



WITMEM 语音芯片工具链 用户使用说明

版本号：V1.3.0

日期：2024.7.29

声明

商标声明：



作为北京知存科技有限公司的商标，本文件中提到的

所有其他商标和商号均为其持有人的财产。

版权声明：

Copyright © 2021 北京知存科技有限公司. All rights reserved.

内容声明：

本文件中的信息如有更改，恕不另行通知。为了确保内容的准确性，文章会做出相关的确认，但本文件中的所有声明、信息和建议不构成任何明示或暗示的保证。

目录

1 环境配置	5
1.1 简介	5
1.2 前端支持	5
1.3 版本信息	5
1.4 安装方式	5
1.4.1 Docker 方式	5
2 工程说明	6
3 使用流程	7
3.1 模型导入	7
3.2 编译 BUILD	7
3.3 运行 RUN	8
3.4 与 TENSORFLOW 结果进行比对	8
3.5 与 PYTORCH 结果进行比对	9
3.6 多网络	10
3.7 优化策略	10
3.7.1 输入数据优化	12
3.7.2 权重数据优化	13
3.7.3 权重数据复制	14
3.7.4 稀疏矩阵优化	14
3.7.5 输出放大优化	15
3.7.6 卷积优化	15
3.8 特殊配置环境变量	16
3.8.1 Array 分配	16
3.8.2 开启 FIFO	16
3.8.3 设置校准数据数量	17
4 ONNX 模型	18
4.1 创建 TENSOR	18
4.2 ONNX 算子	18
4.2.1 Gemm	19
4.2.2 Tdnn	19
4.2.3 Lstm	20
4.2.4 Gru	21
4.2.5 Scale	22
4.2.6 Relu	22
4.2.7 ActLut	22
4.2.8 Mul	23
4.2.9 Add	23

4.2.10 Averagepool	24
4.2.11 Slice	24
4.2.12 Concat	25
4.2.13 Conv	25
4.2.14 ConvTranspose	25
4.2.15 Gru2Array	26
4.3 创建 GRAPH	27
4.4 保存 GRAPH	28
4.5 导入模型	28
5 常见问题和说明	29
5.1 ARRAY 分配 ERROR	29
5.2 REGFILE 空间不足	29
5.3 SRAM 空间不足	30
5.4 输入类型错误	30

1 环境配置

1.1 简介

本文主要介绍语音芯片工具链 `witin_mapper` 的使用。该工具链主要针对各种神经网络设计，用于将神经网络运算 `map` 到 WTM1001/WTM2101 芯片上，生成相应的配置文件和烧录文件。其中烧录文件需使用专门的工具烧写到 WTM1001 和 WTM2101 的 ARRAY 阵列中，配置文件集成到 `sdk` 中使用。

1.2 前端支持

前端转换支持 `tensorflow`, `onnx`, `pytorch` 深度学习框架。

1.3 版本信息

Ubuntu:18.04.5

Cmake 3.16 (18.04 默认是 3.10 版本，需升级)

1.4 安装方式

1.4.1 Docker 方式

方式一：

预先在 windows 系统或者 Liunx 系统下安装 docker 环境。使用 Dockerfile 构建镜像

1 构建镜像：`docker build -t witin/witin_tc_wtm2101_vX.X.X .` (Dockerfile 目录下运行)

2.查看镜像：`docker images`

3.创建容器：

`docker run -it --name xxx witin_tc_witin_toolchain_wtm2101:vX.X.X /bin/bash`

4.查看容器：`docker ps -a`

5 重新进入容器：1)`docker start container_id` ; 2)`docker attach container_id`

6 退出容器：`exit`

7 将本地文件（或文件夹）上传至容器：

`docker cp 本地文件路径 container_id:/workspace/witin_mapper`

8 将容器文件（或文件夹）下载至本地：

`docker cp container_id:/workspace/witin_mapper 本地文件路径`

10.使用 witin mapper 生成的网络的映射配置，一般存放于 witin_mapper/output 路径下，可以使用 docker cp 将其从 docker 容器内拷贝到本地，以供后续步骤的烧录工具和精度分析工具使用。

方式二：

1.拉取镜像：

`docker pull witin/witin_toolchain_wtm2101:vX.X.X`

2.查看镜像：

`docker images`

3.创建容器：

`docker run -it --name xxx witin/witin_toolchain_wtm2101:vX.X.X /bin/bash`

4.查看容器：

`docker ps -a`

5.启动容器

`docker start container_id`

`docker attach container_id`

6.进入容器,默认进入 workspace 路径

7.如若不用，退出容器，执行 exit 即可。

说明：2024 年 6 月份后，中国大陆境内使用方式 2 获取 docker 镜像可能会失败。

2 工程说明

build：存放安装脚本，output 输出文件目录以及 WTM1001/WTM2101 模拟器

model：存放示例的模型文件

python：存放整个工程的 python 文件

tests：部分测试文件

run.py：利用 python3 命令执行部分 tests 文件中的测试文件

README.md：对整个 witin_mapper 工程进行说明

3 使用流程

3.1 模型导入

以 tensorflow 为例，调用接口函数如下：

1. 首先调用 tensorflow 接口函数来读取 pb 文件得到 graph_def
with gfile.GFile(model_path, 'rb') as f:

```
graph_def=tf.GraphDef()  
graph_def.ParseFromString(f.read())  
sess.graph.as_default()  
tf.import_graph_def(graph_def, name="")
```

2. 调用前端转换函数

```
witin_frontend.frontend.from_tensorflow(graph_def, shape_dict)
```

其中 graph_def 为读取 pb 文件得到的文件，shape_dict 为输入节点的名字和形状，字典形式。调用该函数后得到 mod 和 params 两个参数，其中 mod 为图的结构中间表示，params 为用到的参数。

3.2 编译 build

该步骤通过编译得到 WTM1001/WTM2101 上运行的配置文件，然后将该配置文件集成到 sdk 然后下载到芯片中并运行，即可完成整个网络的在 WTM1001/WTM2101 芯片上的运行。

