Yolo 源码解析

2017年5月22日 9:51

./darknet detector train cfg/xx.data cfg/yolo-xx.cfg darknet19.weights.conv.23

darknet.c 训练检测 网络结构 参数初始值

darknet.c
```
int main(int argc, char **argv)
{
    if (0 == strcmp(argv[1], "average")){
        average(argc, argv);
    } else if (0 == strcmp(argv[1], "yolo")){
        run_yolo(argc, argv);
    } else if (0 == strcmp(argv[1], "voxel")){
        run_voxel(argc, argv);
    } else if (0 == strcmp(argv[1], "super")){
        run_super(argc, argv);
    } else if (0 == strcmp(argv[1], "detector")){
        run_detector(argc, argv);
        ...
}
```

//data.h
```
typedef struct{
    int w, h;
    matrix X;
    matrix y;
    int shallow;
    int *num_boxes;
    box **boxes;
} data;
```

//matrix.h
//这里rows是一次加载到内存中的样本的个数（batch*net.subdivisions），cols就是样本的维度，
**vals指向的是样本的值
```
typedef struct matrix{
    int rows, cols;
    float **vals;
} matrix;
```

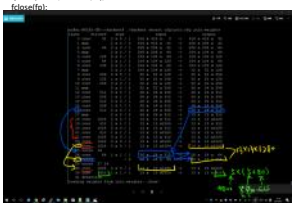
detector.c
```
void run_detector(int argc, char **argv)
{
    ...
    if(0==strcmp(argv[2], "test")) test_detector(datacfg, cfg, weights, filename, thresh, hier_thresh, outfile, fullscreen);
    else if(0==strcmp(argv[2], "train")) train_detector(datacfg, cfg, weights, gpus, ngpus, clear);
    else if(0==strcmp(argv[2], "valid")) validate_detector(datacfg, cfg, weights, outfile);
    else if(0==strcmp(argv[2], "valid2")) validate_detector_flip(datacfg, cfg, weights, outfile);
    else if(0==strcmp(argv[2], "recall")) validate_detector_recall(cfg, weights);
    else if(0==strcmp(argv[2], "demo")) {
        list *options = read_data_cfg(datacfg);
        int classes = option_find_int(options, "classes", 20);
        char *name_list = option_find_str(options, "names", "data/names.list");
        char **names = get_labels(name_list);
        demo(cfg, weights, thresh, cam_index, filename, names, classes, frame_skip, prefix, avg, hier_thresh, width, height, fps, fullscreen);
    }
}
```

```
void train_detector(char *datacfg, char *cfgfile, char *weightfile, int *gpus, int ngpus, int clear)
{
    //read params from the data config file.
    // train images, test validate images ，backup_directory
    list *options = read_data_cfg(datacfg);
    ...
    // parse network and load weights
    nets[i] = load_network(cfgfile, weightfile, clear);
    ...
    data train, buffer;
    ...
    //调节图片旋转角度、曝光度、饱和度、色调等，来增加图片数量
    //声明线程ID
    pthread_t load_thread = load_data(args);
    pthread_join(load_thread, 0);
    /*pthread_join()函数，以阻塞的方式等待thread指定的线程结束。当函数返回时，被等待线程的资源被收回。如果线程已经结束，那么该函数会立即返回。 */
    ...
    printf("Loaded: %lf seconds\n", sec(clock()-time));
    ...
    loss = train_network(net, train);
    ...
}
```

network.c
```
network load_network(char *cfg, char *weights, int clear)
{
    network net = parse_network_cfg(cfg);
    if(weights && weights[0] != 0){
        load_weights(&net, weights);
    }
    if(clear) *net.seen = 0;
    return net;
}
```

Parser.c
```
network parse_network_cfg(char *filename)
{
    list *sections = read_cfg(filename);
    node *n = sections->front;
    if(!n) error("Config file has no sections");
    network net = make_network(sections->size - 1);
    net.gpu_index = gpu_index;
    size_params params;

    section *s = (section *)n->val;
    list *options = s->options;
    if(!is_network(s)) error("First section must be [net] or [network]");
    parse_net_options(options, &net);

    params.h = net.h;
    ...
    size_t workspace_size = 0;
    n = n->next;
    int count = 0;
    free_section(s);
    fprintf(stderr, "layer     filters    size              input                output\n");
    while(n){
        params.index = count;
        fprintf(stderr, "%5d ", count);
        s = (section *)n->val;
        options = s->options;
        layer l = {0};
        LAYER_TYPE lt = string_to_layer_type(s->type);
        if(lt == CONVOLUTIONAL){
            l = parse_convolutional(options, params);
        }else if(lt == DECONVOLUTIONAL){
            l = parse_deconvolutional(options, params);
        ...
        }else if(lt == REGION){
            l = parse_region(options, params);
        }else if(lt == DETECTION){
            l = parse_detection(options, params);
        }else if(lt == SOFTMAX){
            l = parse_softmax(options, params);
            net.hierarchy = l.softmax_tree;
        }else if(lt == NORMALIZATION){
        ...
        }else{
            fprintf(stderr, "Type not recognized: %s\n", s->type);
        }
        ...
        return net;
    }
```

