#include "lstm\_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\_lstm\_layer(int batch, int inputs, int outputs, int steps, int batch\_normalize)

{

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

batch = batch / steps;

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

l.batch = batch;

l.type = LSTM;

l.steps = steps;

l.inputs = inputs;

l.out\_w = 1;

l.out\_h = 1;

l.out\_c = outputs;

l.uf = (layer\*)xcalloc(1, sizeof(layer));

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

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

l.uf->batch = batch;

if (l.workspace\_size < l.uf->workspace\_size) l.workspace\_size = l.uf->workspace\_size;

l.ui = (layer\*)xcalloc(1, sizeof(layer));

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

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

l.ui->batch = batch;

if (l.workspace\_size < l.ui->workspace\_size) l.workspace\_size = l.ui->workspace\_size;

l.ug = (layer\*)xcalloc(1, sizeof(layer));

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

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

l.ug->batch = batch;

if (l.workspace\_size < l.ug->workspace\_size) l.workspace\_size = l.ug->workspace\_size;

l.uo = (layer\*)xcalloc(1, sizeof(layer));

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

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

l.uo->batch = batch;

if (l.workspace\_size < l.uo->workspace\_size) l.workspace\_size = l.uo->workspace\_size;

l.wf = (layer\*)xcalloc(1, sizeof(layer));

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

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

l.wf->batch = batch;

if (l.workspace\_size < l.wf->workspace\_size) l.workspace\_size = l.wf->workspace\_size;

l.wi = (layer\*)xcalloc(1, sizeof(layer));

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

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

l.wi->batch = batch;

if (l.workspace\_size < l.wi->workspace\_size) l.workspace\_size = l.wi->workspace\_size;

l.wg = (layer\*)xcalloc(1, sizeof(layer));

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

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

l.wg->batch = batch;

if (l.workspace\_size < l.wg->workspace\_size) l.workspace\_size = l.wg->workspace\_size;

l.wo = (layer\*)xcalloc(1, sizeof(layer));

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

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

l.wo->batch = batch;

if (l.workspace\_size < l.wo->workspace\_size) l.workspace\_size = l.wo->workspace\_size;

l.batch\_normalize = batch\_normalize;

l.outputs = outputs;

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

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

l.forward = forward\_lstm\_layer;

l.update = update\_lstm\_layer;

l.backward = backward\_lstm\_layer;

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

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

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

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

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

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

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

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

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

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

l.temp2\_cpu = (float\*)xcalloc(batch\*outputs, sizeof(float));

l.temp3\_cpu = (float\*)xcalloc(batch\*outputs, sizeof(float));

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

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

#ifdef GPU

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

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

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

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

l.output\_gpu = cuda\_make\_array(0, batch\*outputs\*steps);

l.delta\_gpu = cuda\_make\_array(0, batch\*l.outputs\*steps);

l.prev\_state\_gpu = cuda\_make\_array(0, batch\*outputs);

l.prev\_cell\_gpu = cuda\_make\_array(0, batch\*outputs);

l.cell\_gpu = cuda\_make\_array(0, batch\*outputs\*steps);

l.f\_gpu = cuda\_make\_array(0, batch\*outputs);

l.i\_gpu = cuda\_make\_array(0, batch\*outputs);

l.g\_gpu = cuda\_make\_array(0, batch\*outputs);

l.o\_gpu = cuda\_make\_array(0, batch\*outputs);

l.c\_gpu = cuda\_make\_array(0, batch\*outputs);

l.h\_gpu = cuda\_make\_array(0, batch\*outputs);

l.temp\_gpu = cuda\_make\_array(0, batch\*outputs);

l.temp2\_gpu = cuda\_make\_array(0, batch\*outputs);

l.temp3\_gpu = cuda\_make\_array(0, batch\*outputs);

l.dc\_gpu = cuda\_make\_array(0, batch\*outputs);

l.dh\_gpu = cuda\_make\_array(0, batch\*outputs);

