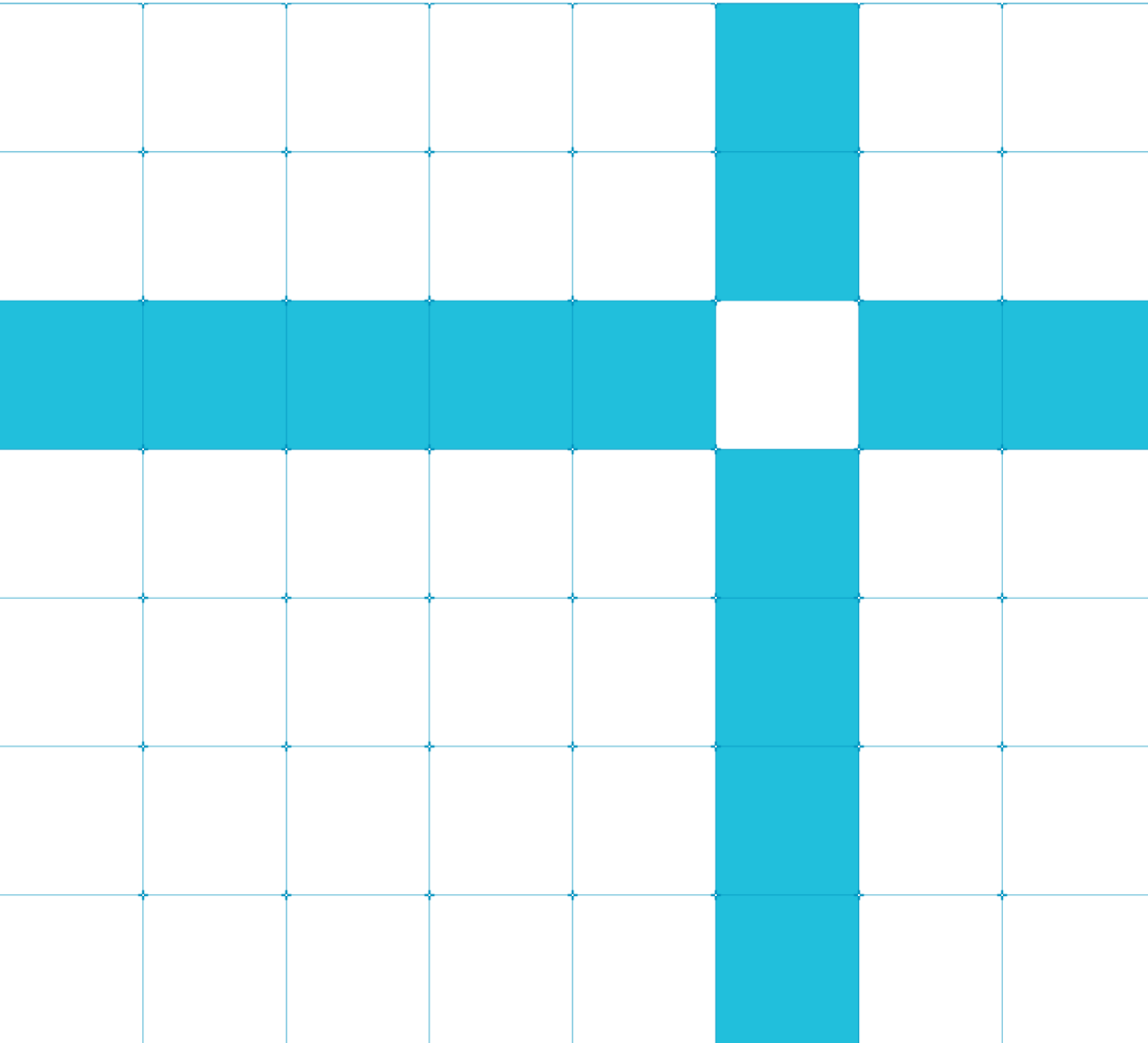
Cross-compiling Arm NN for the Raspberry Pi and TensorFlow

