llama2.c 代码解读，让大模型推理过程不再神秘

风影
292 人赞同了该文章

llama2.c是一个纯c实现llama的推理工程，由openai的Andrej Karpathy 亲自操刀，不依赖任何第三方库就可以进行高效的推理，相比于llama.cpp，代码更通俗易懂。最近花了一天的时间学习了一下，在此对整个工程代码做一个简单的解读。

https://github.com/karpathy/llama2.c
github.com/karpathy/llama2.c

废话不多说，先按照readme跑通整个流程。

```
git clone https://github.com/karpathy/llama2.c.git
cd llama2.c
wget https://huggingface.co/karpathy/tinyllamas/resolve/main/stories15M.bin
make run
./run stories15M.bin
```

```
Once upon a time, there was a boy named Timmy. Timmy loved to play outside
his friends. One day, Timmy saw a butterfly and wanted to catch it. He ask
 friend Billy, "Can you bring me a net?" Billy said, "Sure, I can bring yo
t."
Timmy ran to the shed and found a net. He used the net to catch the butter
illy said, "Wow, you caught a big one!" Timmy said, "Yes, I did it very sl
Suddenly, the butterfly flew away and Timmy felt sad. But then, Billy said
't worry, we can catch it together!" They ran after the butterfly and fina
ught it. Timmy was happy and said, "Thank you, Billy. You are a good frien
ey both smiled and went to play some more. The end.
achieved tok/s: 215.367965
```

接下来，把basemodel替换为chinese-baby-llama2 ，把整个流程走一遍，直接使用作者训练好的baby模型。

1.将hf模型文件转换成.bin文件。

```
python export.py ./models/chinese-baby-llama2.bin --hf ./xxx/chinese-baby-llama2
```

打印模型参数和模型结构如下：

模型参数：

```
ModelArgs(dim=768, n_layers=12, n_heads=12, n_kv_heads=12, vocab_size=32000,
        hidden_dim=2268, multiple_of=256, norm_eps=1e-06, max_seq_len=1024, dropout=0.0)
```

模型结构：

```
Transformer(
  (tok_embeddings): Embedding(32000, 768)
  (dropout): Dropout(p=0.0, inplace=False)
  (layers): ModuleList(
    (0-11): 12 x TransformerBlock(
      (attention): Attention(
        (wq): Linear(in_features=768, out_features=768, bias=False)
        (wk): Linear(in_features=768, out_features=768, bias=False)
        (wv): Linear(in_features=768, out_features=768, bias=False)
        (wo): Linear(in_features=768, out_features=768, bias=False)
        (attn_dropout): Dropout(p=0.0, inplace=False)
        (resid_dropout): Dropout(p=0.0, inplace=False)
      )
      (feed_forward): FeedForward(
        (w1): Linear(in_features=768, out_features=2268, bias=False)
        (w2): Linear(in_features=2268, out_features=768, bias=False)
        (w3): Linear(in_features=768, out_features=2268, bias=False)
        (dropout): Dropout(p=0.0, inplace=False)
      )
      (attention_norm): RMSNorm()
      (ffn_norm): RMSNorm()
    )
  )
  (norm): RMSNorm()
  (output): Linear(in_features=768, out_features=32000, bias=False)
)
```

（1）用pycharm打开工程，debug一下export.py，核心代码见legacy_export()：

```
def legacy_export(model, filepath):  filepath: './models/chinese-baby-llama2/chinese-baby-llama2.bin'  model: Transformer(\n  (tok_embeddin
    """ Original export of llama2.c bin files, i.e. version v0 """
    out_file = open(filepath, 'wb')  out_file: <_io.BufferedWriter name='./models/chinese-baby-llama2/chinese-baby-llama2.bin'>

    # first write out the header
    hidden_dim = model.layers[0].feed_forward.w1.weight.shape[0]  hidden_dim: 2268
    p = model.params  p: ModelArgs(dim=768, n_layers=12, n_heads=12, n_kv_heads=12, vocab_size=32000, hidden_dim=2268, multiple_of=256, norm_
    shared_classifier = torch.equal(model.tok_embeddings.weight, model.output.weight)  shared_classifier: True
    # legacy format uses negative/positive vocab size as a shared classifier flag
    if not shared_classifier:
        p.vocab_size = -p.vocab_size
    n_kv_heads = p.n_heads if p.n_kv_heads is None else p.n_kv_heads  n_kv_heads: 12
    header = struct.pack('iiiiiii', p.dim, hidden_dim, p.n_layers, p.n_heads,n_kv_heads, p.vocab_size, p.max_seq_len)  header: b'\x00\x03\x00
    # ModelArgs(dim=768, n_layers=12, n_heads=12, n_kv_heads=12, vocab_size=32000,
    #           hidden_dim=2268, multiple_of=256, norm_eps=1e-06, max_seq_len=1024, dropout=0.0)
    out_file.write(header) #以字节流的形式保存相关参数信息

    # next write out the embedding weights
    serialize_fp32(out_file, model.tok_embeddings.weight) #保存tok_embeddings层的权重
```

