RXCWNN - A python program for the decoding of morse signals with a neural network

Ties Bos - PA0MBO

May 15th, 2020

Abstract

A python program was developed for the decoding of radio amateur morse signals with a recurrent convolutional neural netwerk. The network was trained with a large set of audio files generated with varying signal to noise ratios, sending speeds from 10 to 40 words per minute, QSB, random small timing variations during the sending of the characters and with impulse noise. The decoding results are on a par with **CWSkimmer** and the program can be run under Windows 10 on a somewhat older CPU, the JETSON NANO and the raspberry pi 4

Keywords: Morse decoding - Hamradio - Recurrent convolutional neural networks

1 INTRODUCTION

Since my retirement in 2001 I did not touch neural network programming until recently I bought a JETSON NANO. This cheap little processor board houses a lot of computing power that enables it to process video streams coming in from a Raspberry Pi camera for the recognition of common objects in real time. The software that comes with it is mostly written in Python and working through its examples I noticed the similarity of the problems the software has to deal with in processing handwritten texts with the problems of decoding morse signals, especially the synchronization problem in noisy conditions.

Having experimented with various implementations of the Goertzel [1] algorithm and the ESP32 processor [2], [3] and being somewhat

disappointed by the results in comparison to the "golden standard" CWSkimmer, I decided to try to find out if decoding morse signals with a neural network instead of using Bayesian statistics would produce useful results. Searching the web for "decoding morse code with neural networks" I learned that quite a lot of work has already been done on this subject, i.e. by Fabián Tamás László [4] and Mauri A Niininen (AG1LE) [5].

Seeing the spectacular results that can be obtained by using a combination of convolutional and bidirectional **LTSTM** (long term short term memory) layers and connectionist temporal classification **CTC** of the outputs of the network in optical character recognition [6] I decided to try that deep learning network architecture in my own experiments. The capacity of this network should be sufficient for recognizing the morse code because for amateur use this code has only 48 classes (the 26 characters of the alphabet, the 10 numbers, some puntuation marks and the prosigns) versus the upper and lower case characters (52) in reference [6].

2 EXPERIMENTS

2.1 Training an test sets

Obtaining sufficient annotated training and test material from the bands seemed too cumbersome for me, so I used the fine program **generate_wav_samples.py** developed by Fabián Lászlói [4] for this purpose. This program generates morse code audio with various amounts and types of noise at various sending speeds and stores the results in wav-files with their annotation in companion text files. The duration of the signal and the amount of textblocks are given to the program in command line arguments. All its other parameters, i.e. the sending speed in wpm, noise, etc. are stated in the program itself and used as given in the distributed file (10 - 40 wpm). However some changes were made to the morse code character set; the prosigns and some punctuation marks like the question mark were added in the **config.py** file. Moreover the program was modified to take the duration of the signal in milliseconds instead of seconds. 100,000 wav-files were generated each of a duration of 8,192 msec for a total amount of somewhat more than 225 hours of morse code. In the training of the network 90,000 files were used as training set whereas 10,000 files were used for validation.

The training was performed with a slightly modified program taken from [6]. The input layer dimension was modified to use 65536 samples of audio directly from the generated wav-files. Training with batches

of 64 files was performed for 100 epochs and took about 30 hours to complete on a system with an AMD Ryzen Threadripper 3970X 32-Core processor and 128 GB of memory. After the 100 epochs training loss as well as validation loss showed slight variations over the epochs but they did not decrease anymore. During training the best performing (lowest validation loss) network weights were saved to file. After finishing the best performing network was tested with a separately generated test set consisting of 100 wav-files from which the accuracy of the network was calculated.

2.2 Setup for real time decoding of band signals

For real time decoding of audio signals from a transceiver the audio samples are acquired from a soundcard in the computer with its mike input connected to the transceivers audio output. The python program **rxcwnn.py** uses routines from the python package **sounddevice** for this purpose. Furthermore this program uses routines from the module **queue** to keep up a constant flow of audio data without interruptions. The processing is performed in chunks of 65536 data points from a buffer that is filled by shifting in a quarter as much audio data at 4096 msec intervals. The audio sample frequency is 8000 Hz. The problem of synchronization at the character level is solved by "stitching" two consecutive overlapping text strings that are obtained from the 65536 samples buffer at 16384 sample intervals.

2.3 Installation of the software

The software consists of 3 files: (1) a python program **rxcwnn.py**, (2) the definition of the deep neural network in a json file **neuralnetcw.json** and (3) the network weight factors **best_modelfast.hdf5**, packed together in a zip-file. To run the program one needs python 3.7 together with packages **keras**, **tensorflow**, **sounddevice** and **curses** installed. The easiest way to provide this environment for Windows 10 is to use **anaconda**

2.3.1 Windows 10

1. Download **Miniconda3** from **https://docs.conda.io/en/latest/miniconda.html** and run the latest version of the program. Tick the item "Just me" and use the standard installation directory. N.B. No ticks at the advanced issues.

2. From the windows start menu open an Miniconda prompt and type the command:
   mkdir morse
3. Run the command:
   cd morse
4. Run the command to build the environment tfenv
   conda create -n tfenv tensorflow
5. Activate this environment with the command:
   conda activate tfenv
6. In this environment run the command:
   pip install –upgrade –user pip
7. In this environment install sounddevice with:
   pip install sounddevice
8. In this environment install keras with:
   pip install keras
9. In this environment install curses with:
   pip install windows-curses
10. Download the file **rxcwnn.zip** from [9] and unzip it in the **morse** subdirectory.