首先配置 target 为 npu，target_host 为 npu，然后调用如下接口函数：

```
with witin.transform.PassContext(opt_level=3):  
    _, _, _, npu_graph =  
witin_frontend.build_module.build(  
    mod,  
    target='npu',  
    target_host='npu',  
    params=params,  
    input_data=input_dt,  
    chip="BB04P1",  
    optimize_method_config=optimize_method_config,  
    output_dir=output_dir,  
    array_distribute=array_distribute)
```

其中：

mod 为 2.1 中模型导入后得到的 mod 表示

target 和 target_host 为 npu

params 为模型导入后得到的 params 文件
input_data 参数为 witin 阵列 Array 校准需要用到的输入数据，推荐至少为 100 组输入。

chip 为使用的芯片版本。

optimize_method_config 为优化配置文件，默认为空字符串""。

output_dir 为输出文件的保存路径，默认为"./build/output"。

array_distribute 为指定权重布局分配的文件，默认为空字符串""，当指定此文件时，会最小化 build 模型，只保留一些必要文件。

在调用 build 后，在工程的 build 的 output 文件夹下会生成如下几个文件和文件夹：

1. net_config.h/net_config.c 文件，配置文件，需集成到 witin 的 sdk 中使用
2. map 文件夹，存放烧写 array 需要用到的 expected_in.bin，expected_out.bin，layers.txt 以及 map.csv 文件。
3. simulator_input 文件夹：存放模拟器仿真输入文件
4. simulator_output 文件夹：存放运行仿真后的输出文件
5. params 文件夹：存放各层映射的权重文件，用于 debug
6. layer_debug 文件夹：存放各层仿真器的输出输出文件，用于 debug 及精度调试工具。

3.3 运行 run

该部分主要提供 build 之后在 witin 的模拟器进行运行的接口。

build 完成后会在 build 的 output 文件下生成配置文件和烧写文件。同时会返回 npu_graph 到 python 端。该 npu_graph 可用于创建 runtime 的 module，然后打注输入并运行，最后得到网络运行的结果。具体函数接口如下：

- m=npu_graph_runtime.create(npu_graph, "BB04P"), 创建运行时 module m。
- m.set_input('in1', wintin.nd.array(data)), 其中 data 为 np.array 的列表, 'in1' 为 input placeholder 的名字。
- m.run(), 在模拟器上运行。
- m.get_output(0), 得到在模拟器上运行的结果。

3.4 与 tensorflow 结果进行比对

设置阈值 threshold，调用 compare_tensorflow_with_npu 函数比较 tensorflow 的运行结果 tf_out_data 和 witin npu 的运行结果。

3.5 与 pytorch 结果进行比对

(1)构建网络模型

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=3, stride=1)
        self.conv2 = nn.Conv2d(10, 14, kernel_size=5, stride=3)
        self.fc = nn.Linear(896, 10)
    def forward(self, x):
        in_size = x.size(0)
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = x.view(in_size, -1) # flatten the tensor
        x = self.fc(x)
        return x
```

(2)传入模型和输入，得到 script_model。

```
input_shape = [1, 1, 28, 28]
shape_list = [(input_name, input_shape)]
model.load_state_dict(model_dict)
data = torch.rand(input_shape)
scripted_model = torch.jit.trace(model, data).eval()
```

(3)调用前端转换函数，得到 mod 和 params

```
mod, params = witin_frontend.frontend.from_pytorch(scripted_model,
shape_list)
```

调用该函数后得到 mod 和 params 两个参数，其中 mod 为图的结构中间表示，params 为用到的参数。

(4)编译 build

```
with witin.transform.PassContext(opt_level=3):
    _, _, _, npu_graph = witin_frontend.build_module.build(
        mod,
        target=target,
        target_host=target_host,
        params=params,
        input_data=input_dt,
        chip=chip)
```

(5)运行 run，和模拟器出来的结果进行对比

```
m = npu_graph_runtime.create(npu_graph, chip)
m.set_input(input_name, witin.nd.array(data))
m.run()
witin_output_list = [m.get_output(i).asnumpy() for i in range(1)]
```

3.6 多网络

支持多网络，详细请参照工程：

witin_mapper/tests/onnx/witin/wtm2101/operator/test_forward_multinet.py

多网络 set_input()的时候，要指定网络输入节点的名称或网络的序号,例如：

1.已知输入节点的名称

```
m.set_input('fingerprint_input', witin.nd.array(data1))
```

```
m.set_input(fingerprint_input', witin.nd.array(data2))
```

2.不知道节点名称，按照 Graph 的顺序

```
m.set_input(0, witin.nd.array(data1))
```

```
m.set_input(1, witin.nd.array(data2))
```

单网络可任意设置节点名称。

3.7 优化策略

为了优化 array 的计算精度，可以设置不同层的优化策略。

现在有支持三种优化策略：输入数据优化、权重数据优化、权重数据复制。

详细定义信息如下：

```
syntax="proto3";
```

```
enum SpMatMode{
```

```
  SIM_MODE = 0;
```

```
  MIX_MODE = 1;
```

```
  DIG_MODE = 2;
```

```
}
```

```
enum ArrayRegion {
```

```
  ANY_REGION = 0;
```

```
  UP_REGION = 1;
```

```
  DOWN_REGION = 2;
```

```
}
```

```
message LayerOptiConfig{
```

```
  string name = 1;
```

```
  message InputOpti{
```

```
    int32 shift = 1; // 0 left 1 right
```

```
int32 num = 2;  
int32 front_scale = 3;  
int32 signed_input = 4;  
}
```

```
message WeightOpti{  
int32 shift = 1;  
int32 num = 2;  
ArrayRegion region = 3;  
}
```

