#ifndef BASE\_LAYER\_H

#define BASE\_LAYER\_H

#include "activations.h"

#include "stddef.h"

#include "tree.h"

#ifdef \_\_cplusplus

extern "C" {

#endif

//struct network\_state;

//struct layer;

//typedef struct layer layer;

//typedef enum {

// CONVOLUTIONAL,

// DECONVOLUTIONAL,

// CONNECTED,

// MAXPOOL,

// SOFTMAX,

// DETECTION,

// DROPOUT,

// CROP,

// ROUTE,

// COST,

// NORMALIZATION,

// AVGPOOL,

// LOCAL,

// SHORTCUT,

// ACTIVE,

// RNN,

// GRU,

// CRNN,

// BATCHNORM,

// NETWORK,

// XNOR,

// REGION,

// YOLO,

// REORG,

// UPSAMPLE,

// REORG\_OLD,

// BLANK

//} LAYER\_TYPE;

//typedef enum{

// SSE, MASKED, SMOOTH

//} COST\_TYPE;

//typedef struct {

// int batch;

// float learning\_rate;

// float momentum;

// float decay;

// int adam;

// float B1;

// float B2;

// float eps;

// int t;

//} update\_args;

/\*

struct layer{

LAYER\_TYPE type;

ACTIVATION activation;

COST\_TYPE cost\_type;

void (\*forward) (struct layer, struct network\_state);

void (\*backward) (struct layer, struct network\_state);

void (\*update) (struct layer, int, float, float, float);

void (\*forward\_gpu) (struct layer, struct network\_state);

void (\*backward\_gpu) (struct layer, struct network\_state);

void (\*update\_gpu) (struct layer, int, float, float, float);

int batch\_normalize;

int shortcut;

int batch;

int forced;

int flipped;

int inputs;

int outputs;

int truths;

int h,w,c;

int out\_h, out\_w, out\_c;

int n;

int max\_boxes;

int groups;

int size;

int side;

int stride;

int reverse;

int spatial;

int pad;

int sqrt;

int flip;

int index;

int binary;

int xnor;

int use\_bin\_output;

int steps;

int hidden;

float dot;

float angle;

float jitter;

float saturation;

float exposure;

float shift;

float ratio;

float learning\_rate\_scale;

int focal\_loss;

int noloss;

int softmax;

int classes;

int coords;

int background;

int rescore;

int objectness;

int does\_cost;

int joint;

int noadjust;

int reorg;

int log;

int tanh;

int \*mask;

int total;

float bflops;

int adam;

float B1;

float B2;

float eps;

int t;

float \*m;

float \*v;

float \* bias\_m;

float \* bias\_v;

float \* scale\_m;

float \* scale\_v;

tree \*softmax\_tree;

int \*map;

float alpha;

float beta;

float kappa;

float coord\_scale;

float object\_scale;

float noobject\_scale;

float mask\_scale;

float class\_scale;

int bias\_match;

int random;

float ignore\_thresh;

float truth\_thresh;

float thresh;

float focus;

int classfix;

int absolute;

int onlyforward;

int stopbackward;

int dontload;

int dontloadscales;

float temperature;

float probability;

float scale;

int \*indexes;

float \*rand;

float \*cost;

char \*cweights;

float \*state;

float \*prev\_state;

float \*forgot\_state;

float \*forgot\_delta;

float \*state\_delta;

float \*concat;

float \*concat\_delta;

float \*binary\_weights;

float \*biases;

float \*bias\_updates;

float \*scales;

float \*scale\_updates;

float \*weights;

float \*weight\_updates;

char \*align\_bit\_weights\_gpu;

float \*mean\_arr\_gpu;

float \*align\_workspace\_gpu;

float \*transposed\_align\_workspace\_gpu;

int align\_workspace\_size;

char \*align\_bit\_weights;

float \*mean\_arr;

int align\_bit\_weights\_size;

int lda\_align;

int new\_lda;

int bit\_align;

