ARL-MR-1031 ● JUNE 2021

TensorFlow Lite Extension to OpenNMT-tf Documentation

by Gerardo Cervantes and Stephen LaRocca

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-MR-1031 ● JUNE 2021

TensorFlow Lite Extension to OpenNMT-tf Documentation

Gerardo Cervantes
Advanced Resource Technologies, Inc.

Stephen LaRocca
Computational and Information Sciences Directorate,
DEVCOM Army Research Laboratory

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
June 2021	Memorandum Report	September 2020–March 2021

4. TITLE AND SUBTITLE
TensorFlow Lite Extension to OpenNMT-tf Documentation

5a. CONTRACT NUMBER
W911QX20D0012

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Gerardo Cervantes and Stephen LaRocca

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
DEVCOM Army Research Laboratory
ATTN: FCDD-RLC-IB
Adelphi, MD 20783-1138

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-MR-1031

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release: distribution unlimited.

13. SUPPLEMENTARY NOTES
ORCID ID(s): Cervantes, 0000-0002-4392-5017; LaRocca, 0000-0003-3341-5520

14. ABSTRACT
Since its release in 2017, the OpenNMT project has provided open development tools for Neural Machine Translation (NMT) including machine learning inference with artificial neural network models on Android platforms. Rapid advances in OpenNMT methods were achieved using TensorFlow since 2018; however, most of these advances were not deployable for use on Android platforms pending completion of the TensorFlow Lite library. The US Army Combat Capabilities Development Command Army Research Laboratory Shareable Components project team closely tracked progress on TensorFlow Lite and succeeded in implementing a new method for converting OpenNMT models from standard TensorFlow to the Lite variant. Deployable on Android devices, these converted models provide important gains in execution speed while occupying less space. A pull request submitted to the OpenNMT was approved and implemented in February 2021. This report describes how OpenNMT developers using TensorFlow can convert several types of models to TensorFlow Lite for use on Android platforms.

15. SUBJECT TERMS
Machine Translation, deep learning, Android, open development tools, TensorFlow Lite

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Stephen LaRocca
Unclassified	Unclassified	Unclassified	UU	18	19b. TELEPHONE NUMBER (Include area code) (301) 394-3198

Contents

List of Tables

1. Introduction

1.1 Document Contents

This report covers the new extension to the popular Machine Translation (MT) open-source package OpenNMT-tf. This extension allows models that were trained using this package to be mobile-compatible with TensorFlow Lite.

1.2 Purpose

The purpose of this report is to convert MT models trained with the OpenNMT-tf package to be mobile friendly with TensorFlow Lite. The conversion of these models to TensorFlow Lite improves the models' run time on mobile environments and allows the models to run using the onboard graphics processing unit (GPU) of the mobile device.

1.3 Distribution

The OpenNMT-tf open-source package can be found on GitHub, https://github.com/OpenNMT/OpenNMT-tf. This code is a pull request to the open-source library that was reviewed and accepted by the OpenNMT developers.

1.4 Tools

The following tools are needed in order to convert OpenNMT models to TensorFlow Lite:

- Python 3.5+
- Python pip
- OpenNMT-tf package, https://pypi.org/project/OpenNMT-tf/

1.5 Convertible Models

The following models from OpenNMT are convertible to TensorFlow Lite:

- NMTSmallV1
- NMTMediumV1
- NMTBigV1
- Luong Attention

2. Running the Conversion

2.1 Installing

To install and run the extension, first download Python (recommended version is Python 3.7). Make sure this download adds Python and pip onto your PATH environment variables so you can use the commands in the command line.

To install the extension in the command line, run this command to get the necessary Python packages:

```
pip install OpenNMT-tf
```

2.2 Running

Make sure to install the Python packages before trying to run the conversion.

To run the conversion from the command line, navigate to the main directory of the code (where the README resides).

To convert an OpenNMT-tf model to TensorFlow Lite, you must first train a model. A small guide on training a model can be found here: https://opennmt.net/OpenNMT-tf/quickstart.html.