#ifdef CUDNN

/\*

cudnnSetTensor4dDescriptor(l.wf->dstTensorDesc, CUDNN\_TENSOR\_NCHW, CUDNN\_DATA\_FLOAT, batch, l.wf->out\_c, l.wf->out\_h, l.wf->out\_w);

cudnnSetTensor4dDescriptor(l.wi->dstTensorDesc, CUDNN\_TENSOR\_NCHW, CUDNN\_DATA\_FLOAT, batch, l.wi->out\_c, l.wi->out\_h, l.wi->out\_w);

cudnnSetTensor4dDescriptor(l.wg->dstTensorDesc, CUDNN\_TENSOR\_NCHW, CUDNN\_DATA\_FLOAT, batch, l.wg->out\_c, l.wg->out\_h, l.wg->out\_w);

cudnnSetTensor4dDescriptor(l.wo->dstTensorDesc, CUDNN\_TENSOR\_NCHW, CUDNN\_DATA\_FLOAT, batch, l.wo->out\_c, l.wo->out\_h, l.wo->out\_w);

cudnnSetTensor4dDescriptor(l.uf->dstTensorDesc, CUDNN\_TENSOR\_NCHW, CUDNN\_DATA\_FLOAT, batch, l.uf->out\_c, l.uf->out\_h, l.uf->out\_w);

cudnnSetTensor4dDescriptor(l.ui->dstTensorDesc, CUDNN\_TENSOR\_NCHW, CUDNN\_DATA\_FLOAT, batch, l.ui->out\_c, l.ui->out\_h, l.ui->out\_w);

cudnnSetTensor4dDescriptor(l.ug->dstTensorDesc, CUDNN\_TENSOR\_NCHW, CUDNN\_DATA\_FLOAT, batch, l.ug->out\_c, l.ug->out\_h, l.ug->out\_w);

cudnnSetTensor4dDescriptor(l.uo->dstTensorDesc, CUDNN\_TENSOR\_NCHW, CUDNN\_DATA\_FLOAT, batch, l.uo->out\_c, l.uo->out\_h, l.uo->out\_w);

\*/

#endif

#endif

return l;

}

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

{

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

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

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

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

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

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

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

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

}

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

{

network\_state s = { 0 };

s.train = state.train;

s.workspace = state.workspace;

int i;

layer wf = \*(l.wf);

layer wi = \*(l.wi);

layer wg = \*(l.wg);

layer wo = \*(l.wo);

layer uf = \*(l.uf);

layer ui = \*(l.ui);

layer ug = \*(l.ug);

layer uo = \*(l.uo);

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

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

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

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

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

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

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

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

if (state.train) {

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

}

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

s.input = l.h\_cpu;

forward\_connected\_layer(wf, s);

forward\_connected\_layer(wi, s);

forward\_connected\_layer(wg, s);

forward\_connected\_layer(wo, s);

s.input = state.input;

forward\_connected\_layer(uf, s);

forward\_connected\_layer(ui, s);

forward\_connected\_layer(ug, s);

forward\_connected\_layer(uo, s);

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

axpy\_cpu(l.outputs\*l.batch, 1, uf.output, 1, l.f\_cpu, 1);

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

axpy\_cpu(l.outputs\*l.batch, 1, ui.output, 1, l.i\_cpu, 1);

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

axpy\_cpu(l.outputs\*l.batch, 1, ug.output, 1, l.g\_cpu, 1);

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

axpy\_cpu(l.outputs\*l.batch, 1, uo.output, 1, l.o\_cpu, 1);

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

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

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

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

copy\_cpu(l.outputs\*l.batch, l.i\_cpu, 1, l.temp\_cpu, 1);

mul\_cpu(l.outputs\*l.batch, l.g\_cpu, 1, l.temp\_cpu, 1);

mul\_cpu(l.outputs\*l.batch, l.f\_cpu, 1, l.c\_cpu, 1);

axpy\_cpu(l.outputs\*l.batch, 1, l.temp\_cpu, 1, l.c\_cpu, 1);

copy\_cpu(l.outputs\*l.batch, l.c\_cpu, 1, l.h\_cpu, 1);

