ARM 기반 임베디드 시스템에서 mixed precision 을 위한 c/tensorflow 프레임워크 구성

이종은[1], 임승호[2]
[1] 한국외국어대학교 컴퓨터전자시스템공학부
[2] 한국외국어대학교 컴퓨터공학부
whddms1208@gmail.com, slim@hufs.ac.kr

A Configuration of the c/tensorflow framework for mixed precision on ARM-based embedded systems

Jong-Eun Lee[1], Seung-Ho Lim[2]
[1]Division of Computer Electronic System Engineering, Hankuk University of Foreign Studies
[2]Division of Computer Engineering, Hankuk University of Foreign Studies

요 약

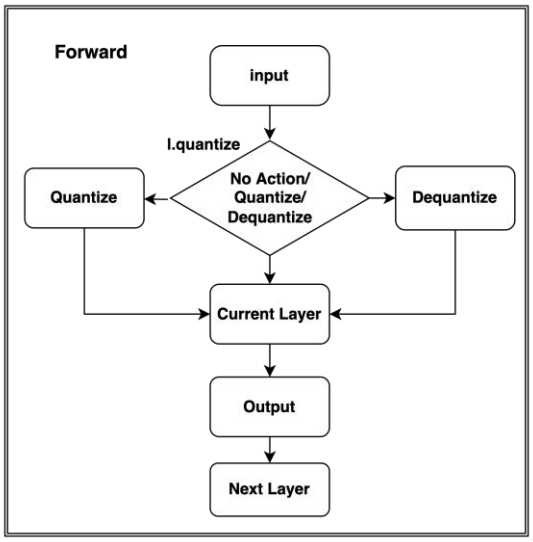
ARM 아키텍처를 사용하는 임베디드 시스템에서 int8, fp16, fp32 데이터를 조합하여 c/c++로 작성된 mixed precision CNN 을 실행시키기 위한 프레임워크 구성으로, 네트워크의 레이어마다 다른 정밀도를 사용하여 네트워크 경량화 및 추론 정확도 향상을 위한 최적의 설정을 탐색하는 실험 및 분석이 가능토록 하는 것을 목적으로 한다. 주요 구성은 network forwarding 중 레이어의 입력이 레이어에 설정된 정밀도와 다를 경우 실행되는 양자화/반양자화를 c/c++로 바인딩된 tensorflow 의 quantization 모듈을 사용하여 진행하고 ARM 시스템에서 c/c++의 fp16 을 사용하기 위해 fp16 를 컴파일이 가능한 ARM compiler 를 사용하는 프레임워크를 제안한다.

1. 서론

딥러닝 신경망에서 높은 정밀도를 가진 파라미터의 수가 커질수록 높은 정확도와 성능을 보이는 경향을 보인다.[1] 하지만 거대해지는 모델을 감당하기 위한 하드웨어 성능 향상에는 많은 비용이 소요되며 이마저도 성능을 무한정 끌어올리는 것에 제한이 있다. 더욱이 자율주행 시스템, 감시장비, 스마트 관제 시설, 모바일 장치 등 성능 제약이 크지만 실시간 추론이 요구되는 장비[2]에 거대한 신경망 모델을 높은 정밀도로 실행하는 것은 효율적이지 못한 선택이다. 이러한 점은 모델 경향화를 통해 해결할 수 있으며 주로 가지치기, 양자화, 지식 증류, 경량 네트워크 설계를 사용한다.[3]

본 논문에서는 양자화에 초점을 맞추고 있다. 양자화는 학습 후 양자화와 학습 중 양자화를 하는 방법이 있으며 float32 데이터를 주로 16bit, 8bit 로 표현하면서 파라미터 삭제 없이 모델의 크기를 줄일 수 있으며 모델의 성능 저하가 적다는 장점이 있다.[4], [5] 하지만 정밀도를 낮추게 되면 필연적으로 학습 중 오차가 발생할 수 있으며 학습이 진행되지 않는 문제가 발생할 수 있다.[6] 이러한 문제를 mixed precsion 으로 해결할 수 있어[7] 본 논문에서는 모델의 성능 저하가 적은 학습 후 양자화된 모델을 mixed precision 네트워크에 적용하여 네트워크의 경량화의 정도와 정확도 향상을 위한 실험 및 분석 환경을 구축을 위한 프레임워크를 구성하였다.