Run this command with your model-specific options to convert the model to TensorFlow Lite:

```
python opennmt/bin/main.py --model_type NMTSmallV1 --config
opensubs/data.yml export --output_dir
./opensubs/smallnmt/tflite --export_format tflite
```

Parameter descriptions are as follows:

- *--model_type* – The model architecture you are trying to load to convert to TFLite; these can be found in opennmt/models/catalog.py.
- *--config* – The data configuration file, written in YAML. The file also contains the path to the model. Parameters can be found here: https://opennmt.net/OpenNMT-tf/configuration.html
- *export* – This tells the algorithm that you want to save the model as a TensorFlow Lite model.
- *--output_dir* – This is the directory where you want your model to be saved.
- --format – This tells OpenNMT to export as a TensorFlow Lite model; TensorFlow Lite options are “tflite” and “tflite_float16”.

The conversion should take about 1–2 min to run.

2.3 Data Configuration File

Running anything on OpenNMT requires that you give OpenNMT a data configuration file of structure YAML. All the possible data configurations can be found here: https://opennmt.net/OpenNMT-tf/configuration.html

The data.yml file created when using the quick-start guide in model training is the data configuration file. This data configuration file is needed to run any operation in OpenNMT and is required to convert models to TensorFlow Lite.

One optional parameter added to OpenNMT with the code extension is

```
params:
    tflite_output_size: 250
```

This parameter tells the algorithm what size array to make the output; if this optional parameter is not provided, then the default value used is 250.

2.4 Vocabulary Files and IDs Used for Running

2.4.1 Acquiring Vocabulary Files

During the process of training an OpenNMT-tf model, two vocabulary files will have been created. The location of these vocabulary files are specified in the data configuration file and are required to convert a model to TensorFlow Lite and to run a TensorFlow Lite–converted model on Android.

2.4.2 Description

The vocabulary files consist of a list of words of the language separated by new line characters, where each word in the language is given a unique number, commonly referred to as the word ID. To convert a word not in the vocabulary to an ID, you will have to mark it as an unknown word ID, which will be the biggest unique ID available plus 1.

2.4.3 Use

Running the TensorFlow Lite model requires an array of integers instead of text. To translate text using the model, you must convert the text to IDs using the vocabulary file of the language you are converting from. This is achieved by separating each word with white space. Then, you must look up the unique ID of each word in the vocabulary file. After getting the IDs of all of the words, you can store them in an array and pass the IDs to the model to get a translation. The

returned translation from the model will be integer IDs convertible to text using the other vocabulary file.

2.4.4 Example Code Snippet

The following Android code snippet produces a HashMap that maps a word to the unique ID given in the file.

```
public HashBiMap<String, Integer> readVocab(InputStream file){
        try {
            BufferedReader brFile = new BufferedReader(new
InputStreamReader(file));
            HashBiMap<String, Integer> vocab =
HashBiMap.create();
            String wordRead = brFile.readLine();
            int index = 0;
            while(wordRead != null){
                vocab.put(wordRead, index);
                index += 1;
                wordRead = brFile.readLine();
            }
            return vocab;
        }
        catch(IOException e) {
            return null;
        }
    }
```

2.5 Running on Android

This section provides portions of code that will load and run the model on Android devices.

2.5.1 Gradle Dependencies

Adding these dependencies in the Android Studio Gradle file (build.gradle) will allow TensorFlow Lite models to be run on Android. The following lines provide the "nightly" version for TensorFlow, which is the beta version.

```
implementation 'org.tensorflow:tensorflow-lite:0.0.0-nightly'
implementation 'org.tensorflow:tensorflow-lite-select-tf-ops:0.0.0-
nightly'
implementation 'org.tensorflow:tensorflow-lite-support:0.0.0-nightly'
```