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

mul\_cpu(l.outputs\*l.batch, l.o\_cpu, 1, l.h\_cpu, 1);

copy\_cpu(l.outputs\*l.batch, l.c\_cpu, 1, l.cell\_cpu, 1);

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

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

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

l.cell\_cpu += l.outputs\*l.batch;

increment\_layer(&wf, 1);

increment\_layer(&wi, 1);

increment\_layer(&wg, 1);

increment\_layer(&wo, 1);

increment\_layer(&uf, 1);

increment\_layer(&ui, 1);

increment\_layer(&ug, 1);

increment\_layer(&uo, 1);

}

}

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

{

network\_state s = { 0 };

s.train = state.train;

s.workspace = state.workspace;

int i;

layer wf = \*(l.wf);

layer wi = \*(l.wi);

layer wg = \*(l.wg);

layer wo = \*(l.wo);

layer uf = \*(l.uf);

layer ui = \*(l.ui);

layer ug = \*(l.ug);

layer uo = \*(l.uo);

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

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

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

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

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

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

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

increment\_layer(&uo, 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 += l.outputs\*l.batch\*(l.steps - 1);

l.cell\_cpu += l.outputs\*l.batch\*(l.steps - 1);

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

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

if (i != 0) copy\_cpu(l.outputs\*l.batch, l.cell\_cpu - l.outputs\*l.batch, 1, l.prev\_cell\_cpu, 1);

copy\_cpu(l.outputs\*l.batch, l.cell\_cpu, 1, l.c\_cpu, 1);

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

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

l.dh\_cpu = (i == 0) ? 0 : l.delta - l.outputs\*l.batch;

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

axpy\_cpu(l.outputs\*l.batch, 1, uf.output, 1, l.f\_cpu, 1);

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

axpy\_cpu(l.outputs\*l.batch, 1, ui.output, 1, l.i\_cpu, 1);

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

axpy\_cpu(l.outputs\*l.batch, 1, ug.output, 1, l.g\_cpu, 1);

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

axpy\_cpu(l.outputs\*l.batch, 1, uo.output, 1, l.o\_cpu, 1);

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

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

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

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

copy\_cpu(l.outputs\*l.batch, l.delta, 1, l.temp3\_cpu, 1);

copy\_cpu(l.outputs\*l.batch, l.c\_cpu, 1, l.temp\_cpu, 1);

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

copy\_cpu(l.outputs\*l.batch, l.temp3\_cpu, 1, l.temp2\_cpu, 1);

mul\_cpu(l.outputs\*l.batch, l.o\_cpu, 1, l.temp2\_cpu, 1);

gradient\_array(l.temp\_cpu, l.outputs\*l.batch, TANH, l.temp2\_cpu);

axpy\_cpu(l.outputs\*l.batch, 1, l.dc\_cpu, 1, l.temp2\_cpu, 1);

copy\_cpu(l.outputs\*l.batch, l.c\_cpu, 1, l.temp\_cpu, 1);

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

mul\_cpu(l.outputs\*l.batch, l.temp3\_cpu, 1, l.temp\_cpu, 1);

gradient\_array(l.o\_cpu, l.outputs\*l.batch, LOGISTIC, l.temp\_cpu);

copy\_cpu(l.outputs\*l.batch, l.temp\_cpu, 1, wo.delta, 1);