```
message DoubleWeightOpti{  
int32 multiple = 1;  
}
```

```
message SparseMatrixOpti{  
SpMatMode sp_mode = 1;  
int32 shift = 2;  
}
```

```
message ArrayOutputOpti{  
int32 magnify = 1;  
int32 repeat = 2;  
}
```

```
message ConvOpti{  
int32 multi_point = 1;  
bool tdnn_mode = 2;  
bool conv2DToConv1D_enable = 3;  
int32 fout_burst_point = 4;  
}
```

```
repeated InputOpti inputOpti = 2;  
repeated WeightOpti weightOpti = 3;  
repeated DoubleWeightOpti doubleWeightOpti = 4;  
repeated SparseMatrixOpti sparseMatrixOpti = 5;  
repeated ArrayOutputOpti arrayOutputOpti = 6;  
repeated ConvOpti convOpti = 7;
```

```
}  
  
message OptimizeConfig{  
  repeated LayerOptiConfig layerOptiConfig = 1;  
}
```

其中，name 为要优化的层名，为字符串；InputOpti 为输入数据优化；WeightOpti 为权重数据优化；DoubleWeightOpti 为权重数据复制；SparseMatrixOpti 为稀疏矩阵模式，ArrayOutputOpti 为输出放大优化模式，ConvOpti 为卷积优化模型。

详细参考工程：

witin_mapper/tests/python/frontend/onnx/witin/wtm21011/optimizer/test_l
ayer_optimize_method.py

在 build 中指定 optimize_method_config 参数，为优化策略配置文件。

3.7.1 输入数据优化

```
message InputOpti{  
  int32 shift = 1;  
  int32 num = 2;  
  int32 front_scale = 3;  
  int32 signed_input = 4;  
}
```

- 当需要此层输入数据数据扩大时：
 1. 设置 shift 为 1 时且 num>1，会扩大本层输入的输入，扩大的倍数为 2^{num} 。注意：当此层为首层时，需要人工将数据扩大输入，非首层时，要求前一层必须有 Relu。
 2. 设置 front_scale>0 时，会将本层的上层 scale 值缩小，缩小倍数为 $2^{\text{front_scale}}$ 。注意：此优化不可用于首层。
- 当首层输入为有符号数 int8 时：

设置 signed_input = N > 0。注意：对于有符号输入 X，需要在程序输入的时候手动将输入改变：

 1. $X1 = X[0:N]$
 2. $X2 = X[N:\text{end}] + 128$
 3. $X3 = -X[0:N]$
 4. 将 X1,X2,X3 拼接一起
 5. $X' = \text{np.concatenate}((\text{data0}, \text{data1}, \text{data2}), \text{axis} = 1)$
 6. 将负数变为 0

7. $X'[X' < 0] = 0$

8. 得到新的输入 X' (uint8)

参考示例程序为：

```
witin_mapper/tests/python/frontend/onnx/witin/wtm21011/optimizer/test_1  
ayer_optimize_method_signed_in.py
```

注意：v001.000.024 及以前版本，当首层输入数据为有符号数 int8 时，需要按照上述过程处理，v001.000.025 及以后版本支持首层输入数据为有符号数 int8 类型。

3.7.2 权重数据优化

```
message WeightOpti{  
    int32 shift = 1;  
    int32 num = 2;  
    ArrayRegion region = 3;  
}
```

当需要此层权重数据扩大时：

设置 shift 为 1 时且 $num > 1$ ，会扩大本层输入的输入，扩大的倍数为 2^{num} 。

例如，权重为 2×3 的矩阵[[1,2,3][4,5,6]]，设置

```
weightOpti {  
    shift : 1  
    num : 1  
}
```

权重变为[[2,4,6][8,10,12]]。

region 为权重分配区域指定：

ANY_REGION = 0; // 任意区域

UP_REGION = 1; // 上半区

DOWN_REGION = 2; // 下半区



3.7.3 权重数据复制

```
message DoubleWeightOpti{  
  int32 multiple = 1;  
}
```

当需要对权重复制时使用：

设置 $multiple > 1$ ，会复制本层的权重，复制的倍数为 $2^{multiple}$ 。

例如，权重为 2×3 的矩阵 $\begin{bmatrix} 1,2,3 \\ 4,5,6 \end{bmatrix}$ ，设置

```
doubleWeightOpti {  
  multiple : 1  
}
```

权重变为 4×3 的矩阵 $\begin{bmatrix} 1,2,3 & 1,2,3 \\ 4,5,6 & 4,5,6 \end{bmatrix}$ 。

3.7.4 稀疏矩阵优化

```
message SparseMatrixOpti{  
  SpMatMode sp_mode = 1;  
}  
enum SpMatMode{  
  SIM_MODE = 0;  
  MIX_MODE = 1;  
  DIG_MODE = 2;  
}
```

稀疏矩阵适用于权重大小超过 array 存放限制，即 $[-275, 275]$ 之间时配置，可以将权重分为高 8 位和低 8 位，其中低 8 位使用 array (NPU) 进行计算，高 8 位使用 CPU 计算，最后相加可得最终结果。

由于大部分高位部分是 0，所以是一个稀疏矩阵，但是当高位部分非 0 值较多时，可能会导致空间不足无法布置，mapper 会给出错误。

此优化选项分为三种模式：

SIM_MODE 为默认模式，纯 NPU 计算，权重范围为[-275,275]。

MIX_MODE 为混合模式，低位使用 NPU，高位使用 CP，高低位皆为 8bit。

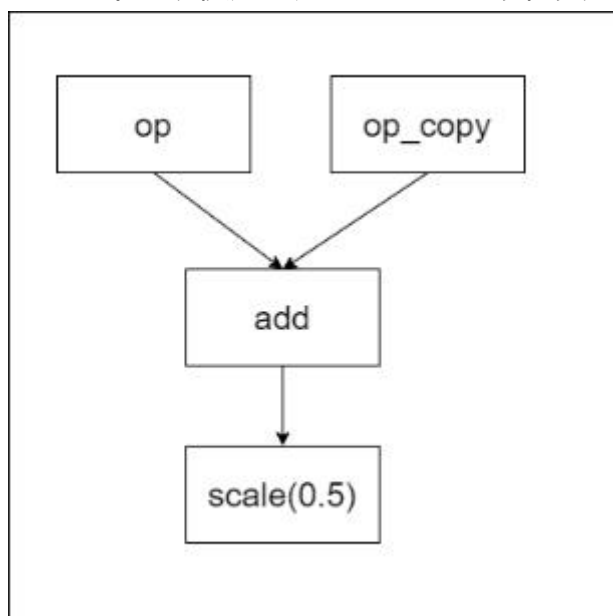
DIG_MODE 为数字模式，纯 CPU 计算，此模式只适用于 kernel shape 很小时的计算，以解决 NPU 模拟计算精度不足的问题，权重范围为[-128,127]。

3.7.5 输出放大优化

