Wind turbine drive train fault detection with Convolutional Neural Networks

Helder Daniel

Departamento de Engenharia Electrónica e Informática
Faculdade de Ciências e Tecnologia
Universidade do Algarve

October 10, 2019

Abstract

Renewable energy sources have strategic importance in the energy production field. Along with hydro power wind power is one of the major renewable sources. The production of wind powered energy is done by huge devices commonly addressed as wind turbines. These devices are usually deployed at high places where winds are strong, but with limited access, making maintenance operation expensive. Unexpected failures and maintenance is a major cost source. Effective failure diagnosis and prediction can reduce this cost. Components expected to fail soon can be changed at scheduled maintenance dates. In (Baltazar, Daniel, Oliveira, and Li [2018]) it is presented a review of neurocomputing based approaches to wind turbines fault diagnosis and prognosis, grouped by components. One major component is the drivetrain, that converts mechanical energy from the blades in electrical energy. In this paper it is shown that a Convolution Neural Network (CNN), trained with data from mechanical and electrical sensors, can be effectively hyperparametrized to identify drivetrain failures with a rate of near 100%.

1 Introduction

The drivetrain is a core component of a wind turbine. It is where the wind mechanical energy collected by the blades is converted in electrical energy. It is also under constant stress. Identifying failures and their causes gives a huge advantage in reducing maintenance cost, since components about to fail can be changed at scheduled maintenance dates. In this paper it is proposed a

CNN hyper parametrized to effectively identify a class of common failures in the drive train.

The data set used was collected assembling in or near a wind turbine model drivetrain a 3D accelerometer sensors to measure vibration, along with sound and current sensors.

Figure 1: Wind turbine model

Figure 2: Drivetrain detail

The possibility of using a CNN to identify drivetrain failures was suggest by its effective use in time-series analyses (Himmetoglu [2017]) and some Human Activity Recognition (HAR) projects (healthDataScience [2019], Shahnawaz [2017], Kwapisz and Moore [2010]).
Collecting data from 3D accelerometers, such as the ones included in cell phones, some projects aim to recognize human activity such as: Standing,Walking, Jogging, climbing Upstairs or Downstairs,etc..

The development platform used was TensorFlow(Abadi et al. [2015]), along with Python with Keras, SKlearn and Numpy packages. Tensorflow has the benefit that can be ran on CPU, GPUs and also TPUs, Google's processors optimized for tensor operation, essentially systolic arrays.
Python was selected since it has the most comprehensive interface to TensorFlow libs, although TensorFlow itself is mainly developed in C and Cuda.
Further TPUs have freely, however limited, access at Google's Coolab, using Python Jupyter notebooks.

The next sections presents the data set characterization, CNN hyper parameters tuning, results and discussion.

Failure code	Fault Type	Fault location	Fault severity
00	No fault		
01	Pitting	Ring gear	Slight
02			Moderate
03	Broken teeth		Partial broken teeth
04			Full broken teeth
05	Double pitting		Moderate
06	Pitting + broken teeth		Moderate
07	wire cut groove		0.5mm wide 0.3mm deep
08	Pitting	Sun gear	Slight
09			Moderate
10	Broken teeth		Partial broken teeth
11			Full broken teeth
12	Double pitting		Moderate
13	Pitting + broken teeth		Moderate
14	wire cut groove		0.5mm wide 0.3mm deep
15	Pitting	Planetary gear	Slight
16			Moderate
17	Double pitting		Moderate
18	Pitting + broken teeth		Moderate
19	double planet wheels broken teeth		Moderate
20	Teeth root crack		broad 0.3mm deep 0.3mm
21			broad 0.3mm deep 0.6mm
22			broad 0.3mm deep 1 mm
23			broad 0.5mm deep 0.3mm
24			broad 1 mm deep 0.3mm
25			broad 1.5mm deep 0.3mm
26	Broken teeth		1/10 teeth missing
27			2/10 teeth missing
28			3/10 teeth missing
29			4/10 teeth missing
30			5/10 teeth missing
31			6/10 teeth missing
32			7/10 teeth missing
33			8/10 teeth missing
34			9/10 teeth missing
35			Fullcut