Version 1.11



Cross-compiling Arm NN for the Raspberry Pi and TensorFlow

Release Information

Document History

Version	Date	Confidentiality	Change
1.0	25 October 2018	Non-Confidential	First release.
1.1	10 June 2019	Non-Confidential	Minor fixes to the Building Arm NN section.
1.2	15 July 2019	Non-Confidential	Adds note on known issue to the Testing your build section.
1.3	13 August 2019	Non-Confidential	Adds note on using the correct cross-compiler version to the Before you begin section.
1.4	03 September 2019	Non-Confidential	Adds new sample code in the Extracting Arm NN to your Raspberry Pi and running a sample program section to support the 19.08 release of Arm NN. Also, updates formatting in the Building the Boost library for your Raspberry Pi section.
1.5	11 November 2019	Non-Confidential	Adds two extra commands for downloading the repositories and bundles.
1.6	27 November 2019	Non-Confidential	Adds extra commands for downloading the repositories and bundles.
1.7	05 December 2019	Non-Confidential	Adds extra commands for downloading the repositories and bundles.
1.8	04 March 2020	Non-Confidential	• Adds instructions on including standalone dynamic backend tests. • Updates the codeblock in the Downloading the repositories and bundles section • Updates the codeblocks in step two of the Building Arm NN section. • Updates the codeblocks in the Extracting Arm NN on your Raspberry Pi and running a sample program section and adds in new step three. • Adds to the note in the Testing your build section.
1.9	02 June 2020	Non-Confidential	• Adds to the list of repositories and bundles you must download in the Download the repositories and bundles section. • Adds Generating the TensorFlow Portobuf library, Building FlatBuffers, Building FlatBuffers for Raspberry Pi, and Building TensorFlow Lite sub-sections to the Building the Google Protobuf Library section. • Updates the commands for placing the library files in the Building Arm NN section. • Adds steps for downloading other libraries.
1.10	10 August 2020	Non-Confidential	Removed the AssertZeroExitCode commands.
1.11	02 September 2020	Non-Confidential	• Updates to reflect support for Google protobuf library 3.5.2 • Updates instructions on running Unit Tests.

Non-Confidential Proprietary Notice

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Web Address

www.arm.com

Contents

1 Overview **7**

2 Before you begin **8**

3 Downloading the repositories and bundles **9**

4 Building the Compute Library **10**

5 Building the Boost library for your Raspberry Pi **11**

6 Building the Google Protobuf library **12**
6.1. Building the Google Protobuf library for your virtual machine 12
6.2. Building the Google Protobuf library for your Raspberry Pi 12
6.3. Generating the TensorFlow Protobuf library 12
6.4. Building ONNX 12
6.5. Buidling FlatBuffers 13
6.6. Building FlatBuffers for your Raspberry Pi 13
6.7. Building TensorFlow Lite 13

7 Building Arm NN **14**

8 Extracting Arm NN to your Raspberry Pi and running a sample program **16**
8.1. Creating an archive of cross-compiled libraries, binaries, and directories 16
8.2. Extract the libraries, binaries, and directories to your Raspberry Pi 17
8.3. Compiling and running a sample Arm NN program on your Raspberry Pi 18

9 Testing your build **19**

10 Next steps **20**

1 Overview

This guide covers what we must do to cross-compile Arm NN using an x86_64 system to target a Raspberry Pi. Cross-compiling Arm NN allows us to work around the limited memory of the Raspberry Pi. Read the guide to find out how to build the Compute Library, Boost, Protobuf, TensorFlow, and Arm NN core libraries that you need for compilation. When we finish, we will be able to compile and run programs that use Arm NN on our Raspberry Pis.

Arm estimates that you will take 90-120 minutes to complete this guide.

2 Before you begin

This guide assumes:

- You have a Raspberry Pi board. Arm has tested these instructions on a Raspberry Pi 2 Model B V1.1 that runs Raspian 9, a Rasberry Pi 3 Model B+ that runs Raspian 8, and a Raspberry Pi 4 Model B that runs Raspian 9.
- You compile on a Linux virtual machine. Arm has tested these instructions on an Ubuntu 16.04 virtual machine that runs on MacOS and Windows 10.

You must install the following software tools on your virtual machine:

Git A version control system software developers use for source code management.

Scons An open-source software construction tool.