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

s.delta = l.dh\_cpu;

backward\_connected\_layer(wo, s);

copy\_cpu(l.outputs\*l.batch, l.temp\_cpu, 1, uo.delta, 1);

s.input = state.input;

s.delta = state.delta;

backward\_connected\_layer(uo, s);

copy\_cpu(l.outputs\*l.batch, l.temp2\_cpu, 1, l.temp\_cpu, 1);

mul\_cpu(l.outputs\*l.batch, l.i\_cpu, 1, l.temp\_cpu, 1);

gradient\_array(l.g\_cpu, l.outputs\*l.batch, TANH, l.temp\_cpu);

copy\_cpu(l.outputs\*l.batch, l.temp\_cpu, 1, wg.delta, 1);

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

s.delta = l.dh\_cpu;

backward\_connected\_layer(wg, s);

copy\_cpu(l.outputs\*l.batch, l.temp\_cpu, 1, ug.delta, 1);

s.input = state.input;

s.delta = state.delta;

backward\_connected\_layer(ug, s);

copy\_cpu(l.outputs\*l.batch, l.temp2\_cpu, 1, l.temp\_cpu, 1);

mul\_cpu(l.outputs\*l.batch, l.g\_cpu, 1, l.temp\_cpu, 1);

gradient\_array(l.i\_cpu, l.outputs\*l.batch, LOGISTIC, l.temp\_cpu);

copy\_cpu(l.outputs\*l.batch, l.temp\_cpu, 1, wi.delta, 1);

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

s.delta = l.dh\_cpu;

backward\_connected\_layer(wi, s);

copy\_cpu(l.outputs\*l.batch, l.temp\_cpu, 1, ui.delta, 1);

s.input = state.input;

s.delta = state.delta;

backward\_connected\_layer(ui, s);

copy\_cpu(l.outputs\*l.batch, l.temp2\_cpu, 1, l.temp\_cpu, 1);

mul\_cpu(l.outputs\*l.batch, l.prev\_cell\_cpu, 1, l.temp\_cpu, 1);

gradient\_array(l.f\_cpu, l.outputs\*l.batch, LOGISTIC, l.temp\_cpu);

copy\_cpu(l.outputs\*l.batch, l.temp\_cpu, 1, wf.delta, 1);

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

s.delta = l.dh\_cpu;

backward\_connected\_layer(wf, s);

copy\_cpu(l.outputs\*l.batch, l.temp\_cpu, 1, uf.delta, 1);

s.input = state.input;

s.delta = state.delta;

backward\_connected\_layer(uf, s);

copy\_cpu(l.outputs\*l.batch, l.temp2\_cpu, 1, l.temp\_cpu, 1);

mul\_cpu(l.outputs\*l.batch, l.f\_cpu, 1, l.temp\_cpu, 1);

copy\_cpu(l.outputs\*l.batch, l.temp\_cpu, 1, l.dc\_cpu, 1);

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

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

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

l.cell\_cpu -= l.outputs\*l.batch;

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

increment\_layer(&wf, -1);

increment\_layer(&wi, -1);

increment\_layer(&wg, -1);

increment\_layer(&wo, -1);

increment\_layer(&uf, -1);

increment\_layer(&ui, -1);

increment\_layer(&ug, -1);

increment\_layer(&uo, -1);

}

}

#ifdef GPU

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

{

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

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

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

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

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

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

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

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

}

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

{

network\_state s = { 0 };

s.train = state.train;

s.workspace = state.workspace;

int i;

layer wf = \*(l.wf);

layer wi = \*(l.wi);

layer wg = \*(l.wg);

layer wo = \*(l.wo);

layer uf = \*(l.uf);

layer ui = \*(l.ui);

layer ug = \*(l.ug);

layer uo = \*(l.uo);

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

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

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

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

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

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

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

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

if (state.train) {

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

}

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

s.input = l.h\_gpu;

forward\_connected\_layer\_gpu(wf, s);

forward\_connected\_layer\_gpu(wi, s);

forward\_connected\_layer\_gpu(wg, s);

forward\_connected\_layer\_gpu(wo, s);

s.input = state.input;

forward\_connected\_layer\_gpu(uf, s);

forward\_connected\_layer\_gpu(ui, s);

forward\_connected\_layer\_gpu(ug, s);

forward\_connected\_layer\_gpu(uo, s);