（2）先按照顺序保存7个int类型的模型参数。然后跟到serialize_fp32()中，将权重参数展开为一维，以float32（4个byte）保存到文件中；同理，serialize_int8()以int8（1个byte）保存到文件中：

```
def serialize_fp32(file, tensor):  file: <_io.BufferedWriter name='./models/chinese-baby-llama2/chinese-baby-llam
    """ writes one fp32 tensor to file that is open in wb mode """
    d = tensor.detach().cpu().view(-1).to(torch.float32).numpy()  d: [-4.88891602e-02  3.93066406e-02 -2.76947021
    b = struct.pack(f'{len(d)}f', *d) #一个float32的数占用4个byte  b: b'\x00@H\xbd\x00\x00!=\x00\xe0\xe2\xbc\x00\x8
    file.write(b)

def serialize_int8(file, tensor):
    """ writes one int8 tensor to file that is open in wb mode """
    d = tensor.detach().cpu().view(-1).numpy().astype(np.int8)
    b = struct.pack(f'{len(d)}b', *d) ##一个int8的数占用1个byte
    file.write(b)
```

Evaluate

Expression:

tensor

Use Ctrl+Shift+Enter to add to Watches

Result:

```
result = {Parameter: (32000, 768)} Parameter containing:\ntensor([[-0.0489,  0.0393, -0.0277,  ...,  0.0288 ··· View
  H = {Tensor: (768, 32000)} tensor([[-0.0489,  0.0042, -0.0489,  ..., -0.0563, -0.0404, -0.0488],\n  [ 0.03 ··· View
  T = {Tensor: (768, 32000)} tensor([[-0.0489,  0.0042, -0.0489,  ..., -0.0563, -0.0404, -0.0488],\n  [ 0.03 ··· View
  data = {Tensor: (32000, 768)} tensor([[-0.0489,  0.0393, -0.0277,  ...,  0.0288, -0.1031, -0.0114],\n  [ 0. ··· View
  device = {device} cpu
  dtype = {dtype} torch.float32
```

struct --- 将字节串解读为打包的二进制数据，struct.pack用法见链接。

struct --- 将字节串解读为打包的二进制数据
docs.python.org/zh-cn/3/library/struct.ht...

（3）接下来按顺序保存相应层的权重值（model.layers为12层）：

```
    # now all the layers
    # attention weights
    for layer in model.layers:  layer: TransformerBlock(\n  (attention): Attention(\n    (wq): Linear(in_features=7
        serialize_fp32(out_file, layer.attention_norm.weight)
    for layer in model.layers:
        serialize_fp32(out_file, layer.attention.wq.weight)
    for layer in model.layers:
        serialize_fp32(out_file, layer.attention.wk.weight)
    for layer in model.layers:
        serialize_fp32(out_file, layer.attention.wv.weight)
    for layer in model.layers:
        serialize_fp32(out_file, layer.attention.wo.weight)
    # ffn weights
    for layer in model.layers:
        serialize_fp32(out_file, layer.ffn_norm.weight)
    for layer in model.layers:
        serialize_fp32(out_file, layer.feed_forward.w1.weight)
    for layer in model.layers:
        serialize_fp32(out_file, layer.feed_forward.w2.weight)
    for layer in model.layers:
        serialize_fp32(out_file, layer.feed_forward.w3.weight)
    # final rmsnorm
    serialize_fp32(out_file, model.norm.weight)
    # freqs_cis
    serialize_fp32(out_file, model.freqs_cos[:p.max_seq_len])
    serialize_fp32(out_file, model.freqs_sin[:p.max_seq_len])
```

以上为导出权重信息的整个流程，简单来说就是从torch中取出每一层的权重值，然后按照自己定义的格式保存下来，以便推理工程使用，举个例子，你在torch中有一个2*3的权重值，[[1.0,2.0,3.0]，[4.0,5.0,6.0]]，type=f32，按照上面流程，参数保存顺序为:

```
2(4)3(4)1.0(4)2.0(4)~~6.0(4)
```

括号内为所占用的字节数，整个参数所占用内存大小为32byte。在读取的时候，按照顺序读取相关权重值，然后reshape到相应的维度，就可以进行相关计算了。

（4）同理tokenizer.py也是一样的流程。

2.用clion打开llama2.c工程，以便debug。

（1）新增一个cmakelists.txt文件，debug代码的时候，使用debug模式，不开启编译优化（-O0）；运行的时候，使用release模式，编译选项开启-O3优化；

run.c CMakeLists.txt

```
cmake_minimum_required(VERSION 3.19)
project(llama2.c)

set(CMAKE_BUILD_TYPE debug)  # Debug  Release
set(CMAKE_MODULE_PATH ${CMAKE_MODULE_PATH} "${CMAKE_SOURCE_DIR}/")
set(CMAKE_CXX_STANDARD 14)

SET(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -O0 -ffast-math -march=native -fopenmp -mavx2 -mfma -DEIGEN_STACK_ALLOCATION_LIMIT=(
SET(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -O0 -ffast-math -march=native -fopenmp -mavx2 -mfma -DEIGEN_STACK_ALLOCATION_LI

add_executable(run run.c)
target_link_libraries(run -lpthread -lm -ldl -m64 -lpthread)
```