2.3.2 Linux

There are quite a number of linux distributions, each requiring its own peculiar installation setup. Here the installation process is only described for two development boards, i.e. the Jetson Nano and the Raspberry Pi 4.

Jetson Nano

1. Flash a 32 GB micro-SD-card with the JetPack 4.4 image downloaded from [7] using installation the procedure given by NVIDIA.
2. Boot the Jetson nano developer kit with connected keyboard and HDMI monitor and this micro SD-card inserted
3. Follow the first use instructions from [7] to set username and password
4. Login and start a terminal
5. Follow the installation procedure given in [8] for a tensorflow version smaller than 2. (version 2 does not work).
6. Run the command:
   sudo apt-get install build-essential libssl-dev libffi-dev python-dev

7. Run the command:
   sudo pip3 install sounddevice
8. Run the command:
   sudo pip3 install keras
9. Run the command:
   mkdir morse
10. Download the file **rxcwnn.zip** from [9] and unzip it in this **morse** directory.

Raspberry Pi 4

1. Prepare a 32 GB micro-SD-card image for your raspberry Pi 4 [10] and boot it.
2. Open a terminal and install **virtualenv** with:
   sudo pip3 install virtualenv
3. Make a new directory **morse** with the command:
   mkdir morse
4. Change to this new directory with the command:
   cd morse
5. In this directory make a new environment with the command:
   python3 -m virtualenv -p python3 cw_env
6. Activate the envireonment with the command:
   source cw_env/bin/activate
7. Now install tensorflow version 2 in this environment as described in [11], but leave out the –user option en do not use sudo if front of the pip3 commands.
8. Install libffi-dev with the command:
   sudo apt-get install libffi-dev
9. Install module sounddevice with:
   pip install sounddevice
10. Install keras with:
    pip install keras
11. Download the file **rxcwnn.zip** from [9] and unzip it in this **morse** directory.

3 USE OF THE PROGRAM

1. For Jetson Nano and Raspberry Pi 4 you have to connect a separate USB sound input device, because they do not have a builtin soundcard.

2. Connect the audio output of your transceiver to the microphone input of the USB sound device.
3. set the default sound input of the used sound device to this audio input.
4. run the command:
   python3 rxcwnn.py
5. tune your trx to a cw transmission

4 RESULTS

The accuracy of the trained network was established by calculating the text belonging to the audio of 100 wav-files generated by the program **generate_wav_files.py** and comparing it with their annotation. The error rate was 6%. Furthermore a number of tests was run with the program **rxcwnn.py** installed as described above on an older ACER INTEL Core(TM)2 Quad CPU Q8200 @2.33 GHz with 4GB of memory. The line-in input of its soundcard was connected to the phone output of a FLEX3000 transceiver with an external hardware audio cable. The FLEX3K was controlled by POWERSDR v2.7.2 running on the same computer. Following contest signals, the callsigns, reports and exchanges were compared with the traffic shown by **CWSkimmer** (connected by VAC) running in parallel. Generally there was a very good agreement. When both programs disagreed the offending callsigns were looked up in qrz.com and more often than not **rxcwnn** was right, although the reverse also occurred sometimes.

In the same way ragchew qso's were followed. Here it was noticed that at keying speeds below 15 wpm **rxcwnn** makes more errors than **CWSkimmer**

5 DISCUSSION

The lesser achievements of **rxcwnn** at low words per minute rates was attributed to the fixed length in time of the audio files used for the training: these contain less characters when the keying speed is low. Training of a network with a smaller range in keying speed, i.e 10 - 20 wpm instead of 10 - 40 wpm however did not improve the situation. Presumably this problem is a synchronization issue: the audio files used for the training start and end with a no signal condition whereas the audio buffer used for the decoding can sometimes start or end within the middle of a morse character. With low keying speed this will occur more often. The "stitching" procedure used in **rxcwnn**

should solve this problem, but further experiments, i.e. using larger audio chunks should clarify if this really is the case. Extending the audio buffer however is bound by the increase in processing time and the increase in the delay between hearing the morse code and seeing the characters on the screen.

6 ACKNOWLEDGEMENTS

The author is very much indebted to Dr.ir.J.Bos (PE1PUZ) for running the training calculations on his powerful Ryzen system.

References

[1] Wikipedia, http://en.wikipedia.org/wiki/goertzel_algortihm.

[2] Loftur E. J'onasson TF3LJ, https://sites.google.com/site/lofturj/cwreceive.

[3] Hjalmar Skovholm Hansen OZ1JHM, skowholm.com/cwdecoder.

[4] Fábian Tamás László, https://github.com/netom/morsenet.

[5] AG1LE, http://ag1le.blogspot.com/2019/02/training-computer-to-listen-and-decode.html.

[6] https://github.com/theailearner/a-crnn-model-for-text-recognition-in-keras.

[7] NVIDIA, https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

[8] NVIDIA, https://docs.nvidia.com/deeplearning/frameworks/install-tf-jetson-platform.

[9] Ties Bos PA0MBO, http://www.pa0mbo.nl/ties/public-html/hamradio/rxmorsenn.

[10] raspberrypi.org, https://www.raspberrypi.org/downloads.

[11] raspberrypi.org, https://github.com/pinto0309/tensorflow-bin.