#ifndef ACTIVATIONS\_H

#define ACTIVATIONS\_H

#include "darknet.h"

#include "dark\_cuda.h"

#include "math.h"

#include "utils.h"

//typedef enum{

// LOGISTIC, RELU, RELIE, LINEAR, RAMP, TANH, PLSE, LEAKY, ELU, LOGGY, STAIR, HARDTAN, LHTAN, SELU, SWISH, MISH

//}ACTIVATION;

#ifdef \_\_cplusplus

extern "C" {

#endif

ACTIVATION get\_activation(char \*s);

char \*get\_activation\_string(ACTIVATION a);

float activate(float x, ACTIVATION a);

float gradient(float x, ACTIVATION a);

void gradient\_array(const float \*x, const int n, const ACTIVATION a, float \*delta);

void gradient\_array\_swish(const float \*x, const int n, const float \* sigmoid, float \* delta);

void gradient\_array\_mish(const int n, const float \* activation\_input, float \* delta);

void activate\_array(float \*x, const int n, const ACTIVATION a);

void activate\_array\_swish(float \*x, const int n, float \* output\_sigmoid, float \* output);

void activate\_array\_mish(float \*x, const int n, float \* activation\_input, float \* output);

void activate\_array\_normalize\_channels(float \*x, const int n, int batch, int channels, int wh\_step, float \*output);

void gradient\_array\_normalize\_channels(float \*x, const int n, int batch, int channels, int wh\_step, float \*delta);

void activate\_array\_normalize\_channels\_softmax(float \*x, const int n, int batch, int channels, int wh\_step, float \*output, int use\_max\_val);

void gradient\_array\_normalize\_channels\_softmax(float \*x, const int n, int batch, int channels, int wh\_step, float \*delta);

#ifdef GPU

void activate\_array\_ongpu(float \*x, int n, ACTIVATION a);

void activate\_array\_swish\_ongpu(float \*x, int n, float \*output\_sigmoid\_gpu, float \*output\_gpu);

void activate\_array\_mish\_ongpu(float \*x, int n, float \*activation\_input\_gpu, float \*output\_gpu);

void gradient\_array\_ongpu(float \*x, int n, ACTIVATION a, float \*delta);

void gradient\_array\_swish\_ongpu(float \*x, int n, float \*sigmoid\_gpu, float \*delta);

void gradient\_array\_mish\_ongpu(int n, float \*activation\_input\_gpu, float \*delta);

void activate\_array\_normalize\_channels\_ongpu(float \*x, int n, int batch, int channels, int wh\_step, float \*output\_gpu);

void gradient\_array\_normalize\_channels\_ongpu(float \*output\_gpu, int n, int batch, int channels, int wh\_step, float \*delta\_gpu);

void activate\_array\_normalize\_channels\_softmax\_ongpu(float \*x, int n, int batch, int channels, int wh\_step, float \*output\_gpu, int use\_max\_val);

void gradient\_array\_normalize\_channels\_softmax\_ongpu(float \*output\_gpu, int n, int batch, int channels, int wh\_step, float \*delta\_gpu);

#endif

static inline float stair\_activate(float x)

{

int n = floorf(x);

if (n%2 == 0) return floorf(x/2.f);

else return (x - n) + floorf(x/2.f);

}

static inline float hardtan\_activate(float x)

{

if (x < -1) return -1;

if (x > 1) return 1;

return x;

}

static inline float linear\_activate(float x){return x;}

static inline float logistic\_activate(float x){return 1.f/(1.f + expf(-x));}

static inline float loggy\_activate(float x){return 2.f/(1.f + expf(-x)) - 1;}

static inline float relu\_activate(float x){return x\*(x>0);}

static inline float relu6\_activate(float x) { return min\_val\_cmp(max\_val\_cmp(x, 0), 6); }

static inline float elu\_activate(float x){return (x >= 0)\*x + (x < 0)\*(expf(x)-1);}

static inline float selu\_activate(float x) { return (x >= 0)\*1.0507f\*x + (x < 0)\*1.0507f\*1.6732f\*(expf(x) - 1); }

static inline float relie\_activate(float x){return (x>0) ? x : .01f\*x;}

static inline float ramp\_activate(float x){return x\*(x>0)+.1f\*x;}

static inline float leaky\_activate(float x){return (x>0) ? x : .1f\*x;}

//static inline float tanh\_activate(float x){return (expf(2\*x)-1)/(expf(2\*x)+1);}

static inline float tanh\_activate(float x) { return (2 / (1 + expf(-2 \* x)) - 1); }

static inline float gelu\_activate(float x) { return (0.5\*x\*(1 + tanhf(0.797885\*x + 0.035677\*powf(x, 3)))); }

static inline float softplus\_activate(float x, float threshold) {

if (x > threshold) return x; // too large

else if (x < -threshold) return expf(x); // too small

return logf(expf(x) + 1);

}

static inline float plse\_activate(float x)

{

if(x < -4) return .01f \* (x + 4);

if(x > 4) return .01f \* (x - 4) + 1;

return .125f\*x + .5f;

}

static inline float lhtan\_activate(float x)

{

if(x < 0) return .001f\*x;

if(x > 1) return .001f\*(x-1) + 1;

return x;

}

static inline float lhtan\_gradient(float x)

{

if(x > 0 && x < 1) return 1;

return .001f;

}

static inline float hardtan\_gradient(float x)

{

if (x > -1 && x < 1) return 1;

return 0;

}

static inline float linear\_gradient(float x){return 1;}

static inline float logistic\_gradient(float x){return (1-x)\*x;}

static inline float loggy\_gradient(float x)

{

float y = (x+1.f)/2.f;

return 2\*(1-y)\*y;

}

static inline float stair\_gradient(float x)

{

if (floor(x) == x) return 0;

return 1.0f;

}

static inline float relu\_gradient(float x){return (x>0);}

static inline float relu6\_gradient(float x) { return (x > 0 && x < 6); }

static inline float elu\_gradient(float x){return (x >= 0) + (x < 0)\*(x + 1);}

static inline float selu\_gradient(float x) { return (x >= 0)\*1.0507f + (x < 0)\*(x + 1.0507f\*1.6732f); }

static inline float relie\_gradient(float x){return (x>0) ? 1 : .01f;}

static inline float ramp\_gradient(float x){return (x>0)+.1f;}

static inline float leaky\_gradient(float x){return (x>0) ? 1 : .1f;}

static inline float tanh\_gradient(float x){return 1-x\*x;}

static inline float sech(float x) { return 2 / (expf(x) + expf(-x)); }

static inline float gelu\_gradient(float x) {

const float x3 = powf(x, 3);

return 0.5\*tanhf(0.0356774\*x3 + 0.797885\*x) + (0.0535161\*x3 + 0.398942\*x) \* powf(sech(0.0356774\*x3 + 0.797885\*x), 2) + 0.5;

}

static inline float plse\_gradient(float x){return (x < 0 || x > 1) ? .01f : .125f;}

#ifdef \_\_cplusplus

}

#endif

#endif