University of Applied Sciences Berlin

Report

AI in der Robotik

by

Tim Oelkers
s0568352

Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

Supervisor: **Patrick Baumann, M.Sc.**

XX. Month, 2021

Abstract

AI in der Robotik

This should be a single paragraph of not more than 500 words, which concisely summarizes the entire proposal.

Contents

Abstract **i**

Contents **ii**

1 Introduction **1**
1.1 Motivation 1
1.2 Project Objective 1
1.3 Proceeding and Structure of the Work 2

2 Fundamentals **3**
2.1 Robot Operating System 3
2.2 Neural Networks 3
2.2.1 Convolutional Neural Networks 4
2.2.2 Fully Connected Neural Network 4

3 Conception **5**
3.1 Dataset 5
3.2 Model 5
3.3 Audio Capture 5
3.4 Evaluation Architecture 6

4 Implementation **7**
4.1 Model Training 7
4.2 Evaluation Environment 8

5 Evaluation **9**

6 Conclusion **10**
6.1 Summary 10

6.2 Future Work 10

List of Figures **I**

List of Tables **II**

Source Code Content **III**

A About Appendices **IV**

Chapter 1

Introduction

1.1 Motivation

Voice recognition systems gained immense popularity over the last years with almost every big tech company (Amazon, Google, Apple, Microsoft, etc.) deveoping their own systems. While these systems require

// require huge amount of ressources and expertise to develop // spoken language is very complex and intend can be given is many different ways // for example in the structure of sentences, the use of differenent synonyms or different pronounciations // human voice is incredible diverse and even one speaker may change it over span of the day // ressource intensive to run as well, all of them using webservices and the power of the cloud for evaluation of the spoken language // // since its so resource intensive, all of these listen for specific commands ("Hey Google", "Alexa", etc.) that signal the start of a more complex // expression. These command // // cut the dependency to data collecting companies and run a more simpler command recoginition locally // no complex language analysys, but still able to detect different voices

1.2 Project Objective

This projects aims to develop a human command recognition system using machine learning. The scope should be limited to simple and short commands and its usefulness should be evaluated in a realistic, imperfect environment. This includes testing the model with a real microphone in a noisy environment with multiple speakers. It should be also evaluated for multiple people with different voice tones. The system should be able to run on a middle class computer. While the learning performance is important and will be evaluated shortly, the main evaluation will be focused on its accuracy.

1.3 Proceeding and Structure of the Work

TODO

Chapter 2

Fundamentals

2.1 Robot Operating System

ROS (Robot Operating System) is an open-source meta operating system. It runs on top of Unix based systems and provides abstraction over process management, low-level communication and hardware. It also provides tools and libraries to develop applications in the field of robotics and distributed systems. ROS is build on the concept of creating a (possibly distributed) peer-to-peer network of nodes. A node is a single, modular process that should perform a dedicated task, which is implemented using C++ (roscpp) or Python (rospy). All nodes register themselfs at the ROS master, which serves as a nameserver and processer manager to the rest of the network. Nodes communicate which each other using ROS messages, which are data structures that define the type of data that is passed. The communication is usually based on ROS topics, which implement an asynchronous publish-subscribe pattern so that nodes are truly decoupled. If direct, synchronous communication with an immediate response is needed, ROS services can be used.

2.2 Neural Networks

Neural networks are a emulation of a human brain with inter-connected nodes (neurons) with weighted edges (synapses). The learning part comes from the process of altering each weight after each evaluation to better map inputs to their expected outputs. This is called supervised learning and inputs with known outputs have to be available for that. The type of neural network used here is called a fully connected, feed forward network which consists of one input layer, one hidden layer and one output layer. Each node of one layer connects to each node on the following layer. During the forward

pass, the data passes through the network while the weights are applied. To achieve non-linear mappings, an activation function is applied to inputs between each layer. After that, the error to the expected value is calculated with a predefined loss function and the weights are updated accoring to a predefined optimizer in a process called back-propagation. The process is repeated for a certain number of epochs so that the output gets closer and closer to the expected value. After the training, the model can be used on unknown input.

2.2.1 Convolutional Neural Networks

2.2.2 Fully Connected Neural Network

Chapter 3

Conception

3.1 Dataset

For this projects, the "SpeechCommands"/cite warden2018speech dataset will be used. This dataset was created by Pete Warden, working for Google, with the intent of creating a fundamental set of simple commands spoken in English with a lot of variance. The data was gather publicly and thus contains audio from different speakers with normal day-to-day audio quality and possibly even background noise. Even though the data comes from a public source, its correctness was checked manually and with various alghorhithms to remove too quiet or interrupted words. In total, the data contains 105.829 audio files (3.4 GB) for 35 words spoken by 2.618 different people. All audio files are exacly one seconds long and the data is alligned so that words all start and end roughly at the same time. The audio comes in the WAVE format with 16bit single channel (mono) values, sampled at a rate of 16kHz. The dataset also contains audio files containing common background noise, that can be used for evaluation. The dataset is split into multiple subsets for training, testing and validation, specified by specific file lists in the same folder.

3.2 Model

The model will be a ...