GNU C and C++ compilers for the armhf architecture The Raspberry Pi uses the armhf Arm architecture.

Note: For the instructions in this guide to work, the cross-compiler version on the host machine must match the compiler version on your Raspberry Pi.

Curl A tool for transferring data to or from a Linux or Unix-like server.

GNU Autoconf A tool for producing configure scripts for building, installing and packaging software.

GNU libtool A generic library support script.

CMake An open-source and cross-platform family of tools for building, testing, and packaging software.

To install the tools you require, open a command window and enter the following commands:

```
sudo apt-get install git
sudo apt-get install scons
sudo apt-get install gcc-arm-linux-gnueabihf
sudo apt-get install g++-arm-linux-gnueabihf
sudo apt-get install curl
sudo apt-get install autoconf
sudo apt-get install libtool
sudo apt-get install cmake
```

3 Downloading the repositories and bundles

Create a directory on your virtual machine to build your Arm NN distribution for your Raspberry Pi.

To create a directory and build your distribution:

1. Enter the following commands:

```
mkdir armnn-pi && cd armnn-pi
export BASEDIR=`pwd`
```

2. Download the Compute Library, Boost, Protobuf, TensorFlow, ONNX, FlatBuffer, and Arm NN git repositories and source bundles. To download the repositories and bundles, enter the following commands:

```
git clone https://github.com/Arm-software/ComputeLibrary.git
git clone https://github.com/Arm-software/armnn
wget https://dl.bintray.com/boostorg/release/1.64.0/source/boost_1_64_0.tar.bz2
tar xf boost_1_64_0.tar.bz2
git clone -b v3.5.2 https://github.com/google/protobuf.git
git clone https://github.com/tensorflow/tensorflow.git
cd tensorflow/
git checkout 590d6eef7e91a6a7392c8ffffb7b58f2e0c8bc6b
git clone https://github.com/onnx/onnx.git
cd onnx
git fetch https://github.com/onnx/onnx.git f612532843bd8e24efeab2815e45b436479cc9ab && git
checkout FETCH_HEAD

wget -O flatbuffers-
1.12.0.tar.gz https://github.com/google/flatbuffers/archive/v1.12.0.tar.gz
tar xf flatbuffers-1.12.0.tar.gz
```

4 Building the Compute Library

Build the Compute Library on your virtual machine. To build the library, enter the following command:

```
cd $BASEDIR/ComputeLibrary
scons extra_cxx_flags="-fPIC" Werror=0 debug=0 asserts=0 neon=1 opencl=0 os=linux arch=armv7a
examples=1
```

Arm estimates that your virtual machine will take approximately 15-20 minutes to execute this command.

5 Building the Boost library for your Raspberry Pi

To build the Boost library for your Raspberry Pi:

1. Enter the following commands:

```
cd $BASEDIR/boost_1_64_0/tools/build
./bootstrap.sh
./b2 install --prefix=$BASEDIR/boost.build
export PATH=$BASEDIR/boost.build/bin:$PATH
```

2. Create a `project-config.jam` file by copying the `user-config.jam` file. To copy the `user-config.jam` file, enter the following command:

```
cp $BASEDIR/boost_1_64_0/tools/build/example/user-config.jam
$BASEDIR/boost_1_64_0/project-config.jam
```

3. Go to the `$BASEDIR/boost_1_64_0/` directory and open the `project-config.jam` file in a text editor. In the `GCC Configuration` section, add the following line:

```
using gcc : arm : arm-linux-gnueabihf-g++ ;
```

4. Save the `project-config.jam` file in the `$BASEDIR/boost_1_64_0/` directory.

5. To complete the build, enter the following commands:

```
cd $BASEDIR/boost_1_64_0
b2 --build-dir=$BASEDIR/boost_1_64_0/build toolset=gcc-arm link=static cxxflags=-fPIC --with-
filesystem --with-test --with-log --with-program_options install --prefix=$BASEDIR/boost
```

Arm estimates that your virtual machine will take approximately 15 minutes to execute these commands.

6 Building the Google Protobuf library

You use two versions of the Google Protobuf library. One version of the library runs on your virtual machine and the other runs on your Raspberry Pi.

