#include "network.h"

#include "cost\_layer.h"

#include "utils.h"

#include "blas.h"

#include "parser.h"

typedef struct {

float \*x;

float \*y;

} float\_pair;

int \*read\_tokenized\_data(char \*filename, size\_t \*read)

{

size\_t size = 512;

size\_t count = 0;

FILE \*fp = fopen(filename, "r");

int\* d = (int\*)xcalloc(size, sizeof(int));

int n, one;

one = fscanf(fp, "%d", &n);

while(one == 1){

++count;

if(count > size){

size = size\*2;

d = (int\*)xrealloc(d, size \* sizeof(int));

}

d[count-1] = n;

one = fscanf(fp, "%d", &n);

}

fclose(fp);

d = (int\*)xrealloc(d, count \* sizeof(int));

\*read = count;

return d;

}

char \*\*read\_tokens(char \*filename, size\_t \*read)

{

size\_t size = 512;

size\_t count = 0;

FILE \*fp = fopen(filename, "r");

char\*\* d = (char\*\*)xcalloc(size, sizeof(char\*));

char \*line;

while((line=fgetl(fp)) != 0){

++count;

if(count > size){

size = size\*2;

d = (char\*\*)xrealloc(d, size \* sizeof(char\*));

}

d[count-1] = line;

}

fclose(fp);

d = (char\*\*)xrealloc(d, count \* sizeof(char\*));

\*read = count;

return d;

}

float\_pair get\_rnn\_token\_data(int \*tokens, size\_t \*offsets, int characters, size\_t len, int batch, int steps)

{

float\* x = (float\*)xcalloc(batch \* steps \* characters, sizeof(float));

float\* y = (float\*)xcalloc(batch \* steps \* characters, sizeof(float));

int i,j;

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

for(j = 0; j < steps; ++j){

int curr = tokens[(offsets[i])%len];

int next = tokens[(offsets[i] + 1)%len];

x[(j\*batch + i)\*characters + curr] = 1;

y[(j\*batch + i)\*characters + next] = 1;

offsets[i] = (offsets[i] + 1) % len;

if(curr >= characters || curr < 0 || next >= characters || next < 0){

error("Bad char");

}

}

}

float\_pair p;

p.x = x;

p.y = y;

return p;

}

float\_pair get\_rnn\_data(unsigned char \*text, size\_t \*offsets, int characters, size\_t len, int batch, int steps)

{

float\* x = (float\*)xcalloc(batch \* steps \* characters, sizeof(float));

float\* y = (float\*)xcalloc(batch \* steps \* characters, sizeof(float));

int i,j;

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

for(j = 0; j < steps; ++j){

unsigned char curr = text[(offsets[i])%len];

unsigned char next = text[(offsets[i] + 1)%len];

x[(j\*batch + i)\*characters + curr] = 1;

y[(j\*batch + i)\*characters + next] = 1;

offsets[i] = (offsets[i] + 1) % len;

if(curr > 255 || curr <= 0 || next > 255 || next <= 0){

/\*text[(index+j+2)%len] = 0;

printf("%ld %d %d %d %d\n", index, j, len, (int)text[index+j], (int)text[index+j+1]);

printf("%s", text+index);

\*/

error("Bad char");

}

}

}

float\_pair p;

p.x = x;

p.y = y;

return p;

}

void reset\_rnn\_state(network net, int b)

{

int i;

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

#ifdef GPU

layer l = net.layers[i];

if(l.state\_gpu){

fill\_ongpu(l.outputs, 0, l.state\_gpu + l.outputs\*b, 1);

}

#endif

}

}

void train\_char\_rnn(char \*cfgfile, char \*weightfile, char \*filename, int clear, int tokenized)

{

srand(time(0));

unsigned char \*text = 0;

int \*tokens = 0;

size\_t size;

if(tokenized){

tokens = read\_tokenized\_data(filename, &size);

