Running TensorFlow Lite for Microcontrollers on Contiki-NG

Atis E

Follow

--

Listen

Share

A guide for running Deep Learning models on Nordic Semiconductors nRF52 and nRF53 series devices

Background

TensorFlow Lite for Microcontrollers (TFLM)

TFLM is a framework that allows to execute a subset ofTensorFlow Machine Learning / Deep Learning models

on microcontrollers. A microcontroller is a device with very limited computational capacities, typically running a real-time operating system and having no virtual memory or extended disk storage available. TFLM allows machine learning engineers to bypass these challenges, as it offers a C++ library for run-time interpretation and inference of .tflite models. The library can be linked with many existing OS, for example Zephyr and ARM mBed.

Contiki-NG

Contiki is one of the original sensor network operating systems, started in the early 2000-s by Adam Dunkels. It has one of the smallest TCP/IP stacks in existence and to this day is actively used in research. Contiki-NG is the Next Generation version of Contiki, a fork of the original system and is currently actively developed. Contiki-NG keeps the good things of the Contiki, including the powerful network stack (IEEE 802.15.4, 6LoWPAN, TSCH / 6TiSCH, RPL and TCP/IP protocols) and the lock-free based cooperative threading model ("protothreads"). The added focus of Contiki-NG is next-generation devices and standard protocols, with the aim to serve the needs of on modern Internet of Things

(IoT) applications.

Applications

Microcontrollers is a new research frontier for Deep Learning models. Some of the cool things that people have done on these devices so far include:

- activity recognition from inertial sensor data
- a limited form of voice recognition
- low-resolution image recognition
- anomaly detection in sensor data streams

Setting up the dependencies

TensorFlow Lite for Microcontrollers (TFLM)

Since June 2021 TFLM has it's own repository at GitHub. To get it:

Contiki-NG

The code of this tutorial is in the author's personal fork of Contiki-NG. The repositorywith the code is based on Contiki-NG release version 4.7. You can get it by cloning the repository (the example is in the main

branch):

Target hardware

We'll be using Nordic Semiconductors nRF series devices for the demo example. The nRF52840 is based on Cortex-M4F System-on-Chip, and the more recent nRF5340 on Cortex-M33. The information in this article should be easy to apply to other ARM Cortex-M based platforms as well, as long as the platforms are supported by Contiki / Contiki-NG. One thing that you're going to need to follow this article is the GCC toolchain for the target architecture; that is, arm-none-eabi-gcc and friends should be installed in your $PATH.

Target application

We're just going to reuse the "Hello World" example from the TFLM repository. The example aims to predict a simple sin() function using a .tflite model.

The C++ code of this example was ported to Contiki-NG — it was slightly modified and simplified to make it compile for the target OS. The alternative way would have been to add an nRF platform with Contiki-NG OS support to the TFLM codebase. However, adding new target platforms to TFLM is nontrivial and is expected to change in the future.

Building an example

TensorFlow Lite for Microcontrollers (TFLM)

Go to the TFLM top level directory and build the microlite target for the target microcontroller architecture:

Let's break down the command in details:

- -f tensorflow/lite/micro/tools/make/Makefile — this tells to use the main Makefile of the TFLM
- TARGET=cortex_m_generic — this selects the target microcontroller family. Some other options, for example for RISC and Xtensa are available.
- TARGET_ARCH=cortex-m4+fp — this selects the ARM Cortex-M core model. For the nRF52 series, cortex-m4 is fine, but nRF32 series use cortex-m33. The extra +fp means to enable support for hardware floating point. Check whether your microcontroller has this hardware module.
- OPTIMIZED_KERNEL_DIR=cmsis_nn — this enables the use of the CMSIS-NN library optimizations. This gives a major performance boost on ARM Cortex core microcontrollers.
- microlite — this is the name of the target, currently telling to just build the library instead of a specific example application.

Expected result:

Contiki-NG

In order to start using Contiki-NG, we recommend to get the Docker image provided by the maintainers.

Once you have got the right branch of the OS, go to the tensorflow-lite-micro example directory and create a link to the microlite library created in the first step. The exact path depends on your folder structure — change it

!

Now also edit the Makefile to point to the correct TFLM directory, by changing the second line:

After that, build the example normally for the target:

TARGET=nrf selects the Contiki-NG hardware platform that supports several nRF devices, and BOARD=nrf52840/dk selects the nRF52840 development kit (PCA10056) in particular. You can run the command make TARGET=nrf BOARD=nrf52840/dk savetarget to store the target and board in a local file; then you can skip explicitly setting them in the next commands.

Expected result:

Connect the devkit and flash the application:

Now connect with a serial port reader program and observe the output.

Expected output:

The x_value

is the "true" value of the sin() function on the given parameter (i.e. the target that must be learned), and y_value

is the number calculated from the output returned by the model (i.e. the guess made by the model).

Looking at the code

The neural network model

The example code already comes with the model provided in a C file called nn-model.c.

The model is converted from the .tflite file provided in the TFLM Hello World example application by using the xxd command. To regenerate the C file:

Change the path to match your directory structure, and afterwards edit the C file to rename the model back to hello_world_tflite.

Warning #1:

the ARM processor requires that all memory accesses are aligned, otherwise it throws a hardware exception. After converting the model, make sure to add alignas(8)

attribute to the byte code array, otherwise the program will crash. Seethe code here for an example.

Warning #2:

if you attempt to use a custom model, be aware that not all TensorFlow Lite operations (kernels) are supported by the Tensorflow Lite Micro library! This means that some .tflite models will fail at runtime, during the setup stage.

The C++ code for model setup and inference

The important bits are in the filenn.cpp, which shows how to prepare the model and run inference in it.

The functions nn_setup() and nn_run_inference use the C calling convention and are simply wrappers around the C++ functions setup() and loop() from the example. The main process in a Contiki-NG application must be written in C, which means that it cannot call C++ functions directly.

The functions setup() and loop() are taken from the TFLM Hello World example; their names are a convention used in Arduino sketches. I have changed the return type of these functions to allow to return an error code.

The most important bit in the setup() function is the call to interpreter->AllocateTensors(). This step may fail because of different reasons, e.g. the model uses unsupported operations or its run-time overhead is too large to fit in the pre-allocated tensor_arena variable.

Be aware that the model itself must also be placed in RAM, not in the flash memory, as it may be modified by the TFLM runtime library: don’t be tempted to add const

in front of the array that contains the model’s binary code.

The loop() function sets up inputs (a single number in this case), peforms inference by calling interpreter->Invoke(), and prints out the dequantized output from the inference.

Acknowledgements

This work was supported by the ERDF Activity 1.1.1.2 “Post-doctoral Research Aid’’ (№1.1.1.2/VIAA/2/18/282). It was part of the FEATURE project developed at the Institute of Electronics and Computer Science (EDI), https://www.edi.lv/en/