（2）找到main()，baby使用的网络小说数据集训练的模型，修改prompt，把相关输入写死方便debug：

```
int main(int argc, char *argv[]) {  argv: 0x7fffd661fb08   argc: 2

    // default parameters
    char *checkpoint_path = NULL;  // e.g. out/model.bin   checkpoint_path: NULL

    float temperature = 1.0f;   // 0.0 = greedy deterministic. 1.0 = original. don't set higher   temperature: 1
    float topp = 0.9f;          // top-p in nucleus sampling. 1.0 = off. 0.9 works well, but slower   topp: 0.899999976
    int steps = 256;            // number of steps to run for   steps: 256
//    char *prompt = NULL;        // prompt string
    char *prompt = "今天是武林大会，我是武林盟主。";   prompt: "今天是武林大会，我是武林盟主。"
    unsigned long long rng_seed = 0; // seed rng with time by default   rng_seed: 0
    char *mode = "generate";    // generate|chat   mode: "generate"
    char *system_prompt = NULL; // the (optional) system prompt to use in chat mode   system_prompt: NULL
    argc = 2;   argc: 2

//    char *tokenizer_path = "/home/wqt/software/clion-2021.1.3/work/llama2.c/models/tokenizer.bin";
//    argv[1] = "/home/wqt/software/clion-2021.1.3/work/llama2.c/models/stories15M.bin";

    char *tokenizer_path = "/home/wqt/software/clion-2021.1.3/work/llama2.c/models/chinese-baby-llama2/tokenizer.bin";   tokenizer_pa
    argv[1] = "/home/wqt/software/clion-2021.1.3/work/llama2.c/models/chinese-baby-llama2/chinese-baby-llama2.bin";   argv: 0x7fffd66
```

（3）继续debug，先看下Transformer和TransformerWeights的数据结构：

```
typedef struct {
    Config config; // the hyperparameters of the architecture (the blueprint
    TransformerWeights weights; // the weights of the model
    RunState state; // buffers for the "wave" of activations in the forward
    // some more state needed to properly clean up the memory mapping (sigh)
    int fd; // file descriptor for memory mapping
    float* data; // memory mapped data pointer
    ssize_t file_size; // size of the checkpoint file in bytes
} Transformer;
```

```c
typedef struct {
    // token embedding table
    float* token_embedding_table;    // (vocab_size, dim)
    // weights for rmsnorms
    float* rms_att_weight; // (layer, dim) rmsnorm weights
    float* rms_ffn_weight; // (layer, dim)
    // weights for matmuls. note dim == n_heads * head_size
    float* wq; // (layer, dim, n_heads * head_size)
    float* wk; // (layer, dim, n_kv_heads * head_size)
    float* wv; // (layer, dim, n_kv_heads * head_size)
    float* wo; // (layer, n_heads * head_size, dim)
    // weights for ffn
    float* w1; // (layer, hidden_dim, dim)
    float* w2; // (layer, dim, hidden_dim)
    float* w3; // (layer, hidden_dim, dim)
    // final rmsnorm
    float* rms_final_weight; // (dim,)
    // (optional) classifier weights for the logits, on the last layer
    float* wcls;
} TransformerWeights;
```

进入build_transformer()，调用另外两个函数read_checkpoint()和malloc_run_state()：

```c
void build_transformer(Transformer *t, char* checkpoint_path) {
    // read in the Config and the Weights from the checkpoint
    read_checkpoint(checkpoint_path, &t->config, &t->weights, &t->fd, &t->data, &t->file_s
    // allocate the RunState buffers
    malloc_run_state(&t->state, &t->config);
}
```