6.1. Building the Google Protobuf library for your virtual machine

To build the Google Protobuf library for your virtual machine:

1. Enter the following commands:

```
cd $BASEDIR/protobuf
git submodule update --init --recursive
./autogen.sh
./configure --prefix=$BASEDIR/protobuf-host
make
```

   Arm estimates that your virtual machine will take approximately 15 minutes to execute these commands.

2. Enter the following commands:

```
make install
make clean
```

6.2. Building the Google Protobuf library for your Raspberry Pi

To build the Google Protobuf library for your Raspberry Pi:

1. Enter the following commands:

```
./configure --prefix=$BASEDIR/protobuf-arm --host=arm-linux CC=arm-linux-gnueabihf-gcc
CXX=arm-linux-gnueabihf-g++ --with-protoc=$BASEDIR/protobuf-host/bin/protoc
make
```

   Arm estimates that your virtual machine will take approximately 15 minutes to execute these commands.

2. Enter the following command:

```
make install
```

6.3. Generating the TensorFlow Protobuf library

To generate the TensorFlow library, enter the following commands:

```
cd $BASEDIR/tensorflow
../armnn/scripts/generate_tensorflow_protobuf.sh ../tensorflow-protobuf ../protobuf-host
```

6.4. Building ONNX

To build ONNX, enter the following commands:

```
cd $BASEDIR/onnx

export LD_LIBRARY_PATH=$BASEDIR/protobuf-host/lib:$LD_LIBRARY_PATH
```

```
$BASEDIR/protobuf-host/bin/protoc onnx/onnx.proto --proto_path=. --proto_path=$BASEDIR/protobuf-host/include --cpp_out $BASEDIR/onnx
```

6.5. Buidling FlatBuffers

To build FlatBuffers, enter the following commands:

```
cd flatbuffers-1.12.0
rm -f CMakeCache.txt
rm -rf build
mkdir build
cd build
CXXFLAGS="-fPIC" cmake .. -DFLATBUFFERS_BUILD_FLATC=1 \
-DCMAKE_INSTALL_PREFIX:PATH=$WORKING_DIR/flatbuffers
make all install
```

6.6. Building FlatBuffers for your Raspberry Pi

To build FlatBuffers for your Raspberry Pi, enter the following commands:

```
cd $BASEDIR/flatbuffers-1.12.0
mkdir build-arm32
cd build-arm32
CXXFLAGS="-fPIC" cmake .. -DCMAKE_C_COMPILER=/usr/bin/arm-linux-gnueabihf-gcc \
-DCMAKE_CXX_COMPILER=/usr/bin/arm-linux-gnueabihf-g++ \
-DFLATBUFFERS_BUILD_FLATC=1 \
-DCMAKE_INSTALL_PREFIX:PATH=$BASEDIR/flatbuffers-arm32 \
-DFLATBUFFERS_BUILD_TESTS=0
make all install
```

6.7. Building TensorFlow Lite

To build TensorFlow Lite, enter the following commands:

```
cd $BASEDIR
mkdir tflite
cd tflite
cp $BASEDIR/tensorflow/tensorflow/lite/schema/schema.fbs .
$BASEDIR/flatbuffers-1.12.0/build/flatc -c --gen-object-api --reflect-types --reflect-names schema.fbs
```

7 Building Arm NN

To build Arm NN:

1. Enter the following commands:

```
cd $BASEDIR/armnn
mkdir build
cd build
```

2. Place the library files you require in the build directory. To place the library files, enter:

```
cmake .. -DCMAKE_LINKER=/usr/bin/arm-linux-gnueabihf-ld \
-DCMAKE_C_COMPILER=/usr/bin/arm-linux-gnueabihf-gcc \
-DCMAKE_CXX_COMPILER=/usr/bin/arm-linux-gnueabihf-g++ \
-DCMAKE_C_COMPILER_FLAGS=-fPIC \
-DCMAKE_CXX_FLAGS=-mfpu=neon \
-DARMCOMPUTE_ROOT=$BASEDIR/ComputeLibrary \
-DARMCOMPUTE_BUILD_DIR=$BASEDIR/ComputeLibrary/build \
-DBOOST_ROOT=$BASEDIR/boost \
-DTF_GENERATED_SOURCES=$BASEDIR/tensorflow-protobuf \
-DBUILD_TF_PARSER=1 \
-DBUILD_ONNX_PARSER=1 \
-DONNX_GENERATED_SOURCES=$BASEDIR/onnx \
-DBUILD_TF_LITE_PARSER=1 \
-DTF_LITE_GENERATED_PATH=$BASEDIR/tflite \
-DFLATBUFFERS_ROOT=$BASEDIR/flatbuffers-arm32 \
-DFLATC_DIR=$BASEDIR/flatbuffers-1.12.0/build \
-DPROTOBUF_ROOT=$BASEDIR/protobuf-arm \
-DARMCOMPUTENEON=1 \
-DARMNNREF=1
make
```