float \*col\_image;

int \* input\_layers;

int \* input\_sizes;

float \* delta;

float \* output;

float \* loss;

float \* squared;

float \* norms;

float \* spatial\_mean;

float \* mean;

float \* variance;

float \* mean\_delta;

float \* variance\_delta;

float \* rolling\_mean;

float \* rolling\_variance;

float \* x;

float \* x\_norm;

struct layer \*input\_layer;

struct layer \*self\_layer;

struct layer \*output\_layer;

struct layer \*input\_gate\_layer;

struct layer \*state\_gate\_layer;

struct layer \*input\_save\_layer;

struct layer \*state\_save\_layer;

struct layer \*input\_state\_layer;

struct layer \*state\_state\_layer;

struct layer \*input\_z\_layer;

struct layer \*state\_z\_layer;

struct layer \*input\_r\_layer;

struct layer \*state\_r\_layer;

struct layer \*input\_h\_layer;

struct layer \*state\_h\_layer;

float \*z\_cpu;

float \*r\_cpu;

float \*h\_cpu;

float \*binary\_input;

size\_t workspace\_size;

#ifdef GPU

float \*z\_gpu;

float \*r\_gpu;

float \*h\_gpu;

int \*indexes\_gpu;

float \* prev\_state\_gpu;

float \* forgot\_state\_gpu;

float \* forgot\_delta\_gpu;

float \* state\_gpu;

float \* state\_delta\_gpu;

float \* gate\_gpu;

float \* gate\_delta\_gpu;

float \* save\_gpu;

float \* save\_delta\_gpu;

float \* concat\_gpu;

float \* concat\_delta\_gpu;

// adam

float \*m\_gpu;

float \*v\_gpu;

float \*bias\_m\_gpu;

float \*scale\_m\_gpu;

float \*bias\_v\_gpu;

float \*scale\_v\_gpu;

float \*binary\_input\_gpu;

float \*binary\_weights\_gpu;

float \* mean\_gpu;

float \* variance\_gpu;

float \* rolling\_mean\_gpu;

float \* rolling\_variance\_gpu;

float \* variance\_delta\_gpu;

float \* mean\_delta\_gpu;

float \* col\_image\_gpu;

float \* x\_gpu;

float \* x\_norm\_gpu;

float \* weights\_gpu;

float \* weight\_updates\_gpu;

float \* weights\_gpu16;

float \* weight\_updates\_gpu16;

float \* biases\_gpu;

float \* bias\_updates\_gpu;

float \* scales\_gpu;

float \* scale\_updates\_gpu;

float \* output\_gpu;

float \* loss\_gpu;

float \* delta\_gpu;

float \* rand\_gpu;

float \* squared\_gpu;

float \* norms\_gpu;

#ifdef CUDNN

cudnnTensorDescriptor\_t srcTensorDesc, dstTensorDesc;

cudnnTensorDescriptor\_t srcTensorDesc16, dstTensorDesc16;

cudnnTensorDescriptor\_t dsrcTensorDesc, ddstTensorDesc;

cudnnTensorDescriptor\_t dsrcTensorDesc16, ddstTensorDesc16;

cudnnTensorDescriptor\_t normTensorDesc, normDstTensorDesc, normDstTensorDescF16;

cudnnFilterDescriptor\_t weightDesc, weightDesc16;

cudnnFilterDescriptor\_t dweightDesc, dweightDesc16;

cudnnConvolutionDescriptor\_t convDesc;

cudnnConvolutionFwdAlgo\_t fw\_algo, fw\_algo16;

cudnnConvolutionBwdDataAlgo\_t bd\_algo, bd\_algo16;

cudnnConvolutionBwdFilterAlgo\_t bf\_algo, bf\_algo16;

cudnnPoolingDescriptor\_t poolingDesc;

#endif // CUDNN

#endif // GPU

};

\*/

//void free\_layer(layer);

#ifdef \_\_cplusplus

}

#endif

#endif