跟到read_checkpoint()中，sizeof(Config)是export.py中保存的7个int类型的模型参数值；

```
void read_checkpoint(char* checkpoint, Config* config, TransformerWeights* weights,  checkpoint: "/home/wqt/software/clion-20
                     int* fd, float** data, ssize_t* file_size) {  fd: 0x7fffd661f7a0  data: 0x7fffd661f7a8  file_size: 0x
    FILE *file = fopen(checkpoint,  modes: "rb"); //打开export.py中保存的模型权重文件  file: 0x5572726ea2c0
    if (!file) { fprintf(stderr,  format: "Couldn't open file %s\n", checkpoint); exit(EXIT_FAILURE); }  checkpoint: "/home/wqt/
    // read in the config header
    if (fread(config, sizeof(Config),  n: 1, file) != 1) { exit(EXIT_FAILURE); }  config: 0x7fffd661f6c0  file: 0x5572726ea2
    // negative vocab size is hacky way of signaling unshared weights. bit yikes.
    int shared_weights = config->vocab_size > 0 ? 1 : 0;
    config->vocab_size = abs(config->vocab_size);
    // figure out the file size
    fseek(file,  off: 0, SEEK_END); // move file pointer to end of file
    *file_size = ftell(file); // get the file size, in bytes
    fclose(file);
    // memory map the Transformer weights into the data pointer
    *fd = open(checkpoint, O_RDONLY); // open in read only mode
    if (*fd == -1) { fprintf(stderr,  format: "open failed!\n"); exit(EXIT_FAILURE); }
    *data = mmap( addr: NULL, *file_size, PROT_READ, MAP_PRIVATE, *fd,  offset: 0);
    if (*data == MAP_FAILED) { fprintf(stderr,  format: "mmap failed!\n"); exit(EXIT_FAILURE); }
    float* weights_ptr = *data + sizeof(Config)/sizeof(float);
    memory_map_weights(weights, config, weights_ptr, shared_weights);//读取每一层的权重值到weights中
}
```

Expression:

```
sizeof(Config)
```

Result:

```
01 result = {int} 28
```

跟到memory_map_weights()中，按顺序依次读取保存的权重信息，权重值读取的顺序、个数以及数据类型与export.py中的保存顺序、个数和数据类型一致，读取的权重信息存到*w中；

```
void memory_map_weights(TransformerWeights *w, Config* p, float* ptr, int shared_weights) {  ptr: 0x7fd1bbb5701c
    int head_size = p->dim / p->n_heads;  p: 0x7ffe20a08b90
    // make sure the multiplications below are done in 64bit to fit the parameter counts of 13B+ models
    unsigned long long n_layers = p->n_layers;
    w->token_embedding_table = ptr;
    ptr += p->vocab_size * p->dim;
    w->rms_att_weight = ptr;
    ptr += n_layers * p->dim;
    w->wq = ptr;
    ptr += n_layers * p->dim * (p->n_heads * head_size);
    w->wk = ptr;
    ptr += n_layers * p->dim * (p->n_kv_heads * head_size);
    w->wv = ptr;
    ptr += n_layers * p->dim * (p->n_kv_heads * head_size);
    w->wo = ptr;
    ptr += n_layers * (p->n_heads * head_size) * p->dim;
    w->rms_ffn_weight = ptr;
    ptr += n_layers * p->dim;
    w->w1 = ptr;
    ptr += n_layers * p->dim * p->hidden_dim;
    w->w2 = ptr;
    ptr += n_layers * p->hidden_dim * p->dim;
    w->w3 = ptr;
    ptr += n_layers * p->dim * p->hidden_dim;
    w->rms_final_weight = ptr;
    ptr += p->dim;
    ptr += p->seq_len * head_size / 2; // skip what used to be freq_cis_real (for RoPE)
    ptr += p->seq_len * head_size / 2; // skip what used to be freq_cis_imag (for RoPE)
    w->wcls = shared_weights ? w->token_embedding_table : ptr;
}
```

跟到malloc_run_state()中，初始化相关变量，*s用于存储中间层的计算结果；

```
void malloc_run_state(RunState* s, Config* p) {   s: 0x7ffe20a08c10    p: 0x7ffe20a08b90
    // we calloc instead of malloc to keep valgrind happy
    int kv_dim = (p->dim * p->n_kv_heads) / p->n_heads;   p: 0x7ffe20a08b90
    s->x = calloc(p->dim, sizeof(float));
    s->xb = calloc(p->dim, sizeof(float));
    s->xb2 = calloc(p->dim, sizeof(float));
    s->hb = calloc(p->hidden_dim, sizeof(float));
    s->hb2 = calloc(p->hidden_dim, sizeof(float));
    s->q = calloc(p->dim, sizeof(float));
    s->key_cache = calloc( nmemb: p->n_layers * p->seq_len * kv_dim, sizeof(float));
    s->value_cache = calloc( nmemb: p->n_layers * p->seq_len * kv_dim, sizeof(float));
    s->att = calloc( nmemb: p->n_heads * p->seq_len, sizeof(float));
    s->logits = calloc(p->vocab_size, sizeof(float));
    // ensure all mallocs went fine
    if (!s->x || !s->xb || !s->xb2 || !s->hb || !s->hb2 || !s->q
     || !s->key_cache || !s->value_cache || !s->att || !s->logits) {
        fprintf(stderr, format: "malloc failed!\n");
        exit(EXIT_FAILURE);
    }
}
```

```
typedef struct {
    // current wave of activations
    float *x; // activation at current time stamp (dim,)
    float *xb; // same, but inside a residual branch (dim,)
    float *xb2; // an additional buffer just for convenience (dim,)
    float *hb; // buffer for hidden dimension in the ffn (hidden_dim,)
    float *hb2; // buffer for hidden dimension in the ffn (hidden_dim,)
    float *q; // query (dim,)
    float *k; // key (dim,)
    float *v; // value (dim,)
    float *att; // buffer for scores/attention values (n_heads, seq_len)
    float *logits; // output logits
    // kv cache
    float* key_cache;   // (layer, seq_len, dim)
    float* value_cache; // (layer, seq_len, dim)
} RunState;
```

