.output\_gpu, 1, l.state\_gpu, 1);

state.input += l.inputs\*l.batch;

l.output\_gpu += l.outputs\*l.batch;

increment\_layer(&input\_z\_layer, 1);

increment\_layer(&input\_r\_layer, 1);

increment\_layer(&input\_h\_layer, 1);

increment\_layer(&state\_z\_layer, 1);

increment\_layer(&state\_r\_layer, 1);

increment\_layer(&state\_h\_layer, 1);

}

}

void backward\_gru\_layer\_gpu(layer l, network\_state state)

{

network\_state s = {0};

s.train = state.train;

s.workspace = state.workspace;

int i;

layer input\_z\_layer = \*(l.input\_z\_layer);

layer input\_r\_layer = \*(l.input\_r\_layer);

layer input\_h\_layer = \*(l.input\_h\_layer);

layer state\_z\_layer = \*(l.state\_z\_layer);

layer state\_r\_layer = \*(l.state\_r\_layer);

layer state\_h\_layer = \*(l.state\_h\_layer);

increment\_layer(&input\_z\_layer, l.steps - 1);

increment\_layer(&input\_r\_layer, l.steps - 1);

increment\_layer(&input\_h\_layer, l.steps - 1);

increment\_layer(&state\_z\_layer, l.steps - 1);

increment\_layer(&state\_r\_layer, l.steps - 1);

increment\_layer(&state\_h\_layer, l.steps - 1);

state.input += l.inputs\*l.batch\*(l.steps-1);

if(state.delta) state.delta += l.inputs\*l.batch\*(l.steps-1);

l.output\_gpu += l.outputs\*l.batch\*(l.steps-1);

l.delta\_gpu += l.outputs\*l.batch\*(l.steps-1);

for (i = l.steps-1; i >= 0; --i) {

if(i != 0) copy\_ongpu(l.outputs\*l.batch, l.output\_gpu - l.outputs\*l.batch, 1, l.prev\_state\_gpu, 1);

float \*prev\_delta\_gpu = (i == 0) ? 0 : l.delta\_gpu - l.outputs\*l.batch;

copy\_ongpu(l.outputs\*l.batch, input\_z\_layer.output\_gpu, 1, l.z\_gpu, 1);

axpy\_ongpu(l.outputs\*l.batch, 1, state\_z\_layer.output\_gpu, 1, l.z\_gpu, 1);

copy\_ongpu(l.outputs\*l.batch, input\_r\_layer.output\_gpu, 1, l.r\_gpu, 1);

axpy\_ongpu(l.outputs\*l.batch, 1, state\_r\_layer.output\_gpu, 1, l.r\_gpu, 1);

activate\_array\_ongpu(l.z\_gpu, l.outputs\*l.batch, LOGISTIC);

activate\_array\_ongpu(l.r\_gpu, l.outputs\*l.batch, LOGISTIC);

copy\_ongpu(l.outputs\*l.batch, input\_h\_layer.output\_gpu, 1, l.h\_gpu, 1);

axpy\_ongpu(l.outputs\*l.batch, 1, state\_h\_layer.output\_gpu, 1, l.h\_gpu, 1);

#ifdef USET

activate\_array\_ongpu(l.h\_gpu, l.outputs\*l.batch, TANH);

#else

activate\_array\_ongpu(l.h\_gpu, l.outputs\*l.batch, LOGISTIC);

#endif

weighted\_delta\_gpu(l.prev\_state\_gpu, l.h\_gpu, l.z\_gpu, prev\_delta\_gpu, input\_h\_layer.delta\_gpu, input\_z\_layer.delta\_gpu, l.outputs\*l.batch, l.delta\_gpu);

#ifdef USET

gradient\_array\_ongpu(l.h\_gpu, l.outputs\*l.batch, TANH, input\_h\_layer.delta\_gpu);

#else

gradient\_array\_ongpu(l.h\_gpu, l.outputs\*l.batch, LOGISTIC, input\_h\_layer.delta\_gpu);

#endif

copy\_ongpu(l.outputs\*l.batch, input\_h\_layer.delta\_gpu, 1, state\_h\_layer.delta\_gpu, 1);

copy\_ongpu(l.outputs\*l.batch, l.prev\_state\_gpu, 1, l.forgot\_state\_gpu, 1);

mul\_ongpu(l.outputs\*l.batch, l.r\_gpu, 1, l.forgot\_state\_gpu, 1);

fill\_ongpu(l.outputs\*l.batch, 0, l.forgot\_delta\_gpu, 1);

s.input = l.forgot\_state\_gpu;

s.delta = l.forgot\_delta\_gpu;

backward\_connected\_layer\_gpu(state\_h\_layer, s);

if(prev\_delta\_gpu) mult\_add\_into\_gpu(l.outputs\*l.batch, l.forgot\_delta\_gpu, l.r\_gpu, prev\_delta\_gpu);

mult\_add\_into\_gpu(l.outputs\*l.batch, l.forgot\_delta\_gpu, l.prev\_state\_gpu, input\_r\_layer.delta\_gpu);

gradient\_array\_ongpu(l.r\_gpu, l.outputs\*l.batch, LOGISTIC, input\_r\_layer.delta\_gpu);

copy\_ongpu(l.outputs\*l.batch, input\_r\_layer.delta\_gpu, 1, state\_r\_layer.delta\_gpu, 1);

gradient\_array\_ongpu(l.z\_gpu, l.outputs\*l.batch, LOGISTIC, input\_z\_layer.delta\_gpu);

copy\_ongpu(l.outputs\*l.batch, input\_z\_layer.delta\_gpu, 1, state\_z\_layer.delta\_gpu, 1);

s.input = l.prev\_state\_gpu;

s.delta = prev\_delta\_gpu;

backward\_connected\_layer\_gpu(state\_r\_layer, s);

backward\_connected\_layer\_gpu(state\_z\_layer, s);

s.input = state.input;

s.delta = state.delta;

backward\_connected\_layer\_gpu(input\_h\_layer, s);

backward\_connected\_layer\_gpu(input\_r\_layer, s);

backward\_connected\_layer\_gpu(input\_z\_layer, s);

state.input -= l.inputs\*l.batch;

if(state.delta) state.delta -= l.inputs\*l.batch;

l.output\_gpu -= l.outputs\*l.batch;

l.delta\_gpu -= l.outputs\*l.batch;

increment\_layer(&input\_z\_layer, -1);

increment\_layer(&input\_r\_layer, -1);

increment\_layer(&input\_h\_layer, -1);

increment\_layer(&state\_z\_layer, -1);

increment\_layer(&state\_r\_layer, -1);

increment\_layer(&state\_h\_layer, -1);

}

}

#endif