#include "rnn\_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\_rnn\_layer(int batch, int inputs, int hidden, int outputs, int steps, ACTIVATION activation, int batch\_normalize, int log)

{

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

batch = batch / steps;

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

l.batch = batch;

l.type = RNN;

l.steps = steps;

l.hidden = hidden;

l.inputs = inputs;

l.out\_w = 1;

l.out\_h = 1;

l.out\_c = outputs;

l.state = (float\*)xcalloc(batch \* hidden \* (steps + 1), sizeof(float));

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

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

\*(l.input\_layer) = make\_connected\_layer(batch, steps, inputs, hidden, activation, batch\_normalize);

l.input\_layer->batch = batch;

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

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

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

\*(l.self\_layer) = make\_connected\_layer(batch, steps, hidden, hidden, (log==2)?LOGGY:(log==1?LOGISTIC:activation), batch\_normalize);

l.self\_layer->batch = batch;

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

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

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

\*(l.output\_layer) = make\_connected\_layer(batch, steps, hidden, outputs, activation, batch\_normalize);

l.output\_layer->batch = batch;

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

l.outputs = outputs;

l.output = l.output\_layer->output;

l.delta = l.output\_layer->delta;

l.forward = forward\_rnn\_layer;

l.backward = backward\_rnn\_layer;

l.update = update\_rnn\_layer;

#ifdef GPU

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

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

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

l.state\_gpu = cuda\_make\_array(l.state, batch\*hidden\*(steps+1));

l.output\_gpu = l.output\_layer->output\_gpu;

l.delta\_gpu = l.output\_layer->delta\_gpu;

#endif

return l;

}

void update\_rnn\_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\_rnn\_layer(layer l, network\_state state)

{

network\_state s = {0};

s.train = state.train;

s.workspace = state.workspace;

int i;

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

layer self\_layer = \*(l.self\_layer);

layer output\_layer = \*(l.output\_layer);

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

fill\_cpu(l.hidden \* l.batch \* l.steps, 0, self\_layer.delta, 1);

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

if(state.train) fill\_cpu(l.hidden \* l.batch, 0, l.state, 1);

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

s.input = state.input;

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

s.input = l.state;

forward\_connected\_layer(self\_layer, s);

float \*old\_state = l.state;

if(state.train) l.state += l.hidden\*l.batch;

if(l.shortcut){

copy\_cpu(l.hidden \* l.batch, old\_state, 1, l.state, 1);

}else{

fill\_cpu(l.hidden \* l.batch, 0, l.state, 1);

}

axpy\_cpu(l.hidden \* l.batch, 1, input\_layer.output, 1, l.state, 1);

axpy\_cpu(l.hidden \* l.batch, 1, self\_layer.output, 1, l.state, 1);

s.input = l.state;

forward\_connected\_layer(output\_layer, s);

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

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

increment\_layer(&self\_layer, 1);

increment\_layer(&output\_layer, 1);

}

}

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

{

network\_state s = {0};

s.train = state.train;

s.workspace = state.workspace;

int i;

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

layer self\_layer = \*(l.self\_layer);

layer output\_layer = \*(l.output\_layer);

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

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

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

l.state += l.hidden\*l.batch\*l.steps;

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

copy\_cpu(l.hidden \* l.batch, input\_layer.output, 1, l.state, 1);

axpy\_cpu(l.hidden \* l.batch, 1, self\_layer.output, 1, l.state, 1);

s.input = l.state;

s.delta = self\_layer.delta;

backward\_connected\_layer(output\_layer, s);

l.state -= l.hidden\*l.batch;

/\*

if(i > 0){

copy\_cpu(l.hidden \* l.batch, input\_layer.output - l.hidden\*l.batch, 1, l.state, 1);

axpy\_cpu(l.hidden \* l.batch, 1, self\_layer.output - l.hidden\*l.batch, 1, l.state, 1);