（4）进入build_tokenizer()，读取tokenizer的相关信息；

```
void build_tokenizer(Tokenizer* t, char* tokenizer_path, int vocab_size) {  t: 0x7ffe20a08c90   tokenizer_path: "/home/wqt/software/
    // i should have written the vocab_size into the tokenizer file... sigh
    t->vocab_size = vocab_size;
    // malloc space to hold the scores and the strings
    t->vocab = (char**)malloc( size: vocab_size * sizeof(char*));
    t->vocab_scores = (float*)malloc( size: vocab_size * sizeof(float));  vocab_size: 32000
    t->sorted_vocab = NULL; // initialized lazily  t: 0x7ffe20a08c90
    for (int i = 0; i < 256; i++) {
        t->byte_pieces[i * 2] = (unsigned char)i;
        t->byte_pieces[i * 2 + 1] = '\0';
    }
    // read in the file
    FILE *file = fopen(tokenizer_path,  modes: "rb");
    if (!file) { fprintf(stderr,  format: "couldn't load %s\n", tokenizer_path); exit(EXIT_FAILURE); }
    if (fread(&t->max_token_length, sizeof(int),  n: 1, file) != 1) { fprintf(stderr,  format: "failed read\n"); exit(EXIT_FAILURE); }
    int len;
    for (int i = 0; i < vocab_size; i++) {
        if (fread( ptr: t->vocab_scores + i, sizeof(float),  n: 1, file) != 1) { fprintf(stderr,  format: "failed read\n"); exit(EXIT_FAI
        if (fread(&len, sizeof(int),  n: 1, file) != 1) { fprintf(stderr,  format: "failed read\n"); exit(EXIT_FAILURE); }
        t->vocab[i] = (char *)malloc( size: len + 1);
        if (fread(t->vocab[i], len,  n: 1, file) != 1) { fprintf(stderr,  format: "failed read\n"); exit(EXIT_FAILURE); }
        t->vocab[i][len] = '\0'; // add the string terminating token
    }
    fclose(file);
}
```

```
// ----------------------------------------------------------------------------
// The Byte Pair Encoding (BPE) Tokenizer that translates strings <-> tokens

typedef struct {
    char *str;
    int id;
} TokenIndex;

typedef struct {
    char** vocab;
    float* vocab_scores;
    TokenIndex *sorted_vocab;
    int vocab_size;
    unsigned int max_token_length;
    unsigned char byte_pieces[512]; // stores all single-byte strings
} Tokenizer;
```

（5）接下来进入到generate()函数中，先对输入的prompt进行encode，然后进行自回归生成过程，生成过程分为两个阶段：预填充(prefill)和解码(decoding)，当达到停止条件的时候，结束生成；

```c
void generate(Transformer *transformer, Tokenizer *tokenizer, Sampler *sampler, char *prompt, int steps) {
    char *empty_prompt = "";
    if (prompt == NULL) { prompt = empty_prompt; }

    // encode the (string) prompt into tokens sequence
    int num_prompt_tokens = 0;
    int* prompt_tokens = (int*)malloc( size: (strlen(prompt)+3) * sizeof(int)); // +3 for '\0', ?BOS, ?EOS
    encode(tokenizer, prompt, bos: 1, eos: 0, prompt_tokens, &num_prompt_tokens); //对prompt进行encoder
    if (num_prompt_tokens < 1) {
        fprintf(stderr, format: "something is wrong, expected at least 1 prompt token\n");
        exit(EXIT_FAILURE);
    }

    // start the main loop
    long start = 0;  // used to time our code, only initialized after first iteration
    int next;        // will store the next token in the sequence
    int token = prompt_tokens[0]; // kick off with the first token in the prompt
    int pos = 0;     // position in the sequence
    while (pos < steps) { //自回归生成过程
        // forward the transformer to get logits for the next token
        float* logits = forward(transformer, token, pos); //前向推理过程
        // advance the state machine
        if (pos < num_prompt_tokens - 1) {
            // if we are still processing the input prompt, force the next prompt token
            next = prompt_tokens[pos + 1]; //阶段1：prefill，属于计算密集型过程，以并行方式处理输入的prompt
        } else {
            // otherwise sample the next token from the logits
            next = sample(sampler, logits); //阶段2：decoding，属于io密集型过程，以自回归的方式逐个生成预测的token
        }
        pos++;
        // data-dependent terminating condition: the BOS (=1) token delimits sequences
        if (next == 1) { break; }
        // print the token as string, decode it with the Tokenizer object
        char* piece = decode(tokenizer, token, next);
        safe_printf(piece); // same as printf("%s", piece), but skips "unsafe" bytes
        fflush(stdout);
        token = next;
        // init the timer here because the first iteration can be slower
        if (start == 0) { start = time_in_ms(); }
    }
    printf( format: "\n");
```