} else {

FILE \*fp = fopen(filename, "rb");

fseek(fp, 0, SEEK\_END);

size = ftell(fp);

fseek(fp, 0, SEEK\_SET);

text = (unsigned char \*)xcalloc(size + 1, sizeof(char));

fread(text, 1, size, fp);

fclose(fp);

}

char\* backup\_directory = "backup/";

char \*base = basecfg(cfgfile);

fprintf(stderr, "%s\n", base);

float avg\_loss = -1;

network net = parse\_network\_cfg(cfgfile);

if(weightfile){

load\_weights(&net, weightfile);

}

int inputs = get\_network\_input\_size(net);

fprintf(stderr, "Learning Rate: %g, Momentum: %g, Decay: %g\n", net.learning\_rate, net.momentum, net.decay);

int batch = net.batch;

int steps = net.time\_steps;

if (clear) {

\*net.seen = 0;

\*net.cur\_iteration = 0;

}

int i = (\*net.seen)/net.batch;

int streams = batch/steps;

printf("\n batch = %d, steps = %d, streams = %d, subdivisions = %d, text\_size = %ld \n", batch, steps, streams, net.subdivisions, size);

printf(" global\_batch = %d \n", batch\*net.subdivisions);

size\_t\* offsets = (size\_t\*)xcalloc(streams, sizeof(size\_t));

int j;

for(j = 0; j < streams; ++j){

offsets[j] = rand\_size\_t()%size;

//printf(" offset[%d] = %d, ", j, offsets[j]);

}

//printf("\n");

clock\_t time;

while(get\_current\_batch(net) < net.max\_batches){

i += 1;

time=clock();

float\_pair p;

if(tokenized){

p = get\_rnn\_token\_data(tokens, offsets, inputs, size, streams, steps);

}else{

p = get\_rnn\_data(text, offsets, inputs, size, streams, steps);

}

float loss = train\_network\_datum(net, p.x, p.y) / (batch);

free(p.x);

free(p.y);

if (avg\_loss < 0) avg\_loss = loss;

avg\_loss = avg\_loss\*.9 + loss\*.1;

int chars = get\_current\_batch(net)\*batch;

fprintf(stderr, "%d: %f, %f avg, %f rate, %lf seconds, %f epochs\n", i, loss, avg\_loss, get\_current\_rate(net), sec(clock()-time), (float) chars/size);

for(j = 0; j < streams; ++j){

//printf("%d\n", j);

if(rand()%10 == 0){

//fprintf(stderr, "Reset\n");

offsets[j] = rand\_size\_t()%size;

reset\_rnn\_state(net, j);

}

}

if(i%1000==0){

char buff[256];

sprintf(buff, "%s/%s\_%d.weights", backup\_directory, base, i);

save\_weights(net, buff);

}

if(i%10==0){

char buff[256];

sprintf(buff, "%s/%s.backup", backup\_directory, base);

save\_weights(net, buff);

}

}

char buff[256];

sprintf(buff, "%s/%s\_final.weights", backup\_directory, base);

save\_weights(net, buff);

}

void print\_symbol(int n, char \*\*tokens){

if(tokens){

printf("%s ", tokens[n]);

} else {

printf("%c", n);

}

}

void test\_char\_rnn(char \*cfgfile, char \*weightfile, int num, char \*seed, float temp, int rseed, char \*token\_file)

{

char \*\*tokens = 0;

if(token\_file){

size\_t n;

tokens = read\_tokens(token\_file, &n);

}

srand(rseed);

char \*base = basecfg(cfgfile);

fprintf(stderr, "%s\n", base);

network net = parse\_network\_cfg\_custom(cfgfile, 1, 1); // batch=1, time\_steps=1

if(weightfile){

load\_weights(&net, weightfile);

}

int inputs = get\_network\_input\_size(net);

int i, j;

for(i = 0; i < net.n; ++i) net.layers[i].temperature = temp;

int c = 0;

int len = strlen(seed);

float\* input = (float\*)xcalloc(inputs, sizeof(float));