2. 본론



(그림 1) Dynamic quantization/dequantization concept for network forwarding

본 논문에서는 jetson nano 에서 yolov3-tiny 모델을 사용한 mixed precision CNN network 의 구성을 제안하며 (그림 1)의 개념도는 이미지 추론을 위한 network forwarding 에서 동적으로 양자화/반양자화를 방법을 나타내고 있다. 현재 레이어로 입력값이 들어오면 (그림 2)와 같이 yolov3-tiny.cfg 파일에 설정한 정밀도와 양자와/반양자화 여부에 따라 적절하게 입력값의 정밀도를 변환한 후 forwarding 을 수행한다.

```
[convolutional]
batch_normalize=1
filters=16
size=3
stride=1
pad=1
activation=leaky
fp32 = 1
fp16 = 0
int8 = 0
quantize = 0
```

(그림 2) conv0 layer configuration in yolov3-tiny.cfg

위 설정에서 각 레이어마다 fp32, fp16, int8, quantize 옵션에 값을 변경하면 그에 맞게 yolov3-tiny 의 네트워크가 동작하여 mixed precision CNN network 에서 경량화 및 정확도 향상에 대한 실험 및 분석을 위한 프레임 워크를 구현할 수 있다.

(그림 1)에서 양자화/반양자화의 실행은 tf.quantization 모듈을 사용한다. tensorflow 는 python 으로 동작하므로 동작결과를 c 언어로 작성되어 있는 yolov3 의 소스로 전달해야 하지만 jetson nano 에 기본적으로 설치되는 python 버전이 3.7.1 이며 이에 호환이 되는 tensorflow 의 버전은 2.3.1 인 점은 python 2.x 에서 동작하는 cpython 라이브러리로는 tf.quantization 결과를 c/c++로 작성된 소스에 반환할 수 없도록 한다. 이를 해결하기 위해 tensorflow 는 c/c++로 빌드하는 방법을 제공한다. tensorflow c 바인딩은 시스템이 사용하고 있는 아키텍처와 OS 에 따라 다르게 제공되고 있으며 bazel 을 사용하는 방법과 cmake 를 사용하는 방법으로 나뉜다. 본 논문에서 사용한 방법은 ARM 아키텍처 ubuntu 18.04 환경에서 bazel 을 사용한 c/tensorflow 를 빌드하였다. (그림 3)은 bazel 을 사용한 tensorflow 를 빌드하기 전 필요한 옵션에 대한 configure 화면이다.

```
boo@BOO:~/tfbuild/tensorflow-2.3.1$ ./configure
You have bazel 3.1.0- (@non-git) installed.
Please specify the location of python. [Default is /usr/bin/python3]:


Found possible Python library paths:
  /usr/lib/python3/dist-packages
  /usr/lib/python3.6/dist-packages
  /usr/local/lib/python3.6/dist-packages
Please input the desired Python library path to use.  Default is [/usr/lib/python3/dist-packages]

Do you wish to build TensorFlow with OpenCL SYCL support? [y/N]: N
No OpenCL SYCL support will be enabled for TensorFlow.

Do you wish to build TensorFlow with ROCm support? [y/N]:
No ROCm support will be enabled for TensorFlow.

Do you wish to build TensorFlow with CUDA support? [y/N]: N
No CUDA support will be enabled for TensorFlow.

Do you wish to download a fresh release of clang? (Experimental) [y/N]: N
Clang will not be downloaded.

Please specify optimization flags to use during compilation when bazel option "--config=opt" is spect
fied [Default is -march=native -Wno-sign-compare]:


Would you like to interactively configure ./WORKSPACE for Android builds? [y/N]:
Not configuring the WORKSPACE for Android builds.

Preconfigured Bazel build configs. You can use any of the below by adding "--config=<>" to your build
 command. See .bazelrc for more details.
        --config=mkl            # Build with MKL support.
        --config=monolithic     # Config for mostly static monolithic build.
        --config=ngraph         # Build with Intel nGraph support.
        --config=numa           # Build with NUMA support.
        --config=dynamic_kernels        # (Experimental) Build kernels into separate shared objects.
        --config=v2             # Build TensorFlow 2.x instead of 1.x.
Preconfigured Bazel build configs to DISABLE default on features:
        --config=noaws          # Disable AWS S3 filesystem support.
        --config=nogcp          # Disable GCP support.
        --config=nohdfs         # Disable HDFS support.
        --config=nonccl         # Disable NVIDIA NCCL support.
Configuration finished
```