Arm estimates that your virtual machine will take approximately 12 minutes to execute these commands.

If you want to include standalone sample dynamic backend tests, add the following argument to enable the tests and the dynamic backend path to the CMake command:

```
-DSAMPLE_DYNAMIC_BACKEND=1 -DDYNAMIC_BACKEND_PATHS=<the location of the sample dynamic backend
on Raspberry Pi>
```

Also, build the standalone sample dynamic backend after building Arm NN using the following commands:

```
#Set the versions based on /armnn/include/armnn/Version.hpp
ARMNN_MAJOR_VERSION=<ARMNN_MAJOR_VERSION>
ARMNN_MINOR_VERSION=<ARMNN_MINOR_VERSION>
ARMNN_PATCH_VERSION=<ARMNN_PATCH_VERSION>
cd $BASEDIR/armnn/src/dynamic/sample
mkdir build
cd build
cmake -DCMAKE_LINKER=/usr/bin/arm-linux-gnueabihf-ld \
-DCMAKE_C_COMPILER=/usr/bin/arm-linux-gnueabihf-gcc \
-DCMAKE_CXX_COMPILER=/usr/bin/arm-linux-gnueabihf-g++ \
-DCMAKE_CXX_FLAGS=--std=c++14 \
-DCMAKE_C_COMPILER_FLAGS=-fPIC \
-DBOOST_ROOT=$BASEDIR/boost \
-DBoost_SYSTEM_LIBRARY=$BASEDIR/boost/lib/libboost_system.a \
-DBoost_FILESYSTEM_LIBRARY=$BASEDIR/boost/lib/libboost_filesystem.a \
```

```
-DARMNN_PATH=$BASEDIR/armnn/build/libarmnn.so.$ARMNN_MAJOR_VERSION.$ARMNN_MINOR_VERSION ..
make
```

8 Extracting Arm NN to your Raspberry Pi and running a sample program

8.1. Creating an archive of cross-compiled libraries, binaries, and directories

To create an archive of cross-compiled libraries, binaries, and directories:

1. Copy the following libraries, binaries, and directories from your virtual machine. To copy these libraries, binaries, and directories enter the following commands:

```
#Set the versions based on /armnn/include/armnn/Version.hpp
ARMNN_MAJOR_VERSION=<ARMNN_MAJOR_VERSION>
ARMNN_MINOR_VERSION=<ARMNN_MINOR_VERSION>
ARMNN_PATCH_VERSION=<ARMNN_PATCH_VERSION>
cd $BASEDIR
mkdir armnn-dist
mkdir armnn-dist/armnn
mkdir armnn-dist/armnn/lib
cp $BASEDIR/armnn/build/libarmnn.so.$ARMNN_MAJOR_VERSION.$ARMNN_MINOR_VERSION $BASEDIR/armnn-
dist/armnn/lib
ln -s libarmnn.so.$ARMNN_MAJOR_VERSION.$ARMNN_MINOR_VERSION $BASEDIR/armnn-
dist/armnn/lib/libarmnn.so.$ARMNN_MAJOR_VERSION
ln -s libarmnn.so.$ARMNN_MAJOR_VERSION $BASEDIR/armnn-dist/armnn/lib/libarmnn.so
cp $BASEDIR/armnn/build/libarmnnTfParser.so.$ARMNN_MAJOR_VERSION.$ARMNN_MINOR_VERSION
$BASEDIR/armnn-dist/armnn/lib
ln -s libarmnnTfParser.so.$ARMNN_MAJOR_VERSION.$ARMNN_MINOR_VERSION $BASEDIR/armnn-
dist/armnn/lib/libarmnnTfParser.so.$ARMNN_MAJOR_VERSION
ln -s libarmnnTfParser.so.$ARMNN_MAJOR_VERSION $BASEDIR/armnn-
dist/armnn/lib/libarmnnTfParser.so
cp $BASEDIR/protobuf-arm/lib/libprotobuf.so.15.0.0 $BASEDIR/armnn-
dist/armnn/lib/libprotobuf.so
cp $BASEDIR/protobuf-arm/lib/libprotobuf.so.15.0.0 $BASEDIR/armnn-
dist/armnn/lib/libprotobuf.so.15
cp -r $BASEDIR/armnn/include $BASEDIR/armnn-dist/armnn/include
cp -r $BASEDIR/boost $BASEDIR/armnn-dist/boost
```