}else{

fill\_cpu(l.hidden \* l.batch, 0, l.state, 1);

}

\*/

s.input = l.state;

s.delta = self\_layer.delta - l.hidden\*l.batch;

if (i == 0) s.delta = 0;

backward\_connected\_layer(self\_layer, s);

copy\_cpu(l.hidden\*l.batch, self\_layer.delta, 1, input\_layer.delta, 1);

if (i > 0 && l.shortcut) axpy\_cpu(l.hidden\*l.batch, 1, self\_layer.delta, 1, self\_layer.delta - l.hidden\*l.batch, 1);

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

if(state.delta) s.delta = state.delta + i\*l.inputs\*l.batch;

else s.delta = 0;

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

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

increment\_layer(&self\_layer, -1);

increment\_layer(&output\_layer, -1);

}

}

#ifdef GPU

void pull\_rnn\_layer(layer l)

{

pull\_connected\_layer(\*(l.input\_layer));

pull\_connected\_layer(\*(l.self\_layer));

pull\_connected\_layer(\*(l.output\_layer));

}

void push\_rnn\_layer(layer l)

{

push\_connected\_layer(\*(l.input\_layer));

push\_connected\_layer(\*(l.self\_layer));

push\_connected\_layer(\*(l.output\_layer));

}

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

{

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

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

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

}

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

{

network\_state s = {0};

s.train = state.train;

s.workspace = state.workspace;

int i;

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

layer self\_layer = \*(l.self\_layer);

layer output\_layer = \*(l.output\_layer);

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

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

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

if(state.train) fill\_ongpu(l.hidden \* l.batch, 0, l.state\_gpu, 1);

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

s.input = state.input;

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

s.input = l.state\_gpu;

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

float \*old\_state = l.state\_gpu;

if(state.train) l.state\_gpu += l.hidden\*l.batch;

if(l.shortcut){

copy\_ongpu(l.hidden \* l.batch, old\_state, 1, l.state\_gpu, 1);

}else{

fill\_ongpu(l.hidden \* l.batch, 0, l.state\_gpu, 1);

}

axpy\_ongpu(l.hidden \* l.batch, 1, input\_layer.output\_gpu, 1, l.state\_gpu, 1);

axpy\_ongpu(l.hidden \* l.batch, 1, self\_layer.output\_gpu, 1, l.state\_gpu, 1);

s.input = l.state\_gpu;

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

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

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

increment\_layer(&self\_layer, 1);

increment\_layer(&output\_layer, 1);

}

}

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

{

network\_state s = {0};

s.train = state.train;

s.workspace = state.workspace;

int i;

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

layer self\_layer = \*(l.self\_layer);

layer output\_layer = \*(l.output\_layer);

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

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

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

l.state\_gpu += l.hidden\*l.batch\*l.steps;

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

s.input = l.state\_gpu;

s.delta = self\_layer.delta\_gpu;

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

l.state\_gpu -= l.hidden\*l.batch;

copy\_ongpu(l.hidden\*l.batch, self\_layer.delta\_gpu, 1, input\_layer.delta\_gpu, 1); // the same delta for Input and Self layers

s.input = l.state\_gpu;

s.delta = self\_layer.delta\_gpu - l.hidden\*l.batch;

if (i == 0) s.delta = 0;

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

//copy\_ongpu(l.hidden\*l.batch, self\_layer.delta\_gpu, 1, input\_layer.delta\_gpu, 1);

if (i > 0 && l.shortcut) axpy\_ongpu(l.hidden\*l.batch, 1, self\_layer.delta\_gpu, 1, self\_layer.delta\_gpu - l.hidden\*l.batch, 1);

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

if(state.delta) s.delta = state.delta + i\*l.inputs\*l.batch;

else s.delta = 0;

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

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

increment\_layer(&self\_layer, -1);

increment\_layer(&output\_layer, -1);

}

}

#endif