跟到forward()中，整个工程最核心的地方，开始做前向计算，是最耗时的地方，也是最需要优化的地方；对比c和py代码一起看；

```c
float* forward(Transformer* transformer, int token, int pos) {

    // a few convenience variables
    Config* p = &transformer->config;
    TransformerWeights* w = &transformer->weights;
    RunState* s = &transformer->state;
    float *x = s->x;
    int dim = p->dim;
```

```
int kv_dim = (p->dim * p->n_kv_heads) / p->n_heads;
int kv_mul = p->n_heads / p->n_kv_heads; // integer multiplier of the kv sharing in multiquery
int hidden_dim =  p->hidden_dim;
int head_size = dim / p->n_heads;

// copy the token embedding into x
float* content_row = w->token_embedding_table + token * dim; //计算prompt的token_embedding
memcpy(x, content_row, dim*sizeof(*x)); //将输入copy到*x

// forward all the layers
for(unsigned long long l = 0; l < p->n_layers; l++) {

    // attention rmsnorm
    rmsnorm(s->xb, x, w->rms_att_weight + l*dim, dim); //先计算rmsnorm，结果存到s->xb中，计算过程与构建model的时候

    // key and value point to the kv cache
    int loff = l * p->seq_len * kv_dim; // kv cache layer offset for convenience
    s->k = s->key_cache + loff + pos * kv_dim; //获取kv-cache
    s->v = s->value_cache + loff + pos * kv_dim;

    // qkv matmuls for this position
    matmul(s->q, s->xb, w->wq + l*dim*dim, dim, dim); //输入是上一层的输出，s->xb，得到qkv的结果，
    matmul(s->k, s->xb, w->wk + l*dim*kv_dim, dim, kv_dim);
    matmul(s->v, s->xb, w->wv + l*dim*kv_dim, dim, kv_dim);

    // RoPE relative positional encoding: complex-valued rotate q and k in each head
    for (int i = 0; i < dim; i+=2) {
        int head_dim = i % head_size;
        float freq = 1.0f / powf(10000.0f, head_dim / (float)head_size);
        float val = pos * freq;
        float fcr = cosf(val);
        float fci = sinf(val);
        int rotn = i < kv_dim ? 2 : 1; // how many vectors? 2 = q & k, 1 = q only
        for (int v = 0; v < rotn; v++) {
            float* vec = v == 0 ? s->q : s->k; // the vector to rotate (query or key)
            float v0 = vec[i];
            float v1 = vec[i+1];
            vec[i]   = v0 * fcr - v1 * fci;
            vec[i+1] = v0 * fci + v1 * fcr;
        }
    }

    // multihead attention. iterate over all heads
    int h;
    //omp并行计算，debug的时候建议注释掉
    #pragma omp parallel for private(h)
    for (h = 0; h < p->n_heads; h++) {
        // get the query vector for this head
        float* q = s->q + h * head_size;
        // attention scores for this head
        float* att = s->att + h * p->seq_len;
        // iterate over all timesteps, including the current one
        for (int t = 0; t <= pos; t++) {
```

```
            // get the key vector for this head and at this timestep
            float* k = s->key_cache + loff + t * kv_dim + (h / kv_mul) * head_size;
            // calculate the attention score as the dot product of q and k
            float score = 0.0f;
            for (int i = 0; i < head_size; i++) {
                score += q[i] * k[i];
            }
            score /= sqrtf(head_size);
            // save the score to the attention buffer
            att[t] = score; //得到attention结果
        }

        // softmax the scores to get attention weights, from 0..pos inclusively
        softmax(att, pos + 1);

        // weighted sum of the values, store back into xb
        float* xb = s->xb + h * head_size;
        memset(xb, 0, head_size * sizeof(float));
        for (int t = 0; t <= pos; t++) {
            // get the value vector for this head and at this timestep
            float* v = s->value_cache + loff + t * kv_dim + (h / kv_mul) * head_size;
            // get the attention weight for this timestep
            float a = att[t];
            // accumulate the weighted value into xb
            for (int i = 0; i < head_size; i++) {
                xb[i] += a * v[i];
            }
        }
    }