2. To test that your build of Arm NN works correctly, you will need to run Unit Tests. To enable the running of Unit Tests, also copy the `libtimelineDecoder.so`, `libtimelineDecoderJson.so` and `libarmnnBasePipeServer.so` libraries from your virtual machine.

Note: If you are also building for the Open Neural Network Exchange (ONNX) format and TensorFlow Lite, you also have to copy and link the `libarmnnOnnxParser.so` and `libarmnnTfLiteParser.so` libraries.

3. Copy the following dynamic backend related files from your virtual machine. To copy these files, enter the following commands:

```
mkdir -p $BASEDIR/armnn-dist/src/backends/backendsCommon/test/
cp -r $BASEDIR/armnn/build/src/backends/backendsCommon/test/testSharedObject $BASEDIR/armnn-
dist/src/backends/backendsCommon/test/testSharedObject/

cp -r $BASEDIR/armnn/build/src/backends/backendsCommon/test/testDynamicBackend/
$BASEDIR/armnn-dist/src/backends/backendsCommon/test/testDynamicBackend/
cp -r $BASEDIR/armnn/build/src/backends/backendsCommon/test/backendsTestPath1/ $BASEDIR/armnn-
dist/src/backends/backendsCommon/test/backendsTestPath1/
```

```
mkdir -p $BASEDIR/armnn-dist/src/backends/backendsCommon/test/backendsTestPath2
cp
$BASEDIR/armnn/build/src/backends/backendsCommon/test/backendsTestPath2/Arm_CpuAcc_backend.so
$BASEDIR/armnn-dist/src/backends/backendsCommon/test/backendsTestPath2/

ln -s Arm_CpuAcc_backend.so $BASEDIR/armnn-
dist/src/backends/backendsCommon/test/backendsTestPath2/Arm_CpuAcc_backend.so.1
ln -s Arm_CpuAcc_backend.so.1 $BASEDIR/armnn-
dist/src/backends/backendsCommon/test/backendsTestPath2/Arm_CpuAcc_backend.so.1.2
ln -s Arm_CpuAcc_backend.so.1.2 $BASEDIR/armnn-
dist/src/backends/backendsCommon/test/backendsTestPath2/Arm_CpuAcc_backend.so.1.2.3
cp
$BASEDIR/armnn/build/src/backends/backendsCommon/test/backendsTestPath2/Arm_GpuAcc_backend.so
$BASEDIR/armnn-dist/src/backends/backendsCommon/test/backendsTestPath2/
ln -s nothing $BASEDIR/armnn-
dist/src/backends/backendsCommon/test/backendsTestPath2/Arm_no_backend.so

mkdir -p $BASEDIR/armnn-dist/src/backends/backendsCommon/test/backendsTestPath3

cp -r $BASEDIR/armnn/build/src/backends/backendsCommon/test/backendsTestPath5/ $BASEDIR/armnn-
dist/src/backends/backendsCommon/test/backendsTestPath5
cp -r $BASEDIR/armnn/build/src/backends/backendsCommon/test/backendsTestPath6/ $BASEDIR/armnn-
dist/src/backends/backendsCommon/test/backendsTestPath6

mkdir -p $BASEDIR/armnn-dist/src/backends/backendsCommon/test/backendsTestPath7

cp -r $BASEDIR/armnn/build/src/backends/backendsCommon/test/backendsTestPath9/ $BASEDIR/armnn-
dist/src/backends/backendsCommon/test/backendsTestPath9

mkdir -p $BASEDIR/armnn-dist/src/backends/dynamic/reference
cp $BASEDIR/armnn/build/src/backends/dynamic/reference/Arm_CpuRef_backend.so $BASEDIR/armnn-
dist/src/backends/dynamic/reference/
```