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

axpy\_ongpu(l.outputs\*l.batch, 1, uf.output\_gpu, 1, l.f\_gpu, 1);

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

axpy\_ongpu(l.outputs\*l.batch, 1, ui.output\_gpu, 1, l.i\_gpu, 1);

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

axpy\_ongpu(l.outputs\*l.batch, 1, ug.output\_gpu, 1, l.g\_gpu, 1);

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

axpy\_ongpu(l.outputs\*l.batch, 1, uo.output\_gpu, 1, l.o\_gpu, 1);

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

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

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

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

copy\_ongpu(l.outputs\*l.batch, l.i\_gpu, 1, l.temp\_gpu, 1);

mul\_ongpu(l.outputs\*l.batch, l.g\_gpu, 1, l.temp\_gpu, 1);

mul\_ongpu(l.outputs\*l.batch, l.f\_gpu, 1, l.c\_gpu, 1);

axpy\_ongpu(l.outputs\*l.batch, 1, l.temp\_gpu, 1, l.c\_gpu, 1);

copy\_ongpu(l.outputs\*l.batch, l.c\_gpu, 1, l.h\_gpu, 1);

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

mul\_ongpu(l.outputs\*l.batch, l.o\_gpu, 1, l.h\_gpu, 1);

copy\_ongpu(l.outputs\*l.batch, l.c\_gpu, 1, l.cell\_gpu, 1);

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

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

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

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

increment\_layer(&wf, 1);

increment\_layer(&wi, 1);

increment\_layer(&wg, 1);

increment\_layer(&wo, 1);

increment\_layer(&uf, 1);

increment\_layer(&ui, 1);

increment\_layer(&ug, 1);

increment\_layer(&uo, 1);

}

}

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

{

network\_state s = { 0 };

s.train = state.train;

s.workspace = state.workspace;

int i;

layer wf = \*(l.wf);

layer wi = \*(l.wi);

layer wg = \*(l.wg);

layer wo = \*(l.wo);

layer uf = \*(l.uf);

layer ui = \*(l.ui);

layer ug = \*(l.ug);

layer uo = \*(l.uo);

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

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

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

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

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

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

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

increment\_layer(&uo, 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.cell\_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.cell\_gpu - l.outputs\*l.batch, 1, l.prev\_cell\_gpu, 1);

copy\_ongpu(l.outputs\*l.batch, l.cell\_gpu, 1, l.c\_gpu, 1);

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

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

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

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

axpy\_ongpu(l.outputs\*l.batch, 1, uf.output\_gpu, 1, l.f\_gpu, 1);

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

axpy\_ongpu(l.outputs\*l.batch, 1, ui.output\_gpu, 1, l.i\_gpu, 1);

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

axpy\_ongpu(l.outputs\*l.batch, 1, ug.output\_gpu, 1, l.g\_gpu, 1);

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

axpy\_ongpu(l.outputs\*l.batch, 1, uo.output\_gpu, 1, l.o\_gpu, 1);

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

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

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

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

copy\_ongpu(l.outputs\*l.batch, l.delta\_gpu, 1, l.temp3\_gpu, 1);

copy\_ongpu(l.outputs\*l.batch, l.c\_gpu, 1, l.temp\_gpu, 1);

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

copy\_ongpu(l.outputs\*l.batch, l.temp3\_gpu, 1, l.temp2\_gpu, 1);

mul\_ongpu(l.outputs\*l.batch, l.o\_gpu, 1, l.temp2\_gpu, 1);

gradient\_array\_ongpu(l.temp\_gpu, l.outputs\*l.batch, TANH, l.temp2\_gpu);

axpy\_ongpu(l.outputs\*l.batch, 1, l.dc\_gpu, 1, l.temp2\_gpu, 1);

copy\_ongpu(l.outputs\*l.batch, l.c\_gpu, 1, l.temp\_gpu, 1);