Parser.c
```
void load_weights(network *net, char *filename)
{
    load_weights_upto(net, filename, 0, net->n);
}

void load_weights_upto(network *net, char *filename, int start, int cutoff)
{
#ifdef GPU
    if(net->gpu_index >= 0){
        cuda_set_device(net->gpu_index);
    }
#endif
    fprintf(stderr, "Loading weights from %s...", filename);
//fflush()函数冲洗流中的信息，该函数通常用于处理磁盘文件。
    fflush(stdout);
    FILE *fp = fopen(filename, "rb");
    if(!fp) file_error(filename);

    int major;
    int minor;
    int revision;
/*size_t fread ( void *buffer, size_t size, size_t count, FILE *stream)
    fread是一个函数。从一个文件流中读数据，最多读取count个项，每个项size个字节，如果调用成功返回实
    际读取到的项个数（小于或等于count），如果不成功或读到文件末尾返回0。*/
    fread(&major, sizeof(int), 1, fp);
    fread(&minor, sizeof(int), 1, fp);
    fread(&revision, sizeof(int), 1, fp);
    fread(net->seen, sizeof(int), 1, fp);
    int transpose = (major > 1000) || (minor > 1000);

    int i;
//cutoff额名思义，之后的参数不加载，fine tuning的时候用
    for(i = start; i < net->n && i < cutoff; ++i){
//读取各层权重
        layer l = net->layers[i];
        if (l.dontload) continue;
        if(l.type == CONVOLUTIONAL || l.type == DECONVOLUTIONAL){
            load_convolutional_weights(l, fp);
        }
        if(l.type == CONNECTED){
            load_connected_weights(l, fp, transpose);
        }
        ...
    }
    fprintf(stderr, "Done!\n");
    fclose(fp);
}
```



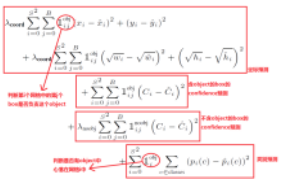
```
void load_convolutional_weights(layer l, FILE *fp)
{
    if(l.binary){
        //load_convolutional_weights_binary(l, fp);
        //return;
    }
    //卷积层的参数个数，卷积核个数×通道数×卷积核长度×卷积核宽度
    int num = l.n*l.c*l.size*l.size;
    //这个估计没用上吧，直接给 0 了
    if(0){
        fread(l.biases + ((l.n != 1374)?0:5), sizeof(float), l.n, fp);
        if (l.batch_normalize && (!l.dontloadscales)){
            fread(l.scales + ((l.n != 1374)?0:5), sizeof(float), l.n, fp);
            fread(l.rolling_mean + ((l.n != 1374)?0:5), sizeof(float), l.n, fp);
            fread(l.rolling_variance + ((l.n != 1374)?0:5), sizeof(float), l.n, fp);
        }
        fread(l.weights + ((l.n != 1374)?0:5*l.c*l.size*l.size), sizeof(float), num, fp);
    }else{
        fread(l.biases, sizeof(float), l.n, fp);
        //如果使用了batch Normalizationn (https://www.zhihu.com/question/38102762, https://arxiv.org/abs/1502.03167)，
        那就加载三个参数
        if (l.batch_normalize && (!l.dontloadscales)){
            fread(l.scales, sizeof(float), l.n, fp);
            fread(l.rolling_mean, sizeof(float), l.n, fp);
            fread(l.rolling_variance, sizeof(float), l.n, fp);
        }
        fread(l.weights, sizeof(float), num, fp);
    }
    if(l.adam){
        fread(l.m, sizeof(float), num, fp);
        fread(l.v, sizeof(float), num, fp);
    }
    //if(l.c == 3) scal_cpu(num, 1./256, l.weights, 1);
    if (l.flipped) {
        transpose_matrix(l.weights, l.c*l.size*l.size, l.n);
    }
    //if (l.binary) binarize_weights(l.weights, l.n, l.c*l.size*l.size, l.weights);
#ifdef GPU
    if(gpu_index >= 0){
        push_convolutional_layer(l);
    }
#endif
}
```