If you enable the standalone sample dynamic backend tests, also copy the dynamic backend file using the following commands:

```
mkdir -p $BASEDIR/armnn-dist/src/dynamic/sample
cp $BASEDIR/armnn/src/dynamic/sample/build/libArm_SampleDynamic_backend.so $BASEDIR/armnn-
dist/src/dynamic/sample/
cp $BASEDIR/armnn/samples/DynamicSample.cpp $BASEDIR/armnn-dist
```

4. Copy the Unit Tests and a sample Arm NN program. To copy this test and program, enter the following commands:

```
cp $BASEDIR/armnn/build/UnitTests $BASEDIR/armnn-dist
cp $BASEDIR/armnn/samples/SimpleSample.cpp $BASEDIR/armnn-dist
```

5. Create the archive for your Raspberry Pi. To create the archive, enter the following command:

```
tar -czf $BASEDIR/armnn-dist.tar.gz armnn-dist
```

8.2. Extract the libraries, binaries, and directories to your Raspberry Pi

Extract the libraries, binaries, and directories to your Raspberry Pi. To extract the libraries, binaries, and directories enter the following commands:

```
cd /home/pi
tar -xzf /home/pi/armnn-dist.tar.gz
```

8.3. Compiling and running a sample Arm NN program on your Raspberry Pi

To compile and run a sample C++ program that uses Arm NN on your Raspberry Pi:

1. Enter the following commands:

```
export LD_LIBRARY_PATH=/home/pi/armnn-dist/armnn/lib
cd /home/pi/armnn-dist
```

   To compile the program, enter the following commands:

```
g++ SimpleSample.cpp -I/home/pi/armnn-dist/armnn/include -I/home/pi/armnn-dist/boost/include -
L/home/pi/armnn-dist/armnn/lib -larmnn -larmnnTfParser -lprotobuf -o SimpleSample
```

2. To run the program, enter the following commands:

```
./SimpleSample
```

   The console returns the following:

```
Please enter a number:
```

   Enter your number. For example:

```
345
```

   The console returns the following:

```
Your number was 345
```

3. If you enable the standalone sample dynamic backend tests, you can run a sample dynamic backend program as a test.

   To compile the program, enter the following commands:

```
g++ DynamicSample.cpp -I/home/pi/armnn-dist/armnn/include -I/home/pi/armnn-dist/boost/include
-L/home/pi/armnn-dist/armnn/lib -larmnn -larmnnTfParser -lprotobuf -o DynamicSample
```

   To run the program, enter the following command:

```
./DynamicSample
```

   If the test is successful, the console returns the following:

```
Addition operator result is {15,11}
```

   If the test fails, the console returns an error message describing the reason for failure.

9 Testing your build

To test that your build of Arm NN works correctly, run the Unit Tests. To run the Unit Tests, enter the following:

```
export LD_LIBRARY_PATH=/home/pi/armnn-dist/armnn/lib
cd /home/pi/armnn-dist
./UnitTests
```

If the tests are successful, they output the following to the console:

```
*** No errors detected
```

If some of the tests are unsuccessful, go back through the steps and check that all the commands have been entered correctly.

Note:

- Arm is aware of an issue resulting in a NeonTimerMeasure unit test error on Raspberry Pi. The implementation by the Raspberry Pi of the timing libraries Arm NN uses to get its timing data causes this error. You can safely ignore this error.
- External profiling for Arm NN on the Raspberry Pi platform is currently not fully supported and results in some External Profiling unit tests failing.

10 Next steps

You are now ready to compile and run programs that use Arm NN on your Raspberry Pi.

Find out how to deploy Caffe and TensorFlow models.