(그림 3) configuring tensorflow before building

```
boo@BOO:~/tfbuild/tensorflow-2.3.1$ bazel build --config=opt --config=cuda --config=noaws
    --jobs 4 --local_ram_resources=HOST_RAM*0.5 --verbose_failures //tensorflow:libtensorflow_cc.so
INFO: Options provided by the client:
  Inherited 'common' options: --isatty=1 --terminal_columns=101
INFO: Reading rc options for 'build' from /home/boo/tfbuild/tensorflow-2.3.1/.bazelrc:
  Inherited 'common' options: --experimental_repo_remote_exec
INFO: Reading rc options for 'build' from /home/boo/tfbuild/tensorflow-2.3.1/.bazelrc:
  'build' options: --apple_platform_type=macos --define framework_shared_object=true --define open_so
urce_build=true --java_toolchain=//third_party/toolchains/java:tf_java_toolchain --host_java_toolchai
n=//third_party/toolchains/java:tf_java_toolchain --define=use_fast_cpp_protos=true --define=allow_ov
ersize_protos=true --spawn_strategy=standalone -c opt --announce_rc --define=grpc_no_ares=true --noin
compatible_remove_legacy_whole_archive --noincompatible_prohibit_aapt1 --enable_platform_specific_con
fig --config=v2
INFO: Reading rc options for 'build' from /home/boo/tfbuild/tensorflow-2.3.1/.tf_configure.bazelrc:
  'build' options: --action_env PYTHON_BIN_PATH=/usr/bin/python3 --action_env PYTHON_LIB_PATH=/usr/li
b/python3/dist-packages --python_path=/usr/bin/python3 --config=xla --action_env TF_CONFIGURE_IOS=0
INFO: Found applicable config definition build:v2 in file /home/boo/tfbuild/tensorflow-2.3.1/.bazelrc
: --define=tf_api_version=2 --action_env=TF2_BEHAVIOR=1
INFO: Found applicable config definition build:xla in file /home/boo/tfbuild/tensorflow-2.3.1/.bazelr
c: --action_env=TF_ENABLE_XLA=1 --define=with_xla_support=true
INFO: Found applicable config definition build:opt in file /home/boo/tfbuild/tensorflow-2.3.1/.tf_con
figure.bazelrc: --copt=-march=native --copt=-Wno-sign-compare --host_copt=-march=native --define with
_default_optimizations=true
INFO: Found applicable config definition build:cuda in file /home/boo/tfbuild/tensorflow-2.3.1/.bazel
rc: --config=using_cuda --define=using_cuda_nvcc=true
INFO: Found applicable config definition build:using_cuda in file /home/boo/tfbuild/tensorflow-2.3.1/
.bazelrc: --define=using_cuda=true --action_env TF_NEED_CUDA=1 --crosstool_top=@local_config_cuda//cr
osstool:toolchain
INFO: Found applicable config definition build:noaws in file /home/boo/tfbuild/tensorflow-2.3.1/.baze
lrc: --define=no_aws_support=true
INFO: Found applicable config definition build:linux in file /home/boo/tfbuild/tensorflow-2.3.1/.baze
lrc: --copt=-w --define=PREFIX=/usr --define=LIBDIR=$(PREFIX)/lib --define=INCLUDEDIR=$(PREFIX)/inclu
de --cxxopt=-std=c++14 --host_cxxopt=-std=c++14 --config=dynamic_kernels
INFO: Found applicable config definition build:dynamic_kernels in file /home/boo/tfbuild/tensorflow-2
.3.1/.bazelrc: --define=dynamic_loaded_kernels=true --copt=-DAUTOLOAD_DYNAMIC_KERNELS
WARNING: /home/boo/tfbuild/tensorflow-2.3.1/tensorflow/core/BUILD:1749:1: in linkstatic attribute of
cc_library rule //tensorflow/core:lib_internal: setting 'linkstatic=1' is recommended if there are no
 object files. Since this rule was created by the macro 'cc_library', the error might have been cause
d by the macro implementation
WARNING: /home/boo/tfbuild/tensorflow-2.3.1/tensorflow/core/BUILD:2161:1: in linkstatic attribute of
cc_library rule //tensorflow/core:framework_internal: setting 'linkstatic=1' is recommended if there
are no object files. Since this rule was created by the macro 'tf_cuda_library', the error might have
 been caused by the macro implementation
INFO: Analyzed target //tensorflow:libtensorflow_cc.so (195 packages loaded, 18922 targets configured
).
INFO: Found 1 target...
INFO: Deleting stale sandbox base /home/boo/.cache/bazel/_bazel_boo/5b2ab958a80850ef91525ab20be29e13/
sandbox
[37 / 1,949] 4 actions, 2 running
    @com_google_protobuf//:protobuf; 3s local
    @com_google_protobuf//:protobuf; 2s local
    [Sched] linking external/zlib/libzlib.a [for host]; 36s
    [Sched] linking external/com_google_protobuf/libprotobuf_lite.lo [for host]
```