```
message ArrayOutputOpti{  
    int32 magnify = 1;  
    int32 repeat = 2;  
}
```

magnify 直接将当前层 scale 值缩小，方法为右移 magnify 位数。

repeat 复制倍数， 2^{repeat} 次：插入同样的 op_conv 算子再插入一个 add_op 和 scale_op，将 op 与 op_conv 输出结果通过 add_op 相加然后通过 scale_op 除 2，在 round 中可以使用后处理 add 结果移位实现除 2，如下图所示。



3.7.6 卷积优化

```
message ConvOpti{  
    int32 multi_point = 1;  
    bool tdnn_mode = 2;  
    bool conv2DToConv1D_enable = 3;  
}
```

此处是设定卷积的计算模式和多点并行数(加速计算)。

multi_point 为多点并行个数，设置 multi_point，需要将 tdnn_mode 或者 conv2DToConv1D_enable 为 ture 的情况下，而 tdnn_mode 和 conv2DToConv1D_enable 不能同时为 ture。

tdnn_mode:DCCRN 单独计算模式

conv2DToConv1D_enable：将 2D 卷积在 W 方向进行拆分，根据 2D 卷积计算原理，需要在 W 方向进行划窗，将 W 方向划窗的动作单独拆分为一层。

在 WTM2101 中，卷积有两种计算：普通卷积模式和 tdnn 卷积模式。

普通卷积模式是指卷积按照通常计算方法，即连续完成多次滑窗得到最终的输出 feature map，如果加速多点计算，需将 conv2DToConv1D_enable 设置为 ture，此时 tdnn_mode 必须为 false。

而 tdnn 卷积模式是将 W 方向上赋予时间序列信息，并按照 tdnn 模式计算，多用于 DCCRN 网络中，同时需将 tdnn_mode 设置为 ture，此时 conv2DToConv1D_enable 必须为 false。

3.8 特殊配置环境变量

3.8.1 Array 分配

因为硬件制作原因，有些芯片的前 N 列的 Array 计算会有较大误差，在烧写芯片测试后若发现可能为此方面的原因，可以设置环境变量：

```
export ARRAY_ALLOC_RESERVED_COLUMN=N
```

跳过后 N 列的 Array 分配。

在进行 Array 分配时，默认搜索超时时间比较短，通常在 10s~100s 之间，具体跟权重块的数量相关，在网络权重块比较多的场景下，可能搜索不到最优解或无解，可以通过环境变量设置超时时长，设置如下：

```
export ARRAY_ALLOC_TIMEOUT_LIMITS = 300
```

3.8.2 开启 FIFO

硬件支持 fifo 的使用，可以在 TDNN 的情况下提高数据输入的效率，不开启时需要一次输入 TDNN 的所需的所有帧数据，开启后可以只输入一帧数据，通过设置环境变量：

```
export WITIN_FIFO_EN = 1
```

开启。

注意：开启此选项可能导致 regfile 空间不足，若出现此问题，可尝试关闭。

3.8.3 设置校准数据数量

校准文件 expect_in/out 用于板卡烧写测试模型的准确率，文件默认有 100 帧的校准数据，若需要增大、减小数据的个数，可以设置环境变量：

```
export WITIN_EXPECTED_NUM=N
```

会保存 N 帧的校准数据。

4 Onnx 模型

4.1 创建 Tensor

1. 创建 tensor 数据, 为 int8 数据(-128~127), shape 为[514,128], 最终的数据类型为 float32。

```
params = np.random.randint(-128, 127, size = (514, 128), dtype = np.int32).astype(np.float32)
```

2. 创建 onnx tensor, 参数分别为: tensor 名称, 数据类型, 大小, 值。

```
params_tensor = onnx.helper.make_tensor("params_tensor",  
                                         data_type=onnx.TensorProto.FLOAT,  
                                         dims=(514, 128),  
                                         vals=params.flatten())
```

4.2 onnx 算子

Onnx 模型支持自定义算子。

常用算子有 Gemm (矩阵乘、全连接层、DNN 层), Tdnn (时延神经网络), Add (向量加), Relu (激活函数-relu), Scale (标量乘), LSTM (长短期记忆网络), GRU (门控循环单元), ActLut (自定义激活函数查找表), Mul(向量乘)。

常见算子列表					
序号	算子名称	作用	序号	算子名称	作用
1	Gemm	矩阵乘、全连接层、DNN 层	8	GRU	门控循环单元
2	Tdnn	时延神经网络	9	ActLut	自定义激活函数查找表
3	Add	向量加	10	Mul	向量乘

4	Relu	激活函数-relu	11	LSTM	长短期记忆网络
5	Scale	标量乘	12	Average Pool	平均池化
6	Concat	对指定维度进行拼接	13	Conv	卷积
7	Slice	切片操作	14	ConvTranspose	转置卷积

注: Tdnn、GRU、LSTM 和 ActLut 算子为自定义算子, 因此它们必须用 onnx 模型, 同时也建议其他算子也使用 onnx 构建网络模型。

4.2.1 Gemm

一般矩阵乘法 (General Matrix multiplication) 。