/\*

fill\_cpu(inputs, 0, input, 1);

for(i = 0; i < 10; ++i){

network\_predict(net, input);

}

fill\_cpu(inputs, 0, input, 1);

\*/

for(i = 0; i < len-1; ++i){

c = seed[i];

input[c] = 1;

network\_predict(net, input);

input[c] = 0;

print\_symbol(c, tokens);

}

if(len) c = seed[len-1];

print\_symbol(c, tokens);

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

input[c] = 1;

float \*out = network\_predict(net, input);

input[c] = 0;

for(j = 32; j < 127; ++j){

//printf("%d %c %f\n",j, j, out[j]);

}

for(j = 0; j < inputs; ++j){

if (out[j] < .0001) out[j] = 0;

}

c = sample\_array(out, inputs);

//c = sample\_array\_custom(out, inputs);

//c = max\_index(out, inputs);

//c = top\_max\_index(out, inputs, 2);

print\_symbol(c, tokens);

}

printf("\n");

}

void test\_tactic\_rnn(char \*cfgfile, char \*weightfile, int num, float temp, int rseed, char \*token\_file)

{

char \*\*tokens = 0;

if(token\_file){

size\_t n;

tokens = read\_tokens(token\_file, &n);

}

srand(rseed);

char \*base = basecfg(cfgfile);

fprintf(stderr, "%s\n", base);

network net = parse\_network\_cfg(cfgfile);

if(weightfile){

load\_weights(&net, weightfile);

}

int inputs = get\_network\_input\_size(net);

int i, j;

for(i = 0; i < net.n; ++i) net.layers[i].temperature = temp;

int c = 0;

float\* input = (float\*)xcalloc(inputs, sizeof(float));

float \*out = 0;

while((c = getc(stdin)) != EOF){

input[c] = 1;

out = network\_predict(net, input);

input[c] = 0;

}

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

for(j = 0; j < inputs; ++j){

if (out[j] < .0001) out[j] = 0;

}

int next = sample\_array(out, inputs);

if(c == '.' && next == '\n') break;

c = next;

print\_symbol(c, tokens);

input[c] = 1;

out = network\_predict(net, input);

input[c] = 0;

}

printf("\n");

}

void valid\_tactic\_rnn(char \*cfgfile, char \*weightfile, char \*seed)

{

char \*base = basecfg(cfgfile);

fprintf(stderr, "%s\n", base);

network net = parse\_network\_cfg(cfgfile);

if(weightfile){

load\_weights(&net, weightfile);

}

int inputs = get\_network\_input\_size(net);

int count = 0;

int words = 1;

int c;

int len = strlen(seed);

float\* input = (float\*)xcalloc(inputs, sizeof(float));

int i;

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

c = seed[i];

input[(int)c] = 1;

network\_predict(net, input);

input[(int)c] = 0;

}

float sum = 0;

c = getc(stdin);

float log2 = log(2);

int in = 0;

while(c != EOF){

int next = getc(stdin);

if(next == EOF) break;

if(next < 0 || next >= 255) error("Out of range character");

input[c] = 1;

float \*out = network\_predict(net, input);

input[c] = 0;

if(c == '.' && next == '\n') in = 0;

if(!in) {

if(c == '>' && next == '>'){

in = 1;

++words;

}

c = next;

continue;

}

++count;

sum += log(out[next])/log2;

c = next;

printf("%d %d Perplexity: %4.4f Word Perplexity: %4.4f\n", count, words, pow(2, -sum/count), pow(2, -sum/words));

}

}

void valid\_char\_rnn(char \*cfgfile, char \*weightfile, char \*seed)

{

char \*base = basecfg(cfgfile);

fprintf(stderr, "%s\n", base);

network net = parse\_network\_cfg(cfgfile);

if(weightfile){

load\_weights(&net, weightfile);