(그림 4) building tensorflow with bazel

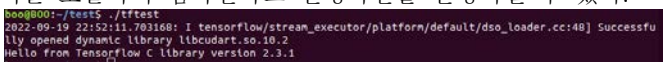
Jetson nano 의 경우 CUDA core 를 사용하는 GPU 를 가지고 있지만 일반적인 임베디드에서 CPU 만 사용하는 환경과 동일하게 구성하기 위해 OpenCL, CUDA, TensorRT 와 같은 기능을 모두 제외하고 빌드를 하도록 구성하였다. Bazel 로 빌드를 시작하면 (그림 4)와 같은 화면에서 사전에 다운로드 받은 tensorflow 소스를 c 로 빌드를 진행하며 Jetson nano 기준으로 약 34 시간 소요된다.

libtensorflow_cc.so	447.5 MB	30 7월
libtensorflow_cc.so.2	447.5 MB	30 7월
libtensorflow_cc.so.2.3.1	447.5 MB	30 7월
libtensorflow_cc.so.2.3.1-2.params	185.1 kB	29 7월
libtensorflow_framework.so.2	29.5 MB	30 7월
libtensorflow_framework.so.2.3.1	29.5 MB	30 7월
libtensorflow_framework.so.2.3.1-2.params	48.4 kB	28 7월

(그림 4) c/tensorflow building results

빌드가 완료되면 (그림 4)에서 볼 수 있는 so 파일들을 확인할 수 있고 결과물이 있는 디렉토리 경로로 링커 환경

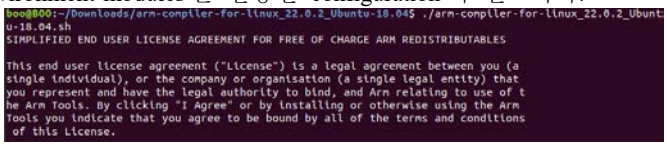
변수를 구성하면 (그림 5)처럼 c/c++에서 tensorflow 라이브러리를 호출하여 컴파일하고 실행파일을 실행시킬 수 있다.



(그림 5) calling the c/tensorflow library test result

위와 같이 c/tensorflow 를 구성하였다면 tf.quantization 모듈을 사용할 수 있는 환경이 구축이 되었다. 하지만 fp16 데이터를 사용하기 위해서는 추가적인 작업이 필요하다. 일반적으로 c/c++의 data type 에는 실수형으로 float 을 사용하며 32bit 크기를 가지지만 32bit 의 절반에 해당하는 float 16bit 에 해당하는 data type 은 없고 이를 gcc 컴파일러로 컴파일 할 수 없다. 그러나 시스템이 ARM 아키텍처로 구성되어 있을 때 ARM 에서 제공하는 arm compiler 를 사용하면 fp16 을 컴파일할 수 있다.