```
node = onnx.helper.make_node('Gemm',  
                              inputs=['input', 'params',  
                                      'bias'],  
                              outputs=['output'],  
                              name='dnn_node0')
```

其中,

inputs list[tensor]为输入、权重、偏置

outputs list[tensor]输出

name string节点名称

注意: Gemm 后通常跟一个 “scale” 节点, 代表 G 值。

4.2.2 Tdnn

时间延迟网络 (Time Delay Neural Network) 。

```
node = onnx.helper.make_node('Tdnn',  
                              inputs=['input', 'params'],  
                              outputs=['output'],  
                              time_offsets=offsets,  
                              bias_params=bias,  
                              scale_params=1024,
```

```
name='tdnn_node0')
```

其中,
inputs list[tensor]输入、权重
outputs list[tensor]输出
time_offsets tensor时延信息
biastensor偏置
scale_paramsfloat缩放值 (G 值)
name string节点名称

4.2.3 Lstm

长短期记忆网络。

```
lstm6_node = onnx.helper.make_node(  
    'Lstm',  
    inputs=['conv5_reshape_out', 'lstm6_w_ifo_tensor',  
'lstm6_w_c_tensor', 'lstm6_b_ifo_tensor', 'lstm6_b_c_tensor'],  
    scale_ioft=1024*2,  
    scale_ct=1024,  
    activate_type=['sigmoid', 'tanh', 'tanh'],  
    activate_table=act_table,  
    shift_bits=[0,0,-7, -7],  
    clean_ctht=10,  
    outputs=['lstm6_out'],  
    name="lstm6")
```

其中,
inputs list[tensor] 输入、ioft 权重、ctl 权重、ioft 偏置、ct 偏置
scale_ioft float ioft 部分 G 值
scale_ct float ct 部分 G 值
activate_typelist[string]激活表的类型, 只能为 sigmoid/tanh
activate_table tensor激活表, 每个激活表长度为 256, LSTM 有三个为 [3,256]
shift_bits list[int] 长度为 4,
 [0]为对 $ft \times ct$ (16bit)结果的移位数值,
 [1]为对 $it \times ctl$ (16bit)结果的移位数值,
 [2]为对 $ft \times ct$ (16bit)+ $it \times ctl$ (16bit)的结果 ct (16bit)移位数值,
 [3]为对 $ot(8bit) \times \tanh2_ct(8bit)$ 的结果 ht (16bit)移位数值,
 不设置默认为 [0,0,-7,-7]。
clean_ctht int 表示 $htct$ 在执行该设置次数后置为 0, 当 clean_ctht 设置为

0 或者不设置，为不清除 ht/ct.

outputs list[tensor] 输出 Tensor

name string 节点名称

计算公式如下：

$$iof_t = \text{sigmoid}(W_{iof} * (x_t, h_{t-1}))$$

$$c_{tl} = \text{tanh}_1(W_{ct} * (x_t, h_{t-1}))$$

$$c_t = f_t * c_{t-1} + i_t * c_{tl}$$

$$h_t = \text{tanh}_2(c_t) * o_t$$

4.2.4 Gru

门控循环单元。

```
node = onnx.helper.make_node(
    'Gru',
    inputs=['in', 'gru_zrt_params', 'gru_ht_params',
'gru_zrt_bias', 'gru_ht_bias'],
    scale_zr=1024,
    scale_ht=1024,
    activate_type=['sigmoid', 'tanh'],
    activate_table=act_table,
    shift_bits=[-7, -7],
    outputs=['out'],
    name="gru_node1")
```

其中，

inputs list[tensor]输入、zrt 权重、ht 权重、zrt 偏置、ht 偏置

scale_zr float zrt 部分 G 值

scale_ht float ht 部分 G 值

activate_typelist[string]激活表的类型，只能为 sigmoid/tanh

activate_table tensor激活表，每个激活表长度为 256, Gru 有两个为[2,256]

shift_bitslist[int]长度为 2，

[0]为对 $rt(8\text{bit}) * ht(8\text{bit})$ 的结果 $ct(16\text{bit})$ 移位，

[1]为对 $[(1-zt)*ht](16\text{bit}) + [zt*ht\sim](16\text{bit})$ 的结果 $ht(16\text{bit})$ 移位，

不设置默认为[-7,-7]。

outputs list[tensor]输出 Tensor

name string节点名称

计算公式如下：

$$\begin{aligned}z_t &= \sigma(W_z \cdot [h_{t-1}, x_t]) \\r_t &= \sigma(W_r \cdot [h_{t-1}, x_t]) \\ \tilde{h}_t &= \tanh(W \cdot [r_t * h_{t-1}, x_t]) \\ h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t\end{aligned}$$

4.2.5 Scale

标量乘。

```
node = onnx.helper.make_node('Scale',  
                              inputs=['in'],  
                              outputs=['out'],  
                              scale=0.0009765625)
```

其中,

inputs list[tensor]输入

outputs list[tensor]输出

scale float缩放值

注意: scale 的值取值范围为 $[-128, 127] \cup (-1, 1)$ 。当 scale 的值为 $(-1, 1)$ 时, scale 的值只支持 2^{-n} , n 为整数, $n \leq 13$: $n < 8$ 时, 为后处理乘法, 使用移位进行计算。 $n \geq 8$, $n \leq 13$ 时, 若 scale 层跟在 Gemm 层、Conv2d 层后, 代表为上层 Gemm/Conv2d 的 G 值, 即最小 G 值为 256, 最大为 8192; 若不跟在 Gemm 层、Conv2d 层后, 则为单独的乘法计算。当 scale 的值取值范围为 $[-128, 127]$ 时, scale 只能为其中的整数。

4.2.6 Relu

激活函数。