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

mul\_ongpu(l.outputs\*l.batch, l.temp3\_gpu, 1, l.temp\_gpu, 1);

gradient\_array\_ongpu(l.o\_gpu, l.outputs\*l.batch, LOGISTIC, l.temp\_gpu);

copy\_ongpu(l.outputs\*l.batch, l.temp\_gpu, 1, wo.delta\_gpu, 1);

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

s.delta = l.dh\_gpu;

backward\_connected\_layer\_gpu(wo, s);

copy\_ongpu(l.outputs\*l.batch, l.temp\_gpu, 1, uo.delta\_gpu, 1);

s.input = state.input;

s.delta = state.delta;

backward\_connected\_layer\_gpu(uo, s);

copy\_ongpu(l.outputs\*l.batch, l.temp2\_gpu, 1, l.temp\_gpu, 1);

mul\_ongpu(l.outputs\*l.batch, l.i\_gpu, 1, l.temp\_gpu, 1);

gradient\_array\_ongpu(l.g\_gpu, l.outputs\*l.batch, TANH, l.temp\_gpu);

copy\_ongpu(l.outputs\*l.batch, l.temp\_gpu, 1, wg.delta\_gpu, 1);

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

s.delta = l.dh\_gpu;

backward\_connected\_layer\_gpu(wg, s);

copy\_ongpu(l.outputs\*l.batch, l.temp\_gpu, 1, ug.delta\_gpu, 1);

s.input = state.input;

s.delta = state.delta;

backward\_connected\_layer\_gpu(ug, s);

copy\_ongpu(l.outputs\*l.batch, l.temp2\_gpu, 1, l.temp\_gpu, 1);

mul\_ongpu(l.outputs\*l.batch, l.g\_gpu, 1, l.temp\_gpu, 1);

gradient\_array\_ongpu(l.i\_gpu, l.outputs\*l.batch, LOGISTIC, l.temp\_gpu);

copy\_ongpu(l.outputs\*l.batch, l.temp\_gpu, 1, wi.delta\_gpu, 1);

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

s.delta = l.dh\_gpu;

backward\_connected\_layer\_gpu(wi, s);

copy\_ongpu(l.outputs\*l.batch, l.temp\_gpu, 1, ui.delta\_gpu, 1);

s.input = state.input;

s.delta = state.delta;

backward\_connected\_layer\_gpu(ui, s);

copy\_ongpu(l.outputs\*l.batch, l.temp2\_gpu, 1, l.temp\_gpu, 1);

mul\_ongpu(l.outputs\*l.batch, l.prev\_cell\_gpu, 1, l.temp\_gpu, 1);

gradient\_array\_ongpu(l.f\_gpu, l.outputs\*l.batch, LOGISTIC, l.temp\_gpu);

copy\_ongpu(l.outputs\*l.batch, l.temp\_gpu, 1, wf.delta\_gpu, 1);

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

s.delta = l.dh\_gpu;

backward\_connected\_layer\_gpu(wf, s);

copy\_ongpu(l.outputs\*l.batch, l.temp\_gpu, 1, uf.delta\_gpu, 1);

s.input = state.input;

s.delta = state.delta;

backward\_connected\_layer\_gpu(uf, s);

copy\_ongpu(l.outputs\*l.batch, l.temp2\_gpu, 1, l.temp\_gpu, 1);

mul\_ongpu(l.outputs\*l.batch, l.f\_gpu, 1, l.temp\_gpu, 1);

copy\_ongpu(l.outputs\*l.batch, l.temp\_gpu, 1, l.dc\_gpu, 1);

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

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

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

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

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

increment\_layer(&wf, -1);

increment\_layer(&wi, -1);

increment\_layer(&wg, -1);

increment\_layer(&wo, -1);

increment\_layer(&uf, -1);

increment\_layer(&ui, -1);

increment\_layer(&ug, -1);

increment\_layer(&uo, -1);

}

}

#endif