2022.02 부터 ARM 에서 Linux OS 에 한해서 ARM Compiler 의 라이센스를 요구하지 않아 Linux 를 사용하는 누구나 ARM Developer 사이트에서 다운로드 받아 사용할 수 있으며 지원하는 Linux 는 RHEL, SUSE, Ubuntu 가 있고 사용하는 버전에 맞게 설치할 수 있다. 설치 과정은 (그림 6)과 같이 shell script 를 실행시키면 설치를 진행할 수 있으며 설치 후에는 “armclang” 커멘드를 사용할 수 있도록 environment-modules 를 활용한 configuration 이 필요하다.



(그림 6) installing the arm compiler

위 과정을 거치면 ARM 아키텍처 시스템에서 fp16 을 c/c++에서 컴파일을 할 수 있는 환경이 구성이 된다. (그림 7)과 같이 테스트를 진행할 때 <arm_fp16.h>에서 __Float16 혹은 __fp16 data type 을 사용하여 fp16 형식을 지정할 수 ㅁ그있고 gcc 대신 armclang 을 사용해서 컴파일을 할 때 몇가지 옵션을 지정하면 컴파일이 가능하며 만일 printf 를 출력을 하려면 double 로 cast 를 한 후에 출력해야 하며 (그림 8)에서 확인 가능하다.

```c
#include <stdio.h>
#include <arm_fp16.h>

int main() {
        __Float16 a, c;
        __fp16 b, d;

        a = 1.12;
        b = 0.68;

        c = a + a;
        d = b + b;

        printf("c: %f, d: %f\n", (double)c, (double)d);

        d = a + b;

        printf("d: %f\n", (double)d);
        return 0;
}
```

(그림 7) test code for fp16 with armclang

```
boo@BOO:~/test$ ./arm_test
c: 2.240234, d: 1.360352
d: 1.800781
```

(그림 8) test result for fp16 with armclang

3. 결론

본 논문에서는 임베디드 시스템에서 c/c++로 작성된 신경망에 mixed precision 을 적용을 위해 c/tensorflow 와 arm compiler 를 사용한 프레임워크를 제안한다. 임베디드 시스템에서 빌드한 tensorflow c binding 은 c/c++ 환경에서 tensorflow 의 기능들을 충분히 실행할 수 있음을 확인할 수 있었고 arm complier 를 사용해 c/c++에서도 fp16 을 사용할 수 있음을 확인하였다. 그리고 yolov3 에서 cfg 파일을 수정함으로써 다양한 mixed precision 조합을 구성하여 최적의 설정을 실험 및 분석을 할 수 있는 프레임워크를 구성할 수 있음을 확인하였다. 추후 본 연구를 발전시켜 높은 정밀도를 요구하는 레이어의 위치와 특성을 파악하고 그것을 바탕으로 네트워크의 경량화 수준과 정확도 수준을 분석해볼 예정이다.

참고문헌

[1] 이경하, 김은희, 딥러닝 모델 경량화 기술 분석, 대전, 한국과학기술정보연구원, 엠에스기획, 2020

[2] 유승목, 이경희, 박재복, 윤석진, 조창식, 정영준, 조일연, 임베디드 시스템용 딥러닝 추론엔진 기술 동향, 전자통신동향분석, 34 권, 제 4 호, 24 쪽

[3] 이경하, 김은희, 딥러닝 모델 경량화 기술 분석, 대전, 한국과학기술정보연구원, 엠에스기획, 2020

[4] 최진욱, 최수혁, 정은성, 딥러닝 모델의 압축 방법에 대한 엣지 장치에서의 성능 평가 연구, 2021 년 6 월 전자공학회 논문지, 제 58 권, 제 6 호, 53 쪽

[5] 장두혁, 이정수, 허준영, 양자화 기반의 모델 압축을 이용한 ONNX 경량화, The Journal of The Institute of Internet, Broadcasting and Communication(IIBC), Vol. 21, NO. 1, page 95

[6] Sharan Narang, Gregory Diamos, Erich Elsen(Baidu Research), Paulius Micikevicius, Jonah Alben, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, Hao Wu(Nvidia), MIXED PRECISION TRAINING, ICLR 2018, Vancouver CANADA, 2018, page 8

[7] Sharan Narang, Gregory Diamos, Erich Elsen(Baidu Research), Paulius Micikevicius, Jonah Alben, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, Hao Wu(Nvidia), MIXED PRECISION TRAINING, ICLR 2018, Vancouver CANADA, 2018, page 4