    // final matmul to get the output of the attention
    matmul(s->xb2, s->xb, w->wo + l*dim*dim, dim, dim); //得到最终的attention_output

    // residual connection back into x
    for (int i = 0; i < dim; i++) {
        x[i] += s->xb2[i]; //attention层输出的残差连接
    }

    // ffn rmsnorm
    rmsnorm(s->xb, x, w->rms_ffn_weight + l*dim, dim);

    // Now for FFN in PyTorch we have: self.w2(F.silu(self.w1(x)) * self.w3(x))
    // first calculate self.w1(x) and self.w3(x)
    matmul(s->hb, s->xb, w->w1 + l*dim*hidden_dim, dim, hidden_dim); //三个nn.Linear的ffn层
    matmul(s->hb2, s->xb, w->w3 + l*dim*hidden_dim, dim, hidden_dim);

    // SwiGLU non-linearity
    for (int i = 0; i < hidden_dim; i++) {
        float val = s->hb[i];
        // silu(x)=x*σ(x), where σ(x) is the logistic sigmoid
        val *= (1.0f / (1.0f + expf(-val)));
        // elementwise multiply with w3(x)
        val *= s->hb2[i];
```

```
            s->hb[i] = val;
        }

        // final matmul to get the output of the ffn
        matmul(s->xb, s->hb, w->w2 + l*dim*hidden_dim, hidden_dim, dim);

        // residual connection
        for (int i = 0; i < dim; i++) {
            x[i] += s->xb[i]; //ffn层输出的残差连接
        }
    }

    // final rmsnorm
    rmsnorm(x, x, w->rms_final_weight, dim);

    // classifier into logits
    matmul(s->logits, x, w->wcls, p->dim, p->vocab_size); //transformer的最终输出结果，保存到s->logits中
    return s->logits;
}
```

```
    def forward(self, tokens: torch.Tensor, targets: Optional[torch.Tensor] = None) -> torch.Tensor:
        _bsz, seqlen = tokens.shape
        h = self.tok_embeddings(tokens)
        h = self.dropout(h)
        freqs_cos = self.freqs_cos[:seqlen]
        freqs_sin = self.freqs_sin[:seqlen]

        for layer in self.layers:
            h = layer(h, freqs_cos, freqs_sin)
        h = self.norm(h)

        if targets is not None:
            # if we are given some desired targets also calculate the loss
            logits = self.output(h)
            self.last_loss = F.cross_entropy(logits.view(-1, logits.size(-1)), targets.view(-1), ignore_index=-1
        else:
            # inference-time mini-optimization: only forward the output on the very last position
            logits = self.output(h[:, [-1], :]) # note: using list [-1] to preserve the time dim
            self.last_loss = None

        return logits
```

```python
class FeedForward(nn.Module):
    def __init__(self, dim: int, hidden_dim: int, multiple_of: int, dropout: float):
        super().__init__()
        if hidden_dim is None:
            hidden_dim = 4 * dim
            hidden_dim = int(2 * hidden_dim / 3)
            hidden_dim = multiple_of * ((hidden_dim + multiple_of - 1) // multiple_of)
        self.w1 = nn.Linear(dim, hidden_dim, bias=False)
        self.w2 = nn.Linear(hidden_dim, dim, bias=False)
        self.w3 = nn.Linear(dim, hidden_dim, bias=False)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x):
        return self.dropout(self.w2(F.silu(self.w1(x)) * self.w3(x)))


class TransformerBlock(nn.Module):
    def __init__(self, layer_id: int, args: ModelArgs):
        super().__init__()
        self.n_heads = args.n_heads
        self.dim = args.dim
        self.head_dim = args.dim // args.n_heads
        self.attention = Attention(args)
        self.feed_forward = FeedForward(
            dim=args.dim,
            hidden_dim=args.hidden_dim,
            multiple_of=args.multiple_of,
            dropout=args.dropout,
        )
        self.layer_id = layer_id
        self.attention_norm = RMSNorm(args.dim, eps=args.norm_eps)
        self.ffn_norm = RMSNorm(args.dim, eps=args.norm_eps)

    def forward(self, x, freqs_cos, freqs_sin):
        h = x + self.attention.forward(self.attention_norm(x), freqs_cos, freqs_sin)
        out = h + self.feed_forward.forward(self.ffn_norm(h))
        return out
```