2.5.2 Loading the Model

The following code is used to load the model and assumes you have stored the saved model to the assets folder. The code will look into the assets folder to find the model you specified. The *modelPath* variable should be set to the location of

the model. The *NUM_LITE_THREADS* should be set to an integer that tells it how many threads to use when running the model. Tested values are 1 and 4.

```
        AssetManager assetManager =
this.context.getResources().getAssets();
        ByteBuffer buffer = loadModelFile(assetManager,
modelPath);
        if (buffer == null){
            Log.e("nmt-tf", "Could not load model");
            return;
        }
        Interpreter.Options opt = new Interpreter.Options();
        opt.setNumThreads(NUM_LITE_THREADS);
        tflite = new Interpreter(buffer, opt);
```

2.5.3 Running the Model

The following code will run the model with the given input. Since the function takes in IDs, you should read and load the vocabulary files onto the application. You should also have functions to convert sentences to an integer array of the respective IDs as well as a function to convert an integer array of IDs back to a sentence.

```
        int[] input_ids = sentenceToIds("Hello World");
        int[] output_ids = new int[250];
        tflite.run(input_ids, output_ids);
        String translatedSentence = IdsToSentence(output_ids);
        System.out.println("Translated Sentence: " +
translatedSentence);
```

output_ids is initialized to the array. After *tflite.run* is called, it will modify this variable to have the result of the model inference. *output_ids* should be the same size as *tflite_output_size*.

2.6 Quantization Options

Quantizing the model can be useful for reducing the model size and increasing the run time.

2.6.1 Model Quantization Options

- Normal model
- 16-bit float quantized

2.6.2 How

The model can easily be quantized by modifying the argument value "format" when exporting the model to TensorFlow Lite.

2.6.3 Advantages

The model size is reduced by half, and according to TensorFlow, the run time should be faster.

2.6.4 Future

For an additional speedup, it may be worth quantizing differently. The following options require a representative dataset to be able to smartly know how to quantize the variables to 8 bits:

- Dynamic range quantization
    - Weights to 8-bit integers; outputs are still 32 bit
    - Four times smaller, two to three times speedup
- Full integer quantization
    - Weights to 8-bit integers
    - Four times smaller, three times plus speedup

These are great options to consider if the models need to run faster.

2.7 Tests Created

The tests listed in Table 1 check if the TensorFlow Lite function produces the same results as the original inference function and if the models could be converted.

Table 1 Automated conversion tests

Test name	Model	Test description
testTFLiteOutput	Luong Attention NMTBigV1	Checks that the model prediction is the same when outputting with the TensorFlow Lite conversion.
testTFLiteOutputFile	Luong Attention NMTBigV1	Checks that the code is able to convert and output the model into a file.

The tests can be found and ran from the following file: opennmt/tests/tflite_test.py.

2.8 Function Converted Description

The function being converted is named *infer_tflite* and can be found in the file opennmt/models/sequencetosequence.py. This function runs when you predict using the TensorFlow Lite model that was converted; it runs on any model that uses SequenceToSequence.

2.8.1 Parameter

The function that takes in one parameter is named “ids”.

To run IDs in Python, you give the function a 32-bit integer Tensor, which can be any size as long as it is 1-D. When running on Android, you will have to feed in a 1-D integer array; this array can be any size. The parameter is an integer array and not a String because the function takes in an ID for each word in the sentence. Refer to Section 2.4, Vocabulary Files and IDs Used for Running, to learn how to convert the sentence you want translated back to IDs.

2.8.2 Returns

The function returns a 1-D integer array. The size of the array is by default 250 unless a size was specified in the data configuration file. Section 2.3, Data Configuration File, covers how to specify a different output size.

In Python, the TensorFlow Lite function will return a 32-bit integer Tensor. Running on Android will return a 32-bit integer array. The function does not return a String; it will return IDs, which will have to be converted back to a sentence to find the translation of the sentence. Refer to Section 2.4, Vocabulary Files and IDs Used for Running, to go from IDs back to a sentence.

3. Additional Technical Details

3.1 Code Modifications/Additions

This section is a brief overview of why some files were modified or added, which files changed, and the reason for the modification.

Source file changes are the changes that were made to add the TensorFlow Lite support. Testing files are files that were modified or added to include automated unit tests to check if the TensorFlow Lite conversion works.

3.1.1 Source

The files listed in Table 2 were modified to implement the TensorFlow Lite conversion option.