}

int inputs = get\_network\_input\_size(net);

int count = 0;

int words = 1;

int c;

int len = strlen(seed);

float\* input = (float\*)xcalloc(inputs, sizeof(float));

int i;

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

c = seed[i];

input[(int)c] = 1;

network\_predict(net, input);

input[(int)c] = 0;

}

float sum = 0;

c = getc(stdin);

float log2 = log(2);

while(c != EOF){

int next = getc(stdin);

if(next == EOF) break;

if(next < 0 || next >= 255) error("Out of range character");

++count;

if(next == ' ' || next == '\n' || next == '\t') ++words;

input[c] = 1;

float \*out = network\_predict(net, input);

input[c] = 0;

sum += log(out[next])/log2;

c = next;

printf("%d Perplexity: %4.4f Word Perplexity: %4.4f\n", count, pow(2, -sum/count), pow(2, -sum/words));

}

}

void vec\_char\_rnn(char \*cfgfile, char \*weightfile, char \*seed)

{

char \*base = basecfg(cfgfile);

fprintf(stderr, "%s\n", base);

network net = parse\_network\_cfg(cfgfile);

if(weightfile){

load\_weights(&net, weightfile);

}

int inputs = get\_network\_input\_size(net);

int c;

int seed\_len = strlen(seed);

float\* input = (float\*)xcalloc(inputs, sizeof(float));

int i;

char \*line;

while((line=fgetl(stdin)) != 0){

reset\_rnn\_state(net, 0);

for(i = 0; i < seed\_len; ++i){

c = seed[i];

input[(int)c] = 1;

network\_predict(net, input);

input[(int)c] = 0;

}

strip(line);

int str\_len = strlen(line);

for(i = 0; i < str\_len; ++i){

c = line[i];

input[(int)c] = 1;

network\_predict(net, input);

input[(int)c] = 0;

}

c = ' ';

input[(int)c] = 1;

network\_predict(net, input);

input[(int)c] = 0;

layer l = net.layers[0];

#ifdef GPU

cuda\_pull\_array(l.output\_gpu, l.output, l.outputs);

#endif

printf("%s", line);

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

printf(",%g", l.output[i]);

}

printf("\n");

}

}

void run\_char\_rnn(int argc, char \*\*argv)

{

if(argc < 4){

fprintf(stderr, "usage: %s %s [train/test/valid] [cfg] [weights (optional)]\n", argv[0], argv[1]);

return;

}

char \*filename = find\_char\_arg(argc, argv, "-file", "data/shakespeare.txt");

char \*seed = find\_char\_arg(argc, argv, "-seed", "\n\n");

int len = find\_int\_arg(argc, argv, "-len", 1000);

float temp = find\_float\_arg(argc, argv, "-temp", .7);

int rseed = find\_int\_arg(argc, argv, "-srand", time(0));

int clear = find\_arg(argc, argv, "-clear");

int tokenized = find\_arg(argc, argv, "-tokenized");

char \*tokens = find\_char\_arg(argc, argv, "-tokens", 0);

char \*cfg = argv[3];

char \*weights = (argc > 4) ? argv[4] : 0;

if(0==strcmp(argv[2], "train")) train\_char\_rnn(cfg, weights, filename, clear, tokenized);

else if(0==strcmp(argv[2], "valid")) valid\_char\_rnn(cfg, weights, seed);

else if(0==strcmp(argv[2], "validtactic")) valid\_tactic\_rnn(cfg, weights, seed);

else if(0==strcmp(argv[2], "vec")) vec\_char\_rnn(cfg, weights, seed);

else if(0==strcmp(argv[2], "generate")) test\_char\_rnn(cfg, weights, len, seed, temp, rseed, tokens);

else if(0==strcmp(argv[2], "generatetactic")) test\_tactic\_rnn(cfg, weights, len, temp, rseed, tokens);

}