//data.c
```
void get_next_batch(data d, int n, int offset, float *X, float *y)
{
    int j;
    for(j = 0; j < n; ++j){
        //offset就是第几个batch(*batch)了，j表示的是每个batch中的第几个样本（图像
        int index = offset + j;
        //void *memcpy(void *dest, const void *src, size_t n);
        //memcpy()函数的功能是从源src所指的内存地址的起始位置开始拷贝n个字节到目标dest所指的内存地址的起始位置中
        memcpy(X+j*d.X.cols, d.X.vals[index], d.X.cols*sizeof(float));
        memcpy(y+j*d.y.cols, d.y.vals[index], d.y.cols*sizeof(float));
    }
}
```

network.c
```
float train_network(network net, data d)
{
    assert(d.X.rows % net.batch == 0);
    int batch = net.batch;
        //这里rows是一次加载到内存中的
        //样本的个数（batch*net.subdivisions），
        //cols就是样本的维度
        // 这里n 就是 subdivisions
    int n = d.X.rows / batch;

    int i;
    float sum = 0;
    for(i = 0; i < n; ++i){
        get_next_batch(d, batch, i*batch, net.input, net.truth);
        float err = train_network_datum(net);
        sum += err;
    }
    return (float)sum/(n*batch);
}

float train_network_datum(network net)
{
#ifdef GPU
    if(gpu_index >= 0) return train_network_datum_gpu(net);
#endif
    *net.seen += net.batch;
    net.train = 1;
    forward_network(net);
    backward_network(net);
    float error = *net.cost;
    if(((*net.seen)/net.batch)%net.subdivisions == 0) update_network(net);
    return error;
}

void forward_network(network net)
{
    int i;
    //net.n: number of layers
    for(i = 0; i < net.n; ++i){
        net.index = i;
        layer l = net.layers[i];
        if(l.delta){
            fill_cpu(l.outputs * l.batch, 0, l.delta, 1);
        }
        l.forward(l, net);
        net.input = l.output;
        if(l.truth) {
            net.truth = l.output;
        }
    }
    calc_network_cost(net);
}
```