Table 2 OpenNMT-tf source code modifications

File	Reason
opennmt/utils/exporters.py	Adds "tflite" as a method to export, making models mobile friendly. This adds the class TFLiteExporter to handle exporting. The model produces an error if it is not compatible with TensorFlow Lite. Otherwise, the model is loaded and converted.
opennmt/inputters/text_inputter.py	Adds the function *tflite_call* that gets the word embeddings in a way that is safe for TensorFlow Lite. The fix and problem description can be found in TensorFlow issue #42410.
opennmt/models/sequence_to_sequence.py	Creates the custom inference function, which is the concrete function that will be converted. A parameter is added to *_dynamic_decode* to tell the function that it is running in a TensorFlow Lite safe way, and if running with TensorFlow Lite, then it returns IDs.
opennmt/decoders/decoder.py	Modifies the *dynamic_decode* function to use the modified word embedding function.
opennmt/utils/decoding.py	Modifies the decoding. The original function uses *TensorArrays,* which are not compatible with TensorFlow Lite unless they are specified as static and are pre-allocated.

3.1.2 Testing

The files listed in Table 3 were modified or added to create the tests.

Table 3 Testing files modified

File	Reason
opennmt/tests/tflite_test.py	Contains four test cases created to check if the models NMTBigV1 and Luong Attention produce the same output when running as a TensorFlow Lite model and to check if they were able to convert and save the model.

3.2 Conversion Process

The conversion runs through the command line. Running details are provided in Section 2.2.

3.2.1 Initial Running Tests

After arguments are parsed, the function *save_tflite* is called, which loads the model structure and restores the tensor weights of the model. To test for normal operation, it runs on translating the small example "Hello World". This is to ensure that the

model runs and not to verify that the output is correct. The model then tries to run the custom function created in this TensorFlow Lite modification, which runs the model in a way that is TensorFlow Lite safe; this will be explained later in the report.

3.2.2 TensorFlow Conversion Methods

There are three different ways to convert a model to TensorFlow Lite.

- From a TensorFlow Saved model
    - Converting from a Saved model was problematic because the word embeddings were not being saved onto the TensorFlow Lite model being converted. This resulted in an error.
- From a Keras model
    - We cannot convert using this method because our model is not a Keras model.
- From a concrete function
    - This method worked the best and was used to convert the model to TensorFlow Lite. TensorFlow did much of the work during conversion, including automatically specifying which variables to save. To convert from a concrete function, a custom inference function was created. After ensuring it was compatible with TensorFlow Lite operations, the function was converted.

3.2.3 Custom Inference Function

The custom inference function *infer_tflite* was used to convert the model to TensorFlow Lite. The function inputs and outputs are described in Section 2.8, Function Converted Description. To feed in a sentence to the algorithm, convert the sentence to IDs using the same vocabulary used to train the model.

3.2.4 Major Changes to Running a Normal Inference

Modification to word embedding: The function *tflite_call* was added to get the required word embeddings. This function is called twice: once during encoding and again during decoding. One problem we encountered is also described in TensorFlow issue #42410. The workaround described on the GitHub issue was implemented to fix errors that occurred while trying to make the TensorFlow Lite conversion work.

Different decoding: For decoding, we have to assume the batch size is 1. There is an object from the TensorFlow library named *TensorArray* that is not fully supported on TensorFlow. The object is only supported when the dynamic size flag is set to flag, and the size is specified when creating the *TensorArray*. During decoding, the *TensorArray* created is replaced with a static size *TensorArray*. We assume that the batch size is 1, which is a safe assumption since we are running on Android.

Inputs and outputs: This inference function only takes in an integer array as IDs and outputs IDs as integers, as explained in Section 2.8, Function Converted Description. Since the return of this function is only IDs, this function stops early and only returns the IDs when decoding. The regular inference takes in a dictionary that includes IDs and other required variables.

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	DEVCOM ARL FCDD RLD DCI TECH LIB
1 (PDF)	DEVCOM ARL FCDD RLC IB S LAROCCA