```
node = onnx.helper.make_node('Relu',  
                              inputs = ['input'],  
                              outputs = ['output'])
```

其中,

inputs list[tensor]输入

outputs list[tensor]输出

4.2.7 ActLut

自定义激活查找表函数, 通过 int8 的数据范围-128~127 的映射表, 映射为激活输出。

```
node = onnx.helper.make_node('ActLut',  
                              inputs=['in'],  
                              outputs=['out'],  
                              act_type='sigmoid',  
                              table_params=act_table)
```

其中,

inputs list[tensor]输入

outputs list[tensor]输出

act_typestring激活表类型, 可选为 sigmoid、tanh

table_paramstensor激活表 tensor, 为[1,256]大小

注意: 查找表的对应方式为: 源输入[0~127] 对应查找表下标 0~127, 源输入[-128~-1]对应查找表下标 128~255, 即: 输入[-128~127]的二进制转变为无符号形式为[128~255,0~127], 对应为查找表下标。

4.2.8 Mul

乘

```
node = onnx.helper.make_node('Mul',  
                              inputs=['in', 'mul_x'],  
                              outputs=['out'],  
                              )
```

其中,

inputs list[tensor]输入, 左乘数, 右乘数

outputs list[tensor]输出

注意: 如果左乘数和右乘数都为向量时, 它们的 shape 要求相等; 也支持左乘数和右乘数其中一个是向量, 另外一个为标量的情况。

4.2.9 Add

加

```
node = onnx.helper.make_node('Add',  
                              inputs = ['in', 'add_x'],  
                              shift_bit = -1,  
                              outputs = ['out'])
```

其中,

inputs list[tensor] 输入, 左加数, 右加数

shift_bit int 表示对 add 后的结果移位, 负数表示右移, 正数表示左移, 默认 0

outputs list[tensor] 输出

注意: 如果左加数和右加数都为向量时, 它们的 shape 要求相等; 也支持

左加数和右加数其中一个是向量，另外一个为标量的情况。

4.2.10 Averagepool

平均池化。

```
left_ave_pooling_node = onnx.helper.make_node(
    "AveragePool",
    inputs=["conv16_left_add_relu"],
    outputs=["out"],
    kernel_shape=[35, 2],
    pads=[0, 0, 0, 0], #[top,left,bottom,right]
    strides=[1, 1],
    scale_in=0.0625,
    scale_out=0.0625)
```

其中，

inputs list[*tensor*] 输入、权重、偏置

outputs list[*tensor*] 输出

kernel_shape list 池化核 shape

pads list 填充

strides list 步长

scale_in float 输入缩放尺度，默认为 1

scale_out float 输出缩放尺度，默认为 1

4.2.11 Slice

切片操作。

```
conv3_1_left_slice_node = onnx.helper.make_node(
    "Slice",
    inputs=["gemm3_left_reshape_out",
"conv3_1_left_slice_starts", "conv3_1_left_slice_ends",
"conv3_1_left_slice_axes", "conv3_1_left_slice_steps"],
    outputs=["conv3_1_left_slice_out"])
```

其中，

inputs list[*tensor*] 输入、切片起始索引、切片结束索引、切片指定轴、切片步长

outputs list[*tensor*] 输出

4.2.12 Concat

数据拼接。

```
concat_node = onnx.helper.make_node(
    "Concat",
    inputs=["concat_in1","concat_in2"],
    outputs=["deconv7_concat"],
    axis = 1)
```

其中,

inputs list[tensor] 输入: 左拼接数, 右拼接数

outputs list[tensor] 输出

axis int 指定拼接轴, 目前 WTM2101 仅支持 axis=1.

4.2.13 Conv

卷积。

```
conv3_node = onnx.helper.make_node("Conv",
    inputs=['conv2_relu', 'conv3_w_tensor',
'conv3_b_tensor'],
    outputs=['conv3'],
    kernel_shape=[3, 2],
    strides=[2, 1],
    pads=[1, 1, 0, 0],
    name="conv3")
```

其中,

inputs list[tensor] 输入、权重、偏置

outputs list[tensor] 输出

kernel_shape list[int] 卷积核 shape

strides list[int] 滑动步长

pads list[int] 填充

name string 节点名称

注意:

1) 在构建 onnx 模型权重维度是[O,I,H,W],经工具链内部转换, 维度转换为[O,W,H,I],最终 shape 为[O,W*H*I];

2) 在工具链内部, 卷积 feature map 维度为[N,W,H,C].

4.2.14 ConvTranspose

转置卷积。

```
deconv12_node = onnx.helper.make_node("ConvTranspose",
```

```
inputs=['deconv_input', 'deconv_weight', 'deconv_bias'],
outputs=['deconv_output'],
kernel_shape=[3, 2],
strides=[2, 1],
pads=[1, 0, 1, 0],
output_padding=[1, 0, 1, 0],
name="deconv_node")
```

其中,

inputs list[tensor] 输入、权重、偏置

outputs list[tensor] 输出

kernel_shape list[int] 卷积核 shape

strides list[int] 滑动步长

pads list[int] 填充

output_padding list[int] 输出填充

name string 节点名称

注意:

注意:

1) 在构建 onnx 模型权重维度是[I,O,H,W],经工具链内部将权重转换为等价的 Conv, 转换后维度转换为[O',W',H',I'],最终 shape 为[O' ,W' *H' *I'];

2) 在工具链内部, 转置卷积的 feature map 维度为[N,W,H,C];

3) 在 WTM2101 工具链中, 转置卷积主要应用在 DCCRN 场景中.

4.2.15 Gru2Array

新增双权重 GRU 部署方式, 其接口如下:

```
gru_node1 = onnx.helper.make_node('Gru2Array',
    inputs=['in', 'gru_params1', 'gru_params2', 'gru_bias1',
'gru_bias2'],
    input_offset=0,
    hidden_shift_bit=-1,    #输入 scale
    scale_input=1024,
    scale_hidden=1024,
    scale_ones = 128,
    activate_type=['sigmoid', 'tanh'],
    activate_table=act_table,
    shift_bits=[-7, -7, -2, -3],
    clean_ht=0,
    outputs=['out'],
    name="gru_node1")
```

`inputs list[tensor]` 输入、`input_zrn` 权重、`hidden_zrn` 权重、`input_zrn` 偏置、`hidden_zrn` 偏置

`input_offset`: 对输入的偏移量, 暂未适配, 现仅保留接口

`hidden_shift_bit`: 对 array 计算后的 `hidden_zrn` 进行移位, “-”代表右移位

`scale_input float` 表示 `input_zrn` 部分 G 值:

`scale_hidden float` 表示 `hidden_zrn` 部分 G 值

`activate_typelist[string]` 激活表的类型, 只能为 `sigmoid/tanh`

`activate_table tensor` 激活表, 每个激活表长度为 256, 或者 1024, 目前建议取长度为 1024

`shift_bitslist[int]` 长度为 2,

[0] 为对 $rt(8bit) * hn(8bit)$ 的结果 (16bit) 移位,

[1] 为对 $(1-zt) * ht(16bit) + [zt * ht](16bit)$ 的结果 $ht(16bit)$ 移位,

[2] 为对 $(1-zt) * ht$ 移位, 如果不移位, 必须设置为 0

[3] 为对 $zt * ht$ 进行移位, 如果不移位, 必须设置为 0

不设置默认为 $[-7, -7, -7, -7]$, -7 表示右移七位, -代表右移位。

`outputs: list[tensor]`, 输出 Tensor

`name string`: 节点名称

`scale_ones`: 表示“1”和 `zt` 的缩放尺度, 目前支持 -128~128

`clear_ht:gru` 执行如果次数后, 将 `ht` 清零, 等于 0 时, 代表不清除 `ht`

4.3 创建 graph

节点列表

```
nodes = [node0, node1]
```

图的名称

```
name = 'model_name'
```

输入信息

```
inputs=[onnx.helper.make_tensor_value_info("in",onnx.TensorProto.FLOAT,list(in_shape))]
```

输出信息

```
outputs=[onnx.helper.make_tensor_value_info("out",onnx.TensorProto.FLOAT,list(out_shape))]
```

初始化 Tensor

```
initializer=[tensor1, tensor2, tensor3]
```

```
graph = onnx.helper.make_graph(nodes,
                                name,
                                inputs,
                                outputs,
                                initializer
                                )
```

4.4 保存 graph

```
model=helper.make_model(graph, producer_name=model_name)
with open(ROOT_DIR + model_name + '.onnx', "wb") as of:
    of.write(model.SerializeToString())
```

4.5 导入模型

```
onnx_model = onnx.load(file_path)
```

5 常见问题和说明

5.1 Array 分配 error

```
[10:53:27] [ERROR]low_level_mapping/src/tensor/array_alloc_cp.cpp:169: Failed to allocation array memory
[10:56:07] [ERROR]low_level_mapping/src/tensor/array_alloc_cp.cpp:174: Allocation Result After extend column:
Layer Name ~ Layer Type ~ Weight Size ~ Bias Size ~ Weight Region ~ Bias Region
gemm_node0 ~ OpType::MV_OP ~ [235x123] ~ [ 1x123] ~ [896-1131][256-379] ~ [10-11][256-379]
gemm_node1 ~ OpType::MV_OP ~ [235x123] ~ [ 1x123] ~ [235-470][640-763] ~ [15-16][640-763]
gemm_node2 ~ OpType::MV_OP ~ [235x123] ~ [ 1x123] ~ [ 0-235][768-891] ~ [ 9-10][768-891]
gemm_node3 ~ OpType::MV_OP ~ [235x123] ~ [ 1x123] ~ [470-705][640-763] ~ [14-15][640-763]
gemm_node4 ~ OpType::MV_OP ~ [235x123] ~ [ 4x123] ~ [896-1131][379-502] ~ [ 7-11][379-502]
gemm_node5 ~ OpType::MV_OP ~ [235x123] ~ [ 1x123] ~ [470-705][1024-1147] ~ [15-16][1024-1147]
gemm_node6 ~ OpType::MV_OP ~ [235x123] ~ [ 4x123] ~ [ 0-235][640-763] ~ [ 7-11][640-763]
gemm_node7 ~ OpType::MV_OP ~ [235x123] ~ [ 4x123] ~ [ 0-235][512-635] ~ [ 4- 8][512-635]
gemm_node8 ~ OpType::MV_OP ~ [235x123] ~ [ 4x123] ~ [235-470][896-1019] ~ [ 1- 5][896-1019]
gemm_node9 ~ OpType::MV_OP ~ [235x123] ~ [ 4x123] ~ [896-1131][768-891] ~ [ 5- 9][768-891]
gemm_node10 ~ OpType::MV_OP ~ [235x123] ~ [ 4x123] ~ [1131-1366][256-379] ~ [11-15][256-379]
gemm_node11 ~ OpType::MV_OP ~ [235x123] ~ [ 1x123] ~ [1131-1366][128-251] ~ [ 9-10][128-251]
gemm_node12 ~ OpType::MV_OP ~ [235x123] ~ [ 1x123] ~ [470-705][1152-1275] ~ [ 8- 9][1152-1275]
gemm_node13 ~ OpType::MV_OP ~ [235x123] ~ [ 4x123] ~ [ 0-235][384-507] ~ [ 1- 5][384-507]
gemm_node14 ~ OpType::MV_OP ~ [235x123] ~ [ 4x123] ~ [ 0-235][256-379] ~ [ 6-10][256-379]
gemm_node15 ~ OpType::MV_OP ~ [235x123] ~ [ 4x123] ~ [ 0-235][896-1019] ~ [ 5- 9][896-1019]
gemm_node16 ~ OpType::MV_OP ~ [235x123] ~ [ 4x123] ~ [1366-1601][512-635] ~ [11-15][512-635]
gemm_node17 ~ OpType::MV_OP ~ [235x123] ~ [ 4x123] ~ [1366-1601][896-1019] ~ [ 9-13][896-1019]
gemm_node18 ~ OpType::MV_OP ~ [235x123] ~ [ 4x123] ~ [235-470][768-891] ~ [ 1- 5][768-891]
gemm_node19 ~ OpType::MV_OP ~ [235x123] ~ [ 4x123] ~ [1131-1366][1024-1147] ~ [11-15][1024-1147]
gemm_node20 ~ OpType::MV_OP ~ [235x123] ~ [ 1x123] ~ [896-1131][640-763] ~ [ 6- 7][640-763]
gemm_node21 ~ OpType::MV_OP ~ [235x123] ~ [ 1x123] ~ [896-1131][1152-1275] ~ [15-16][1152-1275]
gemm_node22 ~ OpType::MV_OP ~ [235x123] ~ [ 1x123] ~ [470-705][512-635] ~ [10-11][512-635]
gemm_node23 ~ OpType::MV_OP ~ [235x123] ~ [ 4x123] ~ [1366-1601][128-251] ~ [ 5- 9][128-251]
gemm_node24 ~ OpType::MV_OP ~ [235x123] ~ [ 4x123] ~ [470-705][128-251] ~ [11-15][128-251]
gemm_node25 ~ OpType::MV_OP ~ [235x123] ~ [ 4x123] ~ [ 0-235][128-251] ~ [ 1- 5][128-251]
```

提示:

[ERROR]low_level_mapping/src/tensor/array_alloc_cp.cpp:169: Failed to allocation array memory

说明 array 分配超出限制, 需重新调整网络结构或者优化方式。

5.2 Regfile 空间不足