跟到matmul()和softmax()中，都是最朴实的实现过程，待优化的地方（gemm优化、avx2指令、超越函数近似计算-泰勒展开、使用成熟高效的数值计算库mkl、blas、eigen3等），参考声码器WaveRnn推理性能优化实践；

```c
void softmax(float* x, int size) {
    // find max value (for numerical stability)
    float max_val = x[0];
    for (int i = 1; i < size; i++) {
        if (x[i] > max_val) {
            max_val = x[i];
        }
    }
    // exp and sum
    float sum = 0.0f;
    for (int i = 0; i < size; i++) {
        x[i] = expf( x: x[i] - max_val);
        sum += x[i];
    }
    // normalize
    for (int i = 0; i < size; i++) {
        x[i] /= sum;
    }
}

void matmul(float* xout, float* x, float* w, int n, int d) {
    // W (d,n) @ x (n,) -> xout (d,)
    // by far the most amount of time is spent inside this little function
    int i;
    #pragma omp parallel for private(i)
    for (i = 0; i < d; i++) {
        float val = 0.0f;
        for (int j = 0; j < n; j++) {
            val += w[i * n + j] * x[j];
        }
        xout[i] = val;
    }
}
```

跟到sample()中，对结果进行temperature、topp、topk处理；

```c
int sample(Sampler* sampler, float* logits) {
    // sample the token given the logits and some hyperparameters
    int next;
    if (sampler->temperature == 0.0f) {
        // greedy argmax sampling: take the token with the highest probability
        next = sample_argmax(logits, sampler->vocab_size);
    } else {
        // apply the temperature to the logits
        for (int q=0; q<sampler->vocab_size; q++) { logits[q] /= sampler->temperature; }
        // apply softmax to the logits to get the probabilities for next token
        softmax(logits, sampler->vocab_size);
        // flip a (float) coin (this is our source of entropy for sampling)
        float coin = random_f32(&sampler->rng_state);
        // we sample from this distribution to get the next token
        if (sampler->topp <= 0 || sampler->topp >= 1) {
            // simply sample from the predicted probability distribution
            next = sample_mult(logits, sampler->vocab_size, coin);
        } else {
            // top-p (nucleus) sampling, clamping the least likely tokens to zero
            next = sample_topp(logits, sampler->vocab_size, sampler->topp, sampler->probindex, coi
        }
    }
    return next;
}
```

计算完成后，释放内存；

```c
// run!
if (strcmp(mode, "generate") == 0) {
    generate(&transformer, &tokenizer, &sampler, prompt, steps);
} else if (strcmp(mode, "chat") == 0) {
    chat(&transformer, &tokenizer, &sampler, prompt, system_prompt, steps);
} else {
    fprintf(stderr,  format: "unknown mode: %s\n", mode);
    error_usage();
}

// memory and file handles cleanup
free_sampler(&sampler);
free_tokenizer(&tokenizer);
free_transformer(&transformer);
return 0;
```

至此，整个llama2的推理流程全部结束，最后看一下生成的小说的效果。

```
(base) wqt@wqt:~/software/clion-2021.1.3/work/llama2.c/cmake-build-debug$ ./run
今天是武林大会，我是武林盟主。我的飞行速度平静，应该远在千年之前，以及近几个月的历史，只需
天的靠岸三天，不难飞离这里，离开门派。除了那些修炼一生所历能事，仍是先想手中的偷天弓拐扫，
一带还有那么点荒漠，我心里七上八下，看来自己确实不能飞离开的。“兄弟，不要怕，我能够让你飞
峰里嘛。”之极心烦意乱，一扫方才的狼狈和飘渺，竟然像个小孩子一样调皮，更加卖力的护着我。哎，
为什么，傻木虽然天性善良，而且还是个活泼的小孩子，但人小时候可是活泼的，就是一老大。什么都
犹如是个小孩，什么都不做，很多时候，也都会耍无赖，耍战术，贪生怕死的得寸进尺，连认输那更重
点都不惜。这个时候，正是我阳光正美的时候，阳光正好打在我雨伞身上，不仅他打个呵欠，还朝着我
肢解了过来，花枝乱颤，料想八成是做了很久没有洗干净的澡，饿了回家时，我们喝了几大口稀粥，早
把感动的心中之火给点燃，恨不得马上就睡掉。“
achieved tok/s: 29.947152
(base) wqt@wqt:~/software/clion-2021.1.3/work/llama2.c/cmake-build-debug$ ./run
今天是武林大会，我是武林盟主。见多识广，知道苦行僧就是我，你今次就是受主人之命，来接我们守
，所以我再问你，今天老衲要见你。”“不问可知，佛家与佛家打着什么交道？”于是我回答道。“别的不
便宜主人的说法，你都知道。”左边那人淡淡说了一句。感激的看着我，右边那人了一口气，再回道。“
是将他们全杀了？”我奇怪的望向右边那个不出声的黑发中年人，问道。“是的，神秘师姐，我们是选三
人手不够，动手强盗本就是不该，至于留我们在这里做什么，就这么定了，错了反而更狠。”左边的中年
，向前方径直向前走去，在他刚才往我们这里走时，就清楚了他们的打算。当然，自己这里也有擅长炼
药师，至于炼制药草出来，反正他们药草并不是特别珍贵，你就差些炼制一些，让他们发挥大部分的功
当然，只要在他们内部的一副山谷里开辟一个场地，就足够我们掩饰了。拍了拍身上渐显从今长的嘴角
下了然，想不到七年前，竟然又有人能炼制出有效的丹药。事情进展得很快，十分钟后，我们带来的这
也已到达得很晚，在这短暂的收猎
achieved tok/s: 14.286515
```

学会后，你也可以基于x86平台手撸一个神经网络推理框架了。

最后，安利一波ncnn，炒鸡好用，代码可读性很强，用来学习神经网络推理优化非常适合。

reference

github.com/karpathy/lla...

github.com/DLLXW/baby-l...

flyingfishinwater/chinese-baby-llama2 · Hugging Face

github.com/RahulSChand/...

github.com/tpoisonooo/h...

github.com/Tencent/ncnn

struct --- 将字节串解读为打包的二进制数据