Table 1: Failure classes

2 Data set acquisition

The data set was acquired inducing the 36 failures classes defined in table 1:

The failures were induced in the drivetrain of a fan model RCVA-3000, figures 1 and 2, and acquiring vibration signals, acoustic emission signals, sound signals and current signals. The acoustic emission sensor and the acceleration sensors are assembled at the input and output shaft, the sound sensor is close to the gearbox.
The sensors, sensor channels and sample frequency are defined in table 2:

Sensor description	Channel	Sample frequency (KHz)
Current phase 1, 100mV/A	000	100
Current phase 2, 100mV/A	001	100
Current phase 3, 100mV/A	002	100
Accelerometer vibration X, 100mV/g	003	100
Accelerometer vibration Y, 100mV/g	004	100
Accelerometer vibration Z, 100mV/g	005	100
Acoustic emission, ??	006	1000
Sound/acoustic field, ?? db/mV	007	100

Table 2: Sensors and channels

For each of the 36 class of failures a different experiment was performed with 3 different loads and at 4 rotating speed. Each setup was repeated 10 times, giving a total of 4320 samples in the data set:

```
#samples = failures x loads x rotations x repetitions

4320 = 36 x 3 x 4 x 10
```

Each sample has 20 seconds of data registered from the above defined 8 sensor channels. A total of 86400x8 = 691200 seconds of recording, about 800 GBytes of data.

Load code	Description	Value (Ohms)
1	Small	0 - 0.1
2	Intermediate	1
3	Maximum	10.5

Table 3: Loads

The loads used in the experiments are defined in table 3.

The rotation speeds used in the experiments are defined in table 4. The variable rotation speed was set to: 2 seconds at 20Hz, then about 6 secs to reach 50Hz, hold at 50Hz by 4 seconds, then about 6 seconds to decelerate to 20Hz, and 2 seconds at 20 Hz, for a total of 20 seconds.

Rotation code	Description	rotation speed(HZ)
00	Variable	20 - 50
30	Constant	30
40	Constant	40
50	Constant	50

Table 4: Rotation speeds

3 CNN hyper parameters tuning

3.1 Data set characterization

The tuning of the CNN hyper parameters was done with a sub set of the full set:

- faults = [0,1,3,8,10,15,30]
- loads = [1]
- freqs = [50]
- reps = [1]
- chans = [003, 004, 005]

Using a subset allows faster training, which will be a time saver when tuning parameters. On the other hand using just data from just one of the experience repetitions to train the network, allows to use data from other repetition(s) to test the network efficiency.

Since channels 003, 004 and 005 are sampled with a frequency of 100KHz, each 20 second sample has 2 000 000 data points. The sample will be split in smaller sizes to find a minimum value that carries information about the failure class.

3.2 Training setup

For each class 80% of the samples are randomly chosen to train the network and 20% for evaluation of the trained network.
By default the dataset is shuffled only once before all experiments and all runs.
For Adam and SGD algorithms, at each epoch the train data set is shuffled.
Model weights are shuffled at each run.
The 7 fault classes are renumbered [0, 6]and onehot encoded.

3.3 CNN

The CNN used was a 1D-CNN. 2D-CNNs were also tested but the preliminary results were not promising. The investigation of a suitable 2D-CNN is not disregarded, but at this point it was used only 1D-CNNs, with the generic layer layout, from presentation layer at the top:

CNN1:

```
Conv1D(filters, kernelSize, activation=relu, size(input))
MaxPoling1D()
Conv1D(filters, kernelSize, activation=relu, size(input)/10)
MaxPoling1D()
Conv1D(filters, kernelSize, activation=relu, size(input)/100)
MaxPoling1D()
Dropout(DropoutRatio)
Flatten()
Dense(activation=softmax, size(failureClasses))
```