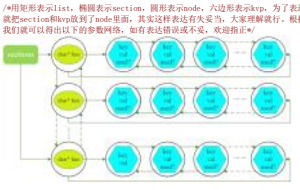
region_layer.c
```
layer make_region_layer(int batch, int w, int h, int n, int classes, int coords)
{
    ...
    l.forward = forward_region_layer;
    ...
}
```

```
void forward_region_layer(const layer l, network net)
{
    int i,j,b,t,n;
    memcpy(l.output, net.input, l.outputs*l.batch*sizeof(float));

    memset(l.delta, 0, l.outputs * l.batch * sizeof(float));
    if(!net.train) return;
    float avg_iou = 0;
    float recall = 0;
    float avg_cat = 0;
    float avg_obj = 0;
    float avg_anyobj = 0;
    int count = 0;
    int class_count = 0;
    *(l.cost) = 0;
    for (b = 0; b < l.batch; ++b) {

        //h*w*n*c. c=(classes+ coords + 1)
        for (j = 0; j < l.h; ++j) {
            for (i = 0; i < l.w; ++i) {
                for (n = 0; n < l.n; ++n) { //for each box
                    int box_index = entry_index(l, b, n*l.w*l.h + j*l.w + i, 0);
                                //biases : anchors. The pred box is calculated based on output and anchors.
                    box pred = get_region_box(l.output, l.biases, n, box_index, i, j, l.w, l.h, l.w*l.h);
                    float best_iou = 0;
                                // assume max 30 objects for the ground truth.
                                // find the best iou among all the ground truthes.
                    for(t = 0; t < 30; ++t){
                        box truth = float_to_box(net.truth + t*5 + b*l.truths, 1);
                        if(!truth.x) break;
                        float iou = box_iou(pred, truth);
                        if (iou > best_iou) {
                            best_iou = iou;
                        }
                    }
                                //计算置信度的损失
                                //预测值的坐标索引
                    int obj_index = entry_index(l, b, n*l.w*l.h + j*l.w + i, 4);
                    avg_anyobj += l.output[obj_index];
                                //对应论文公式，这里先假设g个框中都没有物体
                    l.delta[obj_index] = l.noobject_scale * (0 - l.output[obj_index]);
                    if(l.background) l.delta[obj_index] = l.noobject_scale * (1 - l.output[obj_index]);
                    // larger than threshold-> there is an object !
                                if (best_iou > l.thresh) {
                        l.delta[obj_index] = 0;
                    }
                }
            }
        }
        for(t = 0; t < 30; ++t){
                                //For each object of the whole image.
            box truth = float_to_box(net.truth + t*5 + b*l.truths, 1);

            if(!truth.x) break;
            float best_iou = 0;
            int best_n = 0;
            i = (truth.x * l.w);
            j = (truth.y * l.h);
            //printf("%d %f %d %f\n", i, truth.x*l.w, j, truth.y*l.h);
            box truth_shift = truth;
            truth_shift.x = 0;
            truth_shift.y = 0;
            //printf("index %d %d\n",i, j);

                        // Totally n pred boxes. Find the best box scale for that object.
            for(n = 0; n < l.n; ++n){
                int box_index = entry_index(l, b, n*l.w*l.h + j*l.w + i, 0);
                box pred = get_region_box(l.output, l.biases, n, box_index, i, j, l.w, l.h, l.w*l.h);
                //bias_match is 1. means use anchor box without net output.
                                if(l.bias_match){
                    pred.w = l.biases[2*n]/l.w;
                    pred.h = l.biases[2*n+1]/l.h;
                }
                //printf("pred: (%f, %f) %f x %f\n", pred.x, pred.y, pred.w, pred.h);
                pred.x = 0;
                pred.y = 0;
                                //seems x,y doesn't matter. only w, h matter.
                float iou = box_iou(pred, truth_shift);
                if (iou > best_iou){
                    best_iou = iou;
                    best_n = n;
                }
            }
            int box_index = entry_index(l, b, best_n*l.w*l.h + j*l.w + i, 0);
                                // Calculate the loss for location error .
            float iou = delta_region_box(truth, l.output, l.biases, best_n, box_index, i, j, l.w, l.h, l.delta, l.coord_scale *  (2 - truth.w*truth.h), l.w*l.h);
            // Calculate the recall
                                if(iou > .5) recall += 1;
            avg_iou += iou;

            //l.delta[best_index + 4] = iou - l.output[best_index + 4];
                                //最好的那个BOX的坐标
            int obj_index = entry_index(l, b, best_n*l.w*l.h + j*l.w + i, l.coords);
            avg_obj += l.output[obj_index];
            l.delta[obj_index] = l.object_scale * (1 - l.output[obj_index]);
                                //通过置打分来调整l.delta（预测值与真实值的差）
            if (l.rescore) {
                l.delta[obj_index] = l.object_scale * (iou - l.output[obj_index]);
            }

            int class = net.truth[t*(l.coords + 1) + b*l.truths + l.coords];
            if (l.map) class = l.map[class];
            int class_index = entry_index(l, b, best_n*l.w*l.h + j*l.w + i, l.coords + 1);
            //计算分类的损失
                                delta_region_class(l.output, l.delta, class_index, class, l.classes, l.softmax_tree,
                                                   l.class_scale, l.w*l.h, &avg_cat);
            ++count;
            ++class_count;
        }
    }
    *(l.cost) = pow(mag_array(l.delta, l.outputs * l.batch), 2);
        /*Region Avg IOU：平均的IOU，代表预测的bounding box和ground truth的交集与并集之比，期望该值趋近于1。
    Class:是标注物体的概率，期望该值趋近于1.Obj：期望该值趋近于1.No Obj:期望该值越来越小但不为零.
    Avg Recall：期望该值趋近1 avg：平均损失，期望该值趋近于0*/
        printf("Region Avg IOU: %f, Class: %f, Obj: %f, No Obj: %f, Avg Recall: %f,  count: %d \n", avg_iou/count, avg_cat/class_count, avg_obj/count, avg_anyobj/(l.w*l.h*l.n*l.batch), recall/count, count);
}
```



//首先我们看一下list，section，node，kvp这四个结构体

/*用矩形表示list，椭圆表示section，圆形表示node，六边形表示kvp，为了表达方便，我就把section和kvp放到了node里面，其实这样表达有失妥当，大家理解就行。根据作者代码我们就可以得出以下的参数网络，如有表达错误或不妥，欢迎指正*/



我们来大概解释下该参数网，首先创建一个list，取名sections，记录一共有多少个section（一个section存储了CNN一层所需参数）；然后创建一个node，该node的void类型的指针指向一个新创建的section；该section的char类型指针指向.cfg文件中的某一行（line），然后将该section的list指针指向一个新创建的node，该node的void指针指向一个kvp结构体，kvp结构体中的key就是.cfg文件中的关键字（如：batch，subdivisions等），val就是对应的值；如此循环就形成了上述的参数网络图。

来自 <http://blog.csdn.net/u014540717/article/details/53193433>

来自 <http://blog.csdn.net/u014540717/article/details/53193433>