```
[11:10:54] [ERROR]low_level_mapping/src/memory/mem.cpp:284: No enough memory to alloc, Need: 771
Max memory len: 8192
Mode: FIFO_DISABLED
Free Blocks Size: 1
Free Blocks List:
begin:7616 length:576 state:FREE
Used Blocks Size: 15
Used Blocks List:
begin:0 length:160 state:USED
begin:160 length:800 state:USED
begin:960 length:160 state:USED
begin:1120 length:800 state:USED
begin:1920 length:800 state:USED
begin:2720 length:160 state:USED
begin:2880 length:800 state:USED
begin:3680 length:160 state:USED
begin:3840 length:800 state:USED
begin:4640 length:800 state:USED
begin:5440 length:160 state:USED
begin:5600 length:800 state:USED
begin:6400 length:160 state:USED
begin:6560 length:800 state:USED
begin:7360 length:256 state:USED
```

提示:

[ERROR]low_level_mapping/src/memory/mem.cpp:284: No enough memory to alloc, Need: 771

说明 regfile 资源不够使用，需重新调整网络结构。

5.3 SRAM 空间不足

```
[bt] (6) /home/shuaikailiu/witin_mapper_pro/witin_mapper/build/libwitin.so(witin::base::Session::SaveBoardConfigBin(witin::base::ChipId const&)+0xb28) [0x7fd8f316896c]
[bt] (5) /home/shuaikailiu/witin_mapper_pro/witin_mapper/build/libwitin.so(witin::base::RPTTool::GenComputeFlow()+0x2b0) [0x7fd8f322a1c2]
[bt] (4) /home/shuaikailiu/witin_mapper_pro/witin_mapper/build/libwitin.so(witin::base::BB04P1CodeGen::GenComputeFlow()+0x2268) [0x7fd8f320e682]
[bt] (3) /home/shuaikailiu/witin_mapper_pro/witin_mapper/build/libwitin.so(witin::base::BB04P1CodeGen::GenDmaParams(Json::Value const&)+0x220) [0x7fd8f320b870]
[bt] (2) /home/shuaikailiu/witin_mapper_pro/witin_mapper/build/libwitin.so(int witin::base::SetBitFieldInfo<unsigned int>+0x1c2) [0x7fd8f32121d7]
[bt] (1) /home/shuaikailiu/witin_mapper_pro/witin_mapper/build/libwitin.so(unsigned int witin::base::SetBits<unsigned int>+0x22f) [0x7fd8f3213617]
[bt] (0) /home/shuaikailiu/witin_mapper_pro/witin_mapper/build/libtvm.so(dmlc::LogMessageFatal::~LogMessageFatal()+0x79) [0x7fd8f3b3d619]
[13:48:01]witin_mapper_pro/low_level_mapping/include/witin/target/rpt/bitfield_info.h:67: Bitfield ERROR: json:[paras7, pending_rd_addr, ] name:pending_rd_addr lsb:16 len:16 value:67200 (value << lsb) & (~mask):0 line:0x0
```

提示：

Bitfield ERROR: json:[paras7, pending_rd_addr,] name:pending_rd_addr lsb:16 len:16 value:67200

说明：SRAM 资源不足，需重新调整网络结构。

5.4 输入类型错误

```
scale
  optimize_method_config=opt_config)
File "/home/shuaikailiu/witin_mapper_pro/witin_mapper/python/tvm/relay/build_module.py", line 535, in build
  array_distribute=array_distribute)
File "/home/shuaikailiu/witin_mapper_pro/witin_mapper/python/tvm/relay/build_module.py", line 327, in build
  _judge_input_is_uint(idata)
File "/home/shuaikailiu/witin_mapper_pro/witin_mapper/python/tvm/relay/build_module.py", line 88, in _judge_input_is_uint
  param.dtype))
ValueError: input data only support float32, but now is float64, please use input_data.astype('float32')
(py36) shuaikailiu@tc-virtual-machine:~/witin_mapper_pro/witin_mapper$
```

提示：

ValueError: input data only support float32, but now is float64, please use input_data.astype('float32')

说明：输入数据类型有无，应使用 input_data.astype('float32')。