An optimized variant is below:

CNN2:

```
Conv1D(filters, kernelSize, activation=relu, size(input))
MaxPoling1D(pool, strides)
Conv1D(filters*2, kernelSize, activation=relu, size(input)/10)
MaxPoling1D(pool,strides)
Conv1D(filters*4, kernelSize, activation=relu, size(input)/100)
MaxPoling1D(pool,strides)
Conv1D(filters*2, kernelSize, activation=relu, size(input)/100)
MaxPoling1D(pool, strides)
Dropout(DropoutRatio)
Flatten()
Dense(activation=softmax, size(failureClasses))
```

The main parameters considered was:

```
input size =
filters =
kernelSize =
pool =
strides =
DropOutRatio = 0.5
failureClasses = 7
```

3.4 Initial setup

Initially the data set was transformed by the quantile transform to have an uniform distribution, before fed to the CNN.
In this setup, the 20 second sample is split in smaller samples with sizes: 6250,12500,25000 and 50000.

Parameter	CNN1.1	CNN1.2	CNN1.3	CNN1.4
Filters	10x100x10	10x10x10		
kernelSize	10			
pool	10	10	5	10
strides	5	5	5	10

Table 5: CNN1 variations

Several configurations of CNN1 was trained with optimizer Adam(mse), for 100 epochs, with samples sizes of: 6250, 12500, 25000 and 50000, table 5.

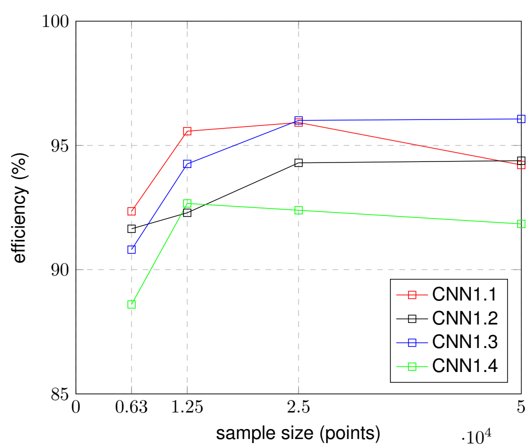

Figure 3: Efficiency for preprocessed data

The best network reaches best efficiency, around 96%, for large samples. In the general case, for small samples the efficiency drops drastically. Analyzing large samples have a performance overhead in training and propagation.

3.5 Optimized setup

It was found that processing the raw data gives better results, even with small data sets.
This 20 second sample is split in smaller 2 ms samples. For each of the 7 failure classes considered there are 1000 samples, so the data set used in this setup has 7000 samples. Each sample has 2000 points representing 2 ms. Since we are considering data from 3 channels, each sample has actually 3x2000 points. So the training set has 7x800 samples and the evaluation set has 7x200 samples.

It was used the CNN2 described above, with the best setup found, so far:

```
filters = 32x64x128x64
kernelSize = 20
pool = 5
strides = 5
DropOutRatio = 0.5
failureClasses = 7
```

Training was performed for 200 epochs with a constant learnig rate (LR) of 0.001, achieving an efficinecy around 99%. Then for more 300 epochs, the learning rate was bounced around values: [0.001-0.0005-0.00025-0.000125], to avoid settling in local minima. This achieved in some case 100% efficiency.

Figure shows the efficiency for 25 experiments. In each experiment the data set is shuffled, the network is trained as defined above, and evaluated. In 19 of the 25 experiments the efficiency in detecting this 7 failure classes was 100%. In the remaining experiences the efficiency was above 99.92%.

4 Conclusion

The drivetrain is a core component of a wind turbine, that is under constant stress. Predicting a time window for its failure can avoid unscheduled maintenance costs, changing the component in the next scheduled maintenance date. In this paper was presented the tuning of hyper parameters of a CNN to effectively identify a set of classes of common drivetrain failures. It was shown

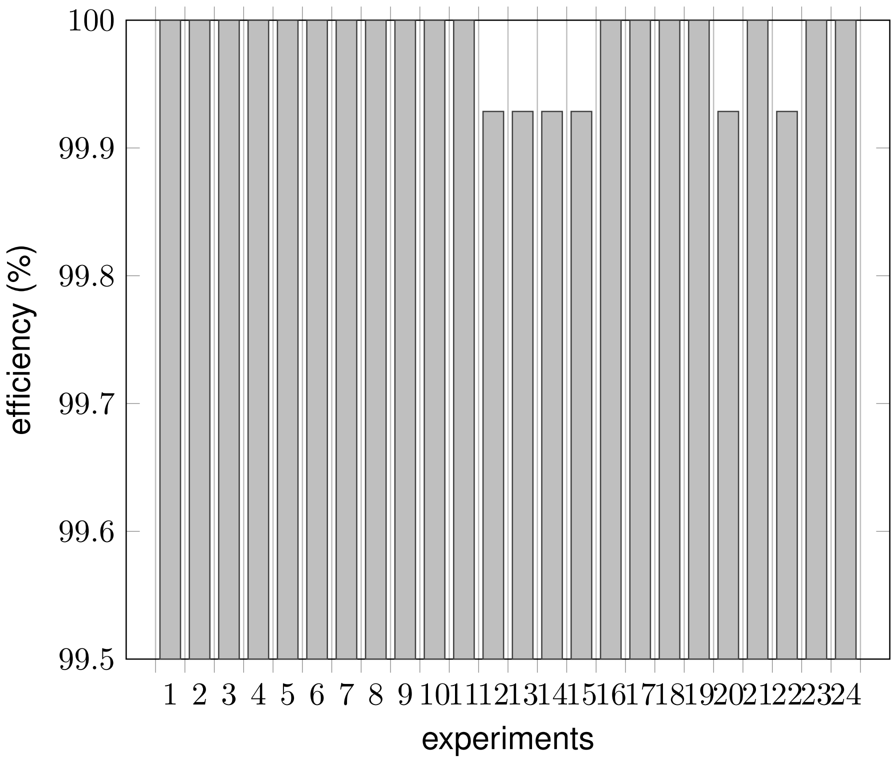

Figure 4: Efficiency for raw data

that, for the considered failure classes, this CNN has an almost 100% identification rate.

Following this project's future road map, this CNN will be further developed to identify a wider range of common failure classes, trained with a richer data set. Also will be addressed the effectiveness of another machine learning solutions such as: Random Forests, Auto-Encoders, Long Short Term Memory and Generative Adversarial Network.

References

Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Sérgio Baltazar, Helder Daniel, José Valente de Oliveira, and Chuan Li. A review on neurocomputing based wind turbines fault diagnosis and prognosis. In P Ding, C Li, RV Sanchez, and S Yang, editors, *Proceedings of Prognostics and System Health Management Conference (PHM-Chongqing)*, pages 437–443, October 2018. doi: 10.1109/PHM-Chongqing.2018.00081.

healthDataScience. Human activity recognition (har), 2019. URL https://github.com/healthDataScience/deep-learning-HAR. [Online; accessed 28-May-2019].

Burak Himmetoglu. Time series classification with tensorflow, 2017. URL https://burakhimmetoglu.com/2017/08/22/time-series-classification-with-tensorflow/. [Online; accessed 28-May-2019].

Jennifer R. Kwapisz and Gary M. Weiss Samuel A. Moore. Activity recognition using cell phone accelerometers. *ACM SIGKDD Explorations Newsletter*, 12 (2):74–82, 2010. doi: 10.1145/1964897.1964918. URL https://dl.acm.org/doi/10.1145/1964897.1964918.

Muhammad Shahnawaz. Human activity recognition using cnn in keras, 2017. URL https://github.com/Shahnawax/HAR-CNN-Keras. [Online; accessed 19-June-2019].