3.3 Audio Capture

As the quality should be evaluated with a real microphone, the audio$_{c}aptureROSpackageisusedforthis$

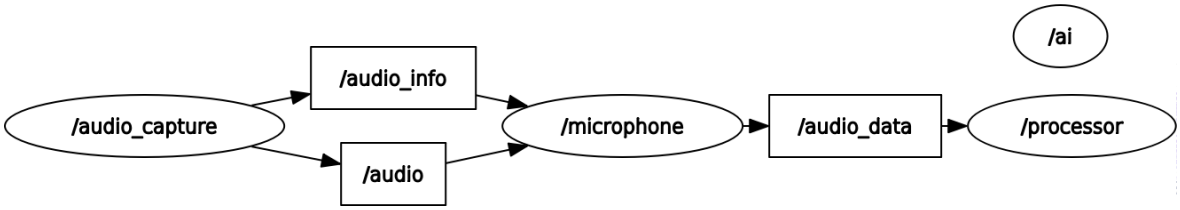

Figure 3.1: ROS architecture

3.4 Evaluation Architecture

The evaluation architecture will be developed as a ROS application as it helps with development and models a potential real world use case as close as possible. The following image shows a graph of the architecture and how the nodes interact with each other.

As described in section 3.3, the raw audio data and metadata get sent to two topics that are subscribed by the microphone node. It is responsible for recording the data when a specific user input has been made. Once another input was made, the data has to be aggregated and sent to the audio$_{d}atatopic.Itissubscribedbytheprocessornode, whichhandlesthenec$

Chapter 4

Implementation

4.1 Model Training

The implementation for the model training is based on a tutorial on command recognition by the pytorch team[**tutorial**]. The first step is to create a training and testing subsets of the whole dataset, that got downloaded and saved locally if needed. The testing and validation filenames are stored in specific files, everything not listed in these files is supposed to be for training. The iteration and downloading of these files is done by the SPEECHCOMMANDS class provided by torchaudio. The dataset class also has some methods to get the labels, the amount of files and the sample rate of the subset. As these methods require iteration over all files and thus take some time to compute, the results were memoized by implementing a file based cache with the help of pythons object serialization (pickle). This speeds up consecutive model trainings. One datapoint then contains the waveform, the sample rate, the label, a speaker id and the label number. As I am only able to do the training on the CPU, the audio gets resampled to 8kHz to improve training time. We then use a standard torch DataLoader and specify a batch count, the number of workers and that the data should be shuffled after each epoch so that the order is not affecting the model in any way. A so called collate function is also defined that pads the waveforms with zeroes so that all have the same length, encodes the label as the index in the label array and throws away the rest of the unnwanted data in the dataset (like the sample rate). The model is implemented according to section 3.2 with a input size equal to the number of labels. We use the Adam optimizer with a relatively high learning rate of 0.01 and a small weight decay of 0.0001 as in the original paper **??**. The loss function used is called NLLLoss, which is often used in 1D classification problems. The model is then trained and tested for 5 epochs. After that, the model itself and metadata about it (labels and sample rate)

are saved to disk, together with a graph showing the accuracy over each epoch.

4.2 Evaluation Environment

All nodes are started via the following launch file. It describes where the nodes are located and how they should be configured. For the audio$_{c}apturenode, wehavetomatchthesetupfromthe$

```
1  <launch>
2    <node name="ai" pkg="ros_ai_report" type="ai.py" output="screen"/>
3    <node name="microphone" pkg="ros_ai_report" type="microphone.py"
         output="screen"/>
4    <node name="processor" pkg="ros_ai_report" type="processor.py"
         output="screen"/>
5    <node name="audio_capture" pkg="audio_capture" type="audio_capture"
         output="screen">
6      <param name="bitrate" value="256"/>
7      <param name="channels" value="1"/>
8      <param name="sample_rate" value="16000"/>
9      <param name="format" value="wave"/>
10     <param name="sink" value="appsink"/>
11     <param name="sample_format" value="S16LE"/>
12   </node>
13 </launch>
```

The microphone node then stores the received data in memory when a specific key is pressed and joins it together when pressed again. It also calculates the sample width by extracting the information from the sample format. The processor node receives the aggregated data and writes it to file using the wave module of the python standard library. This is sadly needed, as torchaudio does not support loading of file-like objects and only expects a string as a file path. The needed metadata like sample rate and sample width are received from the microphone node. The ai node loads the model and its metadata from disk. When a request comes in, it loads the file previously created, resamples the audio to the models sample rate and predicts the command using the model. As the model applies a softmax function at the end, the data represents the logarithm of the probability of being a certain command. That means, that the highest number is the predicted output and by applying the exponential function, the probability is calculated. As the highest element is just the index of the command, the actual command has to be found in the label array stored in the models metadata. The entire process is repeated until the user terminates the program.

Chapter 5

Evaluation

First of all, the accuracy of the model was tested agaist the testing subset provided by the used dataset.

Chapter 6

Conclusion

* length is really important for training, original data is aligned and even padded with zeroes so that all have the same length, that is not the case for the data coming from the microphone, * the model took the waveform at specifc times into consideration, not fully based on the overall structure of the wave

6.1 Summary

TODO

6.2 Future Work

TODO

List of Figures

3.1 ROS architecture 6

List of Tables

Source Code Content

Appendix A

About Appendices

Appendices are